Université de Bordeaux I
Laboratoire Bordelais de Recherche en Informatique

HABILITATION À DIRIGER DES RECHERCHES

au titre de l'école doctorale de Mathématiques et Informatique de Bordeaux

par Monsieur Olivier BEAUMONT

Nouvelles méthodes pour l'ordonnancement sur plates-formes hétérogènes

à présenter devant la commission d'examen formée (pour l'instant) de :

Monsieur Richard Brent,	Rapporteur,	Professeur à Oxford
Monsieur Paul Feautrier,	Rapporteur,	Professeur à l'ENS Lyon
Madame Claire Kenyon,	Rapporteur,	Professeur à l'Ecole Polytechnique
Monsieur Denis Trystram,	Rapporteur,	Professeur à l'INPG
Madame Claire Hanen,	Membre,	Professeur à l'Université Paris X
Monsieur André Raspaud	Membre,	Professeur à l'Université Bordeaux I
Monsieur Jean Roman,	Membre,	Professeur à l'ENSEIRB

Table des matières

Introduction

Un vieux proverbe de l'Ouest assure qu'il est plus facile de conduire un attelage avec deux bœufs qu'avec mille poulets. L'objet du présent document est d'analyser l'impact du remplacement des mille poulets par 300 oies, 400 poulets et 300 poussins. De façon non surprenante, le proverbe de l'Ouest reste vérifié dans ce contexte (pour s'en convaincre, considérer le cas de petites oies et de gros poussins). En fait, comme nous le verrons, la situation est en général plus difficile. Pour pallier cette difficulté, une solution élégante consiste à convaincre le lecteur que le temps nécessaire au rassemblement de toute cette volaille justifie son emploi exclusif à de grandes tâches et que de telles tâches peuvent en général être découpées en sous-tâches nécessitant un minimum de coordination entre les gallinacés.

Les travaux présentés dans cette habilitation correspondent à des recherches effectuées depuis mon arrivée à Bordeaux, en septembre 2001. En termes d'encadrement doctoral, ils correspondent à la fin de la thèse d'Arnaud Legrand (soutenue en décembre 2003) et au début de celle de Loris Marchal (commencée en septembre 2003). Ces travaux ont l'avantage, par rapport à une synthèse plus vaste de mes activités de recherche, de présenter une grande cohérence. Tous portent en effet sur la recherche d'ordonnancements périodiques de débit optimal, pour la distribution de tâches indépendantes (Chapitre 4) et la réalisation de communications collectives (Chapitre 5).

La grande majorité des résultats présentés dans ce document a déjà été publiée, et de nombreuses références permettront au lecteur de trouver les preuves non reproduites ici. Toutefois, pour éviter que cette habilitation soit un simple et creux catalogue de résultats, j'ai décidé de profiter de l'opportunité de la rédaction (puisque c'en est une) pour passer du temps à justifier et à formaliser ces travaux de recherche.

Ainsi, dans les Chapitres 1 et 2, le lecteur trouvera une (tentative de) justification originale des modèles et des objectifs. Le Chapitre 3 présente deux résultats (Théorèmes 3.1 et 3.4) qui n'ont pas été publiés (même si une ébauche du Théorème 3.4 peut être trouvée dans la thèse d'Arnaud Legrand). Il s'agit d'une partie technique assez lourde, mais qui permet ensuite de présenter de façon originale et unifiée les résultats des Chapitres 4 et 5.

Chapitre 1

Plates-formes considérées

1.1 Caractéristiques générales

Nous avons annoncé dans le titre que le lecteur trouverait dans cet opuscule des méthodes nouvelles pour l'ordonnancement sur des plates-formes modernes. Il convient donc d'extraire les caractéristiques de ces plates-formes dont nous devons tenir compte pour réaliser un ordonnancement dont le comportement observé ne soit pas trop éloigné du comportement prédit.

La contrepartie de la modernité des plates-formes que nous allons considérer est sans aucun doute leur complexité. Nul ne peut prévoir avec une grande précision le temps de la communication d'un fichier de 1GO entre deux machines d'une grille de calcul situées de part et d'autre de l'atlantique. En effet, le message sera découpé en petits paquets, qui ne suivront pas tous le même chemin au gré des contentions sur les liens qui sont dues à l'activité des autres utilisateurs et aux succès actuels des films. Il n'est pas raisonnable d'espérer une modélisation très fine du comportement des différents routeurs du réseau et de la sociologie des différents utilisateurs. Toutefois, un modèle qui considérerait que le temps de ce transfert de données ne dépend pas du couple de machines entre lesquels il s'opère pourrait être accusé de simplisme.

Le premier travail consiste donc à extraire un ensemble pertinent de paramètres. Ces paramètres doivent être représentatifs du comportement de la plate-forme, mais ils doivent être suffisamment peu nombreux pour permettre la conception d'algorithmes. Dans toute la suite, un des partis pris de ce travail est de refuser la complexité. Comme le lecteur s'en rendra vite compte, nous sommes prêts à supprimer des paramètres et à changer l'objectif, afin de trouver des algorithmes polynomiaux optimaux, ou au moins, des algorithmes d'approximation garantis.

Il semble que les deux caractéristiques incontournables (puisque constitutives) des plates-formes de calcul de type grille sont la dispersion géographique et l'hétérogénéité. L'hétérogénéité est présente tout à la fois au niveau des liens de communication et des unités de calcul. La dispersion, de son côté, entraîne la dynamicité des ressources, puisqu'on ne peut pas supposer toujours disposer, en particulier, de ressources de communication longue distance dédiées à une application particulière.

Dans la suite de ce chapitre, nous présentons le modèle que nous allons utiliser pour les ressources de calcul (Paragraphe 1.2) et de communication (Paragraphe 1.3). Nous aborderons dans ce même paragraphe l'influence relative des différentes activités (influence d'une communication sur la durée d'une autre, influence d'une communication sur la durée d'un calcul).

Notre but est de concevoir des algorithmes dont le comportement en pratique soit satisfaisant. La validation par l'expérience et la comparaison de différents algorithmes n'est pas un problème simple, comme nous le verrons au Paragraphe 1.4. Nous tenterons de convaincre le lecteur que la simulation est la seule issue pour comparer et évaluer des algorithmes d'ordonnancement sur de telles plates-formes. La simulation doit alors nécessairement s'appuyer sur un modèle précis (plus précis que celui utilisé pour effectuer l'ordonnancement, pour que la validation ne soit pas qu'une version affaiblie d'un vérificateur des théorèmes démontrés). Nous

aborderons donc également au Paragraphe 1.4 les problèmes de simulation et de découverte de la topologie de plates-formes réelles.

1.2 Modélisation des ressources de calcul

1.2.1 Un problème difficile

Le temps d'exécution d'une application sur une machine donnée est extrêmement difficile à prévoir, même pour des routines très simples, comme les BLAS (Basic Linear Algebra Subroutine) et même si on a accès à une description précise de l'architecture cible. Par exemple, ATLAS [100] (Automatically Tuned Linear Algebra Software) est un outil dont l'objet est de générer automatiquement du code pour des primitives d'algèbre linéaire, l'évolution très rapide des processeurs rendant très difficile et très coûteuse la conception "à la main" de telles routines. ATLAS ne s'appuie pas sur une description du matériel utilisé dans la machine cible, mais se fonde sur l'exécution réelle de la primitive considérée, en faisant varier certains paramètres (taille des blocs). La séquence des exécutions choisies, qui représente le problème le plus délicat de cette approche s'appuie sur des principes très généraux d'architecture (présence de plusieurs niveaux de cache) et sur le résultat des exécutions précédentes afin d'instancier les paramètres du modèle. Le code produit automatiquement avec cette technique est alors différent, selon la taille du problème considéré.

Cet exemple nous montre bien que même pour des routines très simples et dont le comportement est très bien connu, l'évaluation du temps d'exécution ne peut être obtenu que par l'exécution effective de la routine sur l'architecture cible.

Si on souhaite obtenir une modélisation très précise du temps d'exécution d'une application sur une machine, la seule solution est donc d'exécuter cette application, pour différentes tailles du problème. On obtient alors une courbe relativement complexe, comme l'ont montré, par exemple, les travaux de Lastovetsky [65]. Cette approche a l'inconvénient d'être extrêmement coûteuse, puisqu'elle nécessite l'exécution, pour chaque application et chaque architecture cible, d'une multitude d'expériences. Elle nécessite en outre de stocker une quantité considérable d'informations pour pouvoir prendre les décisions d'ordonnancement. De plus, elle n'est pas adaptée à des plates-formes non dédiées, puisqu'un modèle d'une telle précision devrait tenir compte à la fois de la charge extérieure et de sa nature (utilisation du cache, du disque, du réseau...).

1.2.2 Modélisation des applications

Nous avons vu qu'il était difficile de concevoir un modèle réaliste pour le temps de calcul d'une application sur un processeur autrement qu'en exécutant effectivement cette application. D'un autre côté, nous avons également vu qu'une modélisation précise nécessite *a priori* un stockage de données (et donc un coût d'ordonnancement) excessif. Dans le but d'obtenir une modélisation aussi réaliste que possible mais avec relativement peu de paramètres, pour permettre la conception d'algorithmes d'ordonnancement, nous considérerons dans la suite que le problème à résoudre est donné sous la forme d'un graphe de tâches. Cette hypothèse a pour conséquence que la granularité des tâches a été fixée *a priori*. Par exemple, si on souhaite réaliser un produit de matrices, on supposera le problème découpé en un certain nombre de produits élémentaires de blocs. Cette hypothèse a plusieurs avantages.

1. Nous n'avons pas à choisir la taille du problème qui sera associé à chaque processeur, mais simplement les tâches qui lui seront allouées.
2. Pour chaque processeur, il suffit donc de connaître, pour chaque tâche, le temps nécessaire à l'exécution de celle-ci.

Cette approche présente quelques inconvénients du point de vue du réalisme de la modélisation. Tout d'abord, contrairement au cas de processeurs homogènes, le grain de calcul utilisé n'est certainement pas optimal pour tous les processeurs (par exemple, il est connu que la taille de bloc optimale pour les routines BLAS dépend fortement de l'architecture). Ensuite, on peut également perdre quelques effets de cache. En effet, si deux tâches sont allouées sur le même processeur et qu'elles partagent certaines données, le temps nécessaire pour

l'exécution des deux tâches sera exactement la somme des temps d'exécution des tâches, comme si le cache avait été vidé. Contrairement au premier problème soulevé, il est à noter que celui ci est également vrai dans le cas homogène.

1.2.3 Modélisation du coût des tâches de calcul

Dans le cadre des grilles de calcul, notre modèle doit nécessairement prendre en compte le fait que les processeurs ont des vitesses de calculs différentes. De la même façon, il semble déraisonnable de considérer qu'ils disposent tous des mêmes bibliothèques. Les applications s'exécutant sur des grilles de calcul nécessitent en outre l'accès à des ressources distantes. Du fait des problèmes d'authenfication, de création de processus, de nombreux appels à des intergiciels (middlewares) sont nécessaires. Ces appels génèrent un coût fixe, qui dépend uniquement du processeur sur lequel s'exécutera la tâche, mais pas de la nature de la tâche et de sa taille (il ne s'agit pas ici de modéliser le coût du transfert de données, mais uniquement le surcoût induit par l'intergiciel).

Dans la suite, nous modéliserons la durée de l'exécution de n tâches de type T_k sur le processeur P_i par

$$L_i^{\text{calc}} + nw_{i,k},$$

où L_i^{calc} représente la latence induite par l'intergiciel, et $w_{i,k}$ le temps d'exécution de la tâche T_k sur le processeur P_i. L'utilisation de $w_{i,k}$ plutôt que $t_i v_k$ (où t_i représenterait le temps pour traiter une tâche de calcul unitaire et v_k le volume de calcul associé à la tâche T_k) est justifiée par le fait que différents processeurs peuvent posséder des bibliothèques de calcul différentes, des tailles de caches différentes.

1.3 Modélisation des ressources de communication

1.3.1 Un problème très difficile

La modélisation des coûts de communication est une tâche très ardue. En effet, si nous considérons une application sur une grille de calcul, nous sommes confrontés aux problèmes suivants

1. On ne peut pas considérer que le réseau est dédié à une application particulière, et il n'est pas possible de prévoir la charge extérieure sur le réseau.
2. On ne peut pas considérer qu'on connaît avec une grande précision la topologie du réseau. Nous aborderons le problème de la découverte de topologie dans les réseaux au Paragraphe 1.4, mais il s'agit de rechercher une topologie "équivalente" au niveau applicatif, plutôt que de déterminer l'emplacement et le comportement de chacun des routeurs, "hubs" et "switches" du réseau.
3. Même en supposant une connaissance exacte de la topologie physique du réseau et de la charge externe, la simple simulation du trafic des paquets dans le réseau en tenant compte des protocoles est une tâche extrêmement longue, comme le montrent les travaux menés avec NS[75], pour lequel le temps de simulation d'un transfert de données est plus grand, de plusieurs ordres de grandeur, que le transfert lui même. Dans tous les cas, une telle précision n'est pas compatible avec la conception d'algorithmes d'ordonnancement efficaces.

Par ailleurs, on peut remarquer que contrairement aux processeurs, les liens de communication dans les machines parallèles actuelles sont hétérogènes. Ces machines ont en effet une organisation hiérarchique et le temps de communication entre deux processeurs est variable, selon que les processeurs sont ou non intégrés à la même unité. Toutefois, les méthodes étudiées dans ce document ne sont sans doute pas les plus adaptées dans le contexte des machines parallèles hiérarchiques. D'autres travaux ont été menés[50, 51], où les coûts de communication peuvent prendre un ensemble discret de valeurs (typiquement 0 ou 1), qui rendent mieux compte de la nature de l'hétérogénéité dans de telles plates-formes.

1.3.2 Modélisation des coûts de communication

Nous présentons ici la modélisation des coûts de communication utilisée tout au long de ce document. Dans le Paragraphe 1.3.3, nous décrirons d'autres modèles. Nous discuterons dans le Paragraphe 1.3.4 des différences et des points communs entre ces modèles, ainsi que des limitations de celui que nous proposons.

Dans la suite, nous modéliserons la durée de l'exécution de la communication d'un message de taille n entre les processeurs P_i et P_j par

$$L_{i,j}^{\text{comm}} + nc_{i,j},$$

où $L_{i,j}^{\text{comm}}$ représente la latence du lien de communication entre P_i et P_j et $c_{i,j}$ représente l'inverse de sa bande passante. Comme pour la modélisation des coûts de calcul, le temps de communication se décompose en une partie fixe ($L_{i,j}^{\text{comm}}$) et une partie proportionnelle à la taille du message ($nc_{i,j}$). La latence de communication, plus encore que la latence de calcul, ne peut être négligée. En effet, l'évolution des technologies de réseau semble indiquer que la bande passante croit très régulièrement au fil des années. Par contre, la latence est elle bornée par la vitesse de la lumière, de sorte que même pour des tailles de messages importantes, les deux termes sont du même ordre de grandeur. Par exemple Casanova [32] cite l'exemple de TeraGrid [96]. Sur cette plate-forme la bande passante entre le SDSC (San Diego Supercomputer Center) et le NCSA (National Center for Supercomputing Applications) est de 40 GO/sec alors que la latence du lien est de l'ordre de 100ms, de sorte que pour un transfert de 1GO, les deux quantités sont comparables. A priori, les liens sont supposés asymétriques, à la fois au niveau de la latence et de la bande passante (i.e. $c_{i,j} \neq c_{j,i}$ et $L_{i,j}^{\text{comm}} \neq L_{j,i}^{\text{comm}}$). Contrairement à la modélisation des coûts de calcul, il est également intéressant de noter que la partie linéaire du temps de transfert ne dépend que du volume de données échangé et non pas de sa nature.

Quand on considère la modélisation des communications, il est également nécessaire de préciser les interactions entre les différentes activités au niveau d'un nœud de la plate-forme, c'est à dire l'influence d'une communication avec une autre au niveau de l'émetteur et du récepteur, ainsi que l'influence d'une communication sur la puissance de calcul de l'émetteur et du récepteur. Dans la suite, nous utiliserons le modèle suivant pour les interactions, appelé dans la suite modèle 1-port bidirectionnel sans contention.

1. Quand P_i envoie un message à P_j, P_i ne peut émettre un nouveau message avant la fin de la communication (i.e. après une durée $L_{i,j}^{\text{comm}} + nc_{i,j}$) et P_j ne peut pas recevoir d'autre message avant la fin de la communication (modèle 1-port).
2. Par contre, pendant ce temps, P_i peut recevoir un message d'un autre troisième processeur et P_j peut émettre un message vers un troisième processeur (modèle bidirectionnel).
3. Pendant la durée de la communication, P_i et P_j peuvent continuer leur activité de calcul sans être ralentis pendant la communication (modèle sans contention).

Ce modèle a été introduit par Bhat et al. [24, 25] pour des messages de taille fixée. Les auteurs justifient son utilisation par le fait que "current hardware and software do not easily enable multiple messages to be transmitted simultaneously".

Nous reviendrons sur ce point et nous discuterons des défauts et qualités de ce modèle dans le Paragraphe 1.3.4. Nous utiliserons également, essentiellement quand nous analyserons la diffusion (broadcast) de messages, le modèle 1-port unidirectionnel sans contention. Dans ce modèle, contrairement au modèle 1-port bidirectionnel, les émissions et les réceptions ne sont pas découplées. Plus précisément, quand P_i envoie un message à P_j, P_i et P_j ne peuvent ni émettre ni recevoir de nouveau message avant la fin de la communication. Le principal défaut de ce modèle par rapport au modèle bidirectionnel est qu'il est beaucoup plus difficile au niveau algorithmique, comme nous verrons au Paragraphe 5.3.

1.3.3 D'autres modèles

Nous présentons ici quelques modèles qui ont été présentés dans la littérature, parce que chacun d'entre eux capture des caractéristiques importantes des réseaux de communication.

1.3.3.1 Modèle de Bar-Noy et al. [7]

Dans ce modèle, le temps d'une communication entre P_i et P_j d'un message de taille n est donné par $\lambda_{i,j}$ et peut être décomposé en trois étapes, qui se recouvrent éventuellement (il est à noter que dans [7] le modèle est présenté pour une taille de message fixée, et donc les parties constantes et linéaires en la taille du message ne sont pas explicitées, l'auteur prend donc toute la responsabilité de cette extension, qui respecte, espérons le, la pensée des auteurs).

1. Pendant la première étape, le processeur émetteur P_i est bloqué pendant un temps fixe s_i, qui ne dépend ni de la taille du message ni de son destinataire. Cette partie correspond à l'émission du message sur le réseau. À l'issue de cette première étape, P_i peut commencer une nouvelle émission.
2. Pendant la dernière étape, le processeur destinataire P_j est bloqué pendant un temps fixe r_j, qui ne dépend ni de la taille du message ni de son émetteur. Cette partie correspond à la réception du message depuis le réseau. Avant cette dernière étape, P_j peut être engagé dans d'autres réceptions.
3. La ressource de communication entre P_i et P_j est elle occupée entre le début de la première étape et la fin de la dernière étape. Ce temps $\lambda_{i,j}$ est une fonction affine de la taille n du message à envoyer.

Le principal apport de ce modèle est de considérer que la durée pendant laquelle le processeur émetteur (resp. destinataire) est bloqué ne dépend que de lui seul (et de son interface réseau), mais pas du destinataire (resp. émetteur). Ce modèle correspond à une vision assez intuitive de l'échange de message : l'émetteur, une fois qu'il a envoyé le message sur le réseau, est libre d'en envoyer un autre. On peut toutefois faire deux remarques sur ce modèle

1. Au niveau physique, l'émetteur reçoit pendant toute la durée de la communication des messages d'acquittement envoyés par le destinataire pour assurer le contrôle de flux. Il est donc en partie ralenti par ces messages (en réception) et il peut être amené à re-émettre certains paquets. Toutefois, au niveau physique, ce modèle colle sans doute mieux à la réalité (au moins pour des machines parallèles) que celui proposé au paragraphe précédent.
2. Au niveau logiciel, la durée pendant laquelle l'émetteur est bloqué dépend fortement de l'implantation de la bibliothèque de communication. En cas d'envoi synchrone (par exemple avec **MPI_Send** et **MPI_Recv** [76]), les deux processeurs sont effectivement bloqués (c'est la sémantique des opérations synchrones) pendant la durée complète de l'échange de message. Nous verrons également au Paragraphe 1.3.4 que pour des tailles de message relativement grandes, le comportement de **MPI** avec des envois asynchrones est en fait identique.

Cappello et al. [85, 1] ont proposé un modèle assez proche, mais dans le cas d'un réseau complet et homogène. On retrouve dans ce modèle les trois phases présentées dans le modèle précédent. À la fin de la première étape, les auteurs proposent d'ajouter une étape (courte) pendant laquelle les deux processeurs émetteur et récepteur sont bloqués, et qui correspond à une phase de "négociation/réservation" de la route.

1.3.3.2 Modèle de Casanova [32]

Dans [32], le problème du partage de bande passante est analysé au niveau des liens longues distances. Quand deux communications point à point partagent un même segment du réseau (ethernet ou internet par exemple), alors la bande passante est partagée entre ces deux communications. Il est généralement admis que dans ce cas, la bande passante est approximativement partagée de façon équilibrée entre les différentes communications. Ainsi, si une bande passante B est disponible sur ce segment, alors chacune des n communications utilisant ce segment se voit allouer une bande passante de $\frac{B}{n}$.

Sur des liens longues distances (dans [32], des communications entre l'université de San Diego et différentes universités aux États Unis et en Europe sont analysées), il est absolument nécessaire de tenir compte de la charge externe du réseau. Toutefois, si b représente la bande passante mesurée lors de l'échange de données, n représente le nombre d'échanges réalisés par le transfert qu'on souhaite analyser, B la bande passante totale du lien et N la charge externe (en nombre de connections), alors le partage de bande passante s'écrit

$$B = b(n + N),$$

ce qui permet de déterminer B et N, en mesurant b pour différentes valeurs de n.

Cette modélisation du partage de la bande passante est très séduisante pour les liens longue distance partagés, et les algorithmes présentés dans ce document bénéficieraient sans doute d'une modélisation plus fine de ce type de liens. En fait, comme nous le verrons dans le Paragraphe 1.4, la méthode utilisée pour découvrir la topologie du réseau nous "sauve" partiellement. Pour déterminer si un transfert entre P_i et P_j et un autre entre P_k et P_l partagent un lien, nous considérons le retard induit sur la communication entre P_i et P_j par une communication simultanée entre P_k et P_l. Si la valeur de N sur le lien partagé est grande, alors nous observerons une interférence minime, et nous en déduirons que les deux communications sont indépendantes et seront donc modélisées par deux liens indépendants. Au contraire, si la valeur de N est faible, l'interférence sera importante et dans la modélisation, un lien sera partagé entre les deux paires de processeurs (les communications seront alors séquentialisées sur ce lien). Si on observe les courbes présentées dans [32], la première situation (communications totalement indépendantes) correspond au cas des communications très longue distance entre l'université de San Diego, l'université de Virginie et l'université de Delft. La deuxième (communications séquentialisées) correspond à la situation observée à l'intérieur du campus de San Diego. Par contre, les phénomènes mitigés observés sur des liens de longueur intermédiaire, entre San Diego, l'université de Washington et l'université de Santa Barbara ne seront pas correctement exprimés par notre modèle.

Un autre modèle de partage de bande passante a été proposé par Jeannot et Wagner [62]. Dans le cas où plusieurs communications partagent un même lien (backbone), ils proposent un modèle dans lequel chaque communication se voit allouer une fraction de bande passante fixe, mais dans lequel le nombre de communications utilisant le lien est limité.

1.3.4 Comparaison des différents modèles

Comme nous l'avons vu, la difficulté de la modélisation des communications a conduit à une multitude de modèles qui capturent tous une ou quelques caractéristiques d'un ou quelques types de réseaux. Mon but n'est pas ici de faire un palmarès de ces modèles (on doit pouvoir montrer sans trop de difficulté que quelque soit le modèle, il existe une situation où il est plus performant que les autres), mais plutôt d'extraire les caractéristiques communes à tous.

Dans tous les modèles que nous avons présentés, les liens sont supposés hétérogènes et le temps de communication est composé d'une partie fixe et d'une partie proportionnelle à la taille du message. De la même façon, tous les modèles s'accordent sur le fait qu'il n'est pas possible de réaliser un grand nombre de communications (partageant le point de départ ou le point d'arrivée) en même temps. Par contre, la durée pendant laquelle l'émetteur ou le récepteur est bloqué et ne peut être engagé dans une communication varie d'un modèle à l'autre.

Le modèle que nous avons proposé et que nous allons utiliser dans le reste de ce document prend en compte l'hétérogénéité, les latences, l'asymétrie des liens et une forme de séquentialisation des envois et des réceptions. Il a l'avantage (du point de vue de l'algorithmicien) de faire intervenir relativement peu de paramètres et celui (du point de vue de l'implantation) que ses paramètres sont relativement faciles à instancier et à observer.

Du point de vue du recouvrement des communications, il s'agit d'un modèle extrême, dans lequel les envois (resp. les emissions) sont séquentialisés. Cette modélisation correspond toutefois, comme nous l'avons vu, à une implantation à partir de primitives synchrones, ce qui est un contexte assez général. De plus, même si des bibliothèques (multi-threadées) de communication asynchrones existent, les auteurs de [24, 25] affirment que toutes les opérations "are eventually serialized by the single hardware port to the network". Récemment, des expériences menées par Saif et Parashar [86] semblent confirmer cette affirmation. En effet, ils démontrent qu'expérimentalement, des envois asynchrones de messages sont en fait réalisés en séquence, dès lors que la taille du message devient importante (de l'ordre de quelques MOctets). Ces résultats s'appuient sur des tests réalisés avec deux implantations populaires de MPI, MPICH [53] sur des clusters Linux et la bibliothèque MPI fournie par IBM pour la machine SP2.

Une autre limitation de notre modèle est que nous supposons un recouvrement complet des calculs et des communications. Cette hypothèse est sans doute trop optimiste. Lors d'un transfert de données, le processeur est sans doute ralenti par le fait que le bus mémoire est en partie utilisé pour le transfert entre la mémoire et la carte réseau. Des études récentes, menés par Barbara Kreaseck [64] montrent que (pour des communications implantées avec Java en tout cas), ce phénomène ne peut être négligé dans le cas de gros transferts de données, particulièrement quand les sites concernés sont éloignés.

1.4 Comment évaluer les algorithmes d'ordonnancement ?

1.4.1 Optimalité et expérimentation directe

Afin d'évaluer et de comparer les algorithmes d'ordonnancement développés dans ce document, on ne peut pas se contenter des preuves d'optimalité asymptotiques et des algorithmes d'approximation garantis qui y sont présentés. En effet, comme nous l'avons vu, tous ces algorithmes s'appuient sur un modèle extrêmement simplifié des ressources de calcul et de communication, même si nous avons essayé dans les paragraphes précédents de convaincre le lecteur que nous avons extrait un ensemble pertinent de paramètres. Dans la suite seront présentés plusieurs algorithmes asymptotiquement optimaux (pour notre modèle, bien sûr). Il est intéressant de vérifier sur des exemples pratiques que l'optimalité asymptotique n'est pas trop longue à venir. De la même façon, nous avons négligé bien d'autres paramètres, comme la taille de la mémoire, celles des caches. Nous avons également affirmé que les performances des liens et des ressources de calcul pouvaient varier au cours de l'exécution. Toutefois, ce document manque singulièrement de toute preuve de stabilité ou de robustesse des algorithmes considérés. Sur ce point là, les algorithmes présentés souffrent donc à la fois de la modélisation et de fondements théoriques. Récemment, des travaux ont été menés pour définir une notion de robustesse d'un ordonnancement et pour proposer des algorithmes robustes (pour des modèles plus simples que celui utilisé dans ce document) [55]. Une perspective intéressante serait d'analyser nos algorithmes et de les adapter pour prouver (au moins dans des contextes simples) leur robustesse vis à vis de petites perturbations.

Si on souhaite disposer d'une plate-forme réaliste d'expérimentation en présence d'une charge extérieure sur le réseau et sur les processeurs, la stratégie la plus naturelle semble être l'exécution de l'application sur une plate-forme réelle. Toutefois, nous pensons qu'il ne s'agit pas de la bonne stratégie pour évaluer et comparer les algorithmes d'ordonnancement pour les raisons suivantes.

1. Tout d'abord, l'expérimentation directe nécessite de disposer d'un code réel, et de le déployer en utilisant les intergiciels comme Globus [47] sur une grille de calcul. Il n'est pas qu'anecdotique d'affirmer que ce travail est extrêmement long et que, s'il se justifie pour des applications pratiques, il peut être disproportionné pour la simple évaluation d'algorithmes d'ordonnancement.
2. Ensuite, la durée des tests est elle-même un obstacle sur de telles plates-formes, pour des problèmes de taille importante comme ceux qui sont considérés dans ce document. Elle peut être un obstacle à une validation sur un grand nombre de tests et un grand nombre de plates-formes.
3. Enfin, nous pensons que l'expérience directe est à rejeter pour une raison plus fondamentale. L'instabilité de ces plates-formes rend la comparaison d'heuristiques impossibles. Il n'est en effet pas possible de comparer deux heuristiques d'ordonnancement dans des conditions de charge des liens du réseau et des processeurs identiques.

1.4.2 La simulation

Nous avons vu que les algorithmes d'ordonnancement développés dans ce document doivent être validés au delà des preuves formelles (qui ne sont valables que pour le modèle choisi) mais que l'expérience directe n'est pas la bonne solution, particulièrement pour comparer plusieurs algorithmes. La simulation[1] permet

[1] les travaux présentés ici font en partie référence à la thèse d'Arnaud Legrand. Ceci étant dit, je ne peux me flatter de la moindre paternité sur la partie simulation. Tout le mérite de cette paternité revient à Henri Casanova, de l'université de San Diego

de contourner les obstacles liés à l'expérimentation directe, aux trois conditions suivantes

1. Le code à développer pour la simulation doit être assez proche de la formulation "papier" pour limiter les coûts de développement.
2. Le temps de simulation doit être sensiblement plus rapide que le temps d'exécution réel, pour permettre des tests sur un grand nombre de plates-formes et de paramètres. En outre, il est à noter que la simulation est naturellement reproductible ce qui permet de comparer objectivement deux heuristiques d'ordonnancement, en leur offrant les mêmes conditions de simulation.
3. Enfin, le modèle utilisé pour la simulation doit être sensiblement plus précis et plus réaliste que celui utilisé pour la conception des heuristiques.

1.4.2.1 Simulation du réseau

L'outil de simulation pour les réseaux le plus employé est NS [75]. Il permet aux concepteurs de protocoles d'analyser les transferts sur le réseau au niveau des paquets pour comprendre le fonctionnement *in situ* des protocoles utilisés et en concevoir d'autres. La contrepartie au réalisme de la simulation proposée par NS est sa lenteur, puisque la simulation est plus lente, de plusieurs ordres de grandeur, que le transfert lui même. Pour effectuer nos simulations, nous avons donc besoin d'une modélisation plus grossière, mais, comme nous le verrons, NS peut être utilisé pour valider à son tour un simulateur plus grossier.

1.4.2.2 Simulation d'applications

De nombreux simulateurs d'applications ont été développés. Certains sont dédiés à la simulation sur une machine parallèle (LAPSE [38] pour la machine Paragon [44]). D'autres utilisent le repliement d'une grille de calcul sur une grappe de processeurs pour émuler une application (mais il faut alors disposer du code de cette application), comme Microgrid [94] ou Panda [36]. L'hétérogénéité et l'instabilité (au niveau des communications) sont alors simulées en ralentissant artificiellement les liens de la grappe, pour représenter le comportement des liens longue distance.

1.4.2.3 SIMGRID : simulation à base de traces

SIMGRID [31, 33, 72] est un logiciel de simulation à base de traces développé à l'université de San Diego et à l'ENS Lyon par Henri Casanova et Arnaud Legrand. Le principe de la simulation avec SIMGRID est (grossièrement) le suivant :

- Chaque ressource (lien de communication ou unité de calcul) est associée à une latence (en secondes) et à un débit (en Flop/s pour les ressources de calcul et en O/s pour les ressources de communications).
- On dispose pour chaque ressource de traces (obtenues par mesure sur des plates-formes réelles, comme nous le verrons au Paragraphe 1.4.3) indiquant le débit (en Flop/s ou en O/s) disponible à chaque instant sur la ressource et la latence observée à chaque instant.
- Chaque tâche (de calcul ou de communication) est associée à un volume (en Flop ou en O)

Ainsi, si la tâche T de volume de calcul (ou de communication) S est allouée à l'instant t_0 à la ressource R dont la latence est donnée par la fonction $L(t)$ et le débit par la fonction $B(t)$, alors la tâche sera terminée (si aucune autre tâche n'est allouée à cette ressource avant la fin de l'opération) à l'instant T vérifiant

$$\int_{t_0+L(t_0)}^{T} B(t)dt = S.$$

Pour les ressources de communication, Casanova et Marchal [34] ont développé un modèle de partage de bande passante qui a été validé à l'aide du logiciel de simulation de réseau NS [75] évoqué au paragraphe précédent.

SIMGRID nous permet de disposer d'un outil simulation des algorithmes développés dans ce document qui répond aux objectifs de facilité d'utilisation (la description des tâches de calcul et de communication est très proche de la modélisation utilisée pour réaliser l'ordonnancement), de rapidité (le code n'est pas émulé,

le simulateur utilise simplement les traces pour déterminer la date de fin de chaque tâche) et de précision (les modèles utilisés pour la simulation de la partie réseau, par exemple, sont beaucoup plus précis que ceux utilisés pour l'ordonnancement et ont été validés sur des simulateurs de réseau).

1.4.3 Observation des plates-formes et récupération des traces

Nous avons vu au paragraphe précédent que SIMGRID était un outil parfait pour réaliser des simulations afin d'évaluer et de comparer des heuristiques. SIMGRID s'appuie sur la description d'une plate-forme et sur des traces associées à cette plate-forme. Il est donc essentiel de posséder la description de plates-formes réelles et de pouvoir récupérer des traces sur celles ci.

Il existe de nombreux outils pour générer aléatoirement des plates-formes de calcul et des réseaux d'interconnection [39, 99], et de nombreux travaux ont été menés pour établir des modèles de la topologie d'internet [46, 39, 30, 73], en essayant d'intégrer sa dimension hiérarchique. Pour les besoins de la simulation avec SIMGRID, ces modèles souffrent de l'absence de modélisation de la charge extérieure appliquée aux liens de communication et aux ressources de calcul.

Pour répondre à ce problème, la technique la plus simple consiste à obtenir une description de la plate-forme, puis ensuite à observer et à enregistrer son comportement réel.

1.4.3.1 Découverte automatique de plates-formes

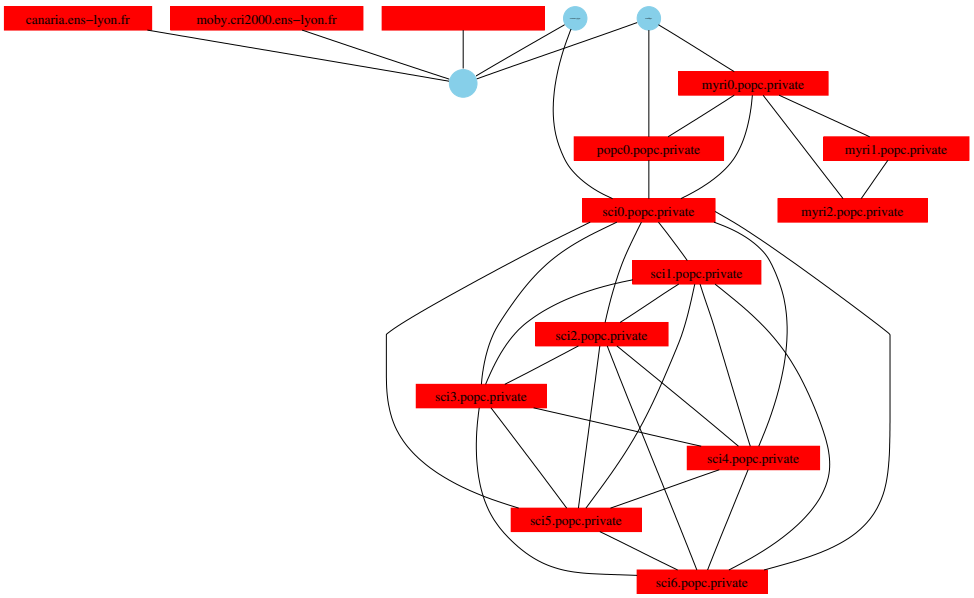
Quelques outils ont été développés pour découvrir la topologie effective des plates-formes de calculs. L'objectif de tels outils n'est pas de fournir une vision détaillée et exacte du réseau, où tous les hubs, switches et routeurs seraient présents, mais plutôt de fournir une topologie "équivalente", qui permette de prédire les communications qui peuvent être effectuées de manière indépendante et celles qui partagent des ressources de communication.

ENV [92, 91] a été développé par Gary Shao. Cet outil a été développé pour évaluer des applications de type maître-esclaves : un nœud joue donc un rôle particulier dans la plate-forme et le graphe d'interconnection est recherché sous la forme d'un arbre (en fait, le graphe obtenu peut ne pas être tout à fait un arbre, mais nous n'insisterons pas sur ce point). Le principe (grossièrement expliqué) de ENV est de raffiner successivement la vision de la plate-forme en utilisant des outils comme traceroute [60] et des tests d'interférence entre communications point à point, de manière à construire un graphe de la plate-forme observée, tel que deux communications point à point sont indépendantes si et seulement si les chemins entre les paires de points dans l'arbre n'ont pas d'arête commune. Les résultats obtenus par ENV sur les plates-formes de Lyon et Strasbourg sont décrits respectivement aux figures 1.1(a) et 1.1(b).

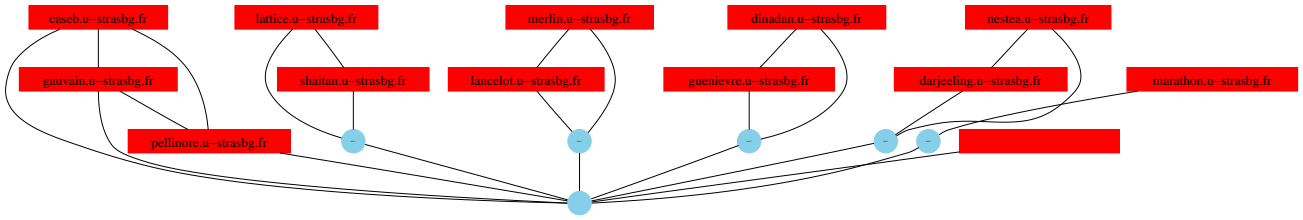
La principale limitation de ENV est liée au fait que cet outil a été construit pour évaluer des applications de type maître-esclaves et permet donc d'obtenir uniquement des plates-formes sous la forme d'arbres. Récemment, Arnaud Legrand, Frédéric Mazoit et Martin Quinson ont développé AlNEM [68], qui ne souffre pas de cette limitation. Malheureusement, les techniques employées par AlNEM sont beaucoup plus raffinées, et son temps d'exécution, même pour des plates-formes de taille raisonnable, reste encore rédibitoire.

1.4.3.2 Récupération des traces

Une fois obtenue la vision de la plate-forme, on peut récupérer effectivement la trace de l'activité des différentes ressources de calcul et de communication, en utilisant par exemple NWS [102, 101]. NWS s'exécute en tâche de fond et mesure régulièrement la latence et le débit des différentes ressources de calcul et de communication, en exécutant de petits bouts de code pour évaluer les ressources de calcul et en réalisant des échanges de petits messages pour évaluer les communications point à point. Pour évaluer les caractéristiques des liens de communication, il est important de ne pas générer d'interférences entre les tests, ce qui est fait en utilisant l'analyse réalisée préalablement par ENV (on pourra consulter [69] pour une description plus précise du protocole) .



(a) Plate-forme de Lyon



(b) Plate-forme de Strasbourg

FIG. 1.1 – Plates-formes de Lyon et Strasbourg découverte par ENV. Les ronds représentent des routeurs et les rectangles des machines

1.5 Conclusion

Nous avons vu que la modélisation de plates-formes hétérogènes, distribuées et non dédiées est une tâche extrêmement complexe. Comme notre objectif est de concevoir des algorithmes d'ordonnancement ayant de bonnes propriétés théoriques, et non quelques heuristiques de listes de plus, nous devons nous appuyer sur un modèle simple, mais aussi réaliste que possible. Nous avons donc décidé de modéliser les applications par des graphes de tâches, de modéliser le temps de calcul de n tâches de type T_k sur un processeur P_i par

$$L_i^{\text{calc}} + nw_{i,k}$$

et de modéliser le temps d'une communication de taille n entre les processeurs P_i et P_j par

$$L_{i,j}^{\text{comm}} + nc_{i,j}.$$

Reste que la modélisation utilisée pour concevoir les algorithmes d'ordonnancement est trop simpliste et que les résultats d'optimalité et les garanties prouvées avec ce modèle doivent être validées par l'expérience. La dynamicité de ces plates-formes rend toutefois la comparaison des heuristiques d'ordonnancement impossible. Pour réaliser cette tâche nous devons donc réaliser des simulations. SIMGRID est un simulateur à base de traces qui permet une évaluation facile et rapide d'heuristiques d'ordonnancement. Pour obtenir la description des plates-formes et les traces de l'activité des différentes ressources, nous utiliserons respectivement ENV [92, 91] et NWS [102, 101]. Malheureusement, il est à noter que nous ne disposons pas de beaucoup de plates-formes (avec les traces d'activités observées des processeurs et des liens), en dehors de celles représentées à la Figure 1.1. Pour les tests présentés dans les prochains paragraphes, nous nous appuierons généralement, pour disposer d'une large classe de plate-forme, sur des outils de génération de topologies "réalistes" comme Tiers [39], ce qui est regrettable.

Chapitre 2

Ordonnancement sur plates-formes hétérogènes

2.1 Minimisation du temps de complétion

L'objectif le plus classiquement utilisé en ordonnancement, quand on considère une seule application, est la minimisation du temps de complétion de cette application. Il est bien connu que ce problème est notoirement difficile, et conduit à de nombreux résultats de NP-Complétude, même dans le cadre de modèles de communications beaucoup plus simple que ceux que nous avons vu au Chapitre 1, Paragraphe 1.3. Notre but n'est pas ici de faire un panorama des résultats de complexité des problèmes d'ordonnancement, le lecteur intéressé pourra consulter la page web [29] pour un tableau complet de ces résultats ou le chapitre sur l'ordonnancement dans [57].

Dans cette partie, nous allons considérer quelques problèmes élémentaires, qui peuvent être résolus par des algorithmes polynomiaux dans le cadre d'un modèle simple (processeurs et liens homogènes, modèle multi-port pour les communications), pour déterminer, sur ces exemples, si le passage au modèle 1-port avec des processeurs et des liens hétérogènes complique la résolution du problème d'ordonnancement. L'étude présentée ici ne prétend pas du tout à l'exhaustivité. Notre but est simplement d'illustrer la difficulté des problèmes d'ordonnancement sous le modèle défini dans le Chapitre 1, en montrant que l'hétérogénéité des liens (Paragraphe 2.1.1) ou l'hétérogénéité des processeurs (Paragraphe 2.1.2) peut conduire à des résultats de NP-Complétude pour des problèmes élémentaires. Il restera alors à convaincre le lecteur que l'utilisation d'autres métriques est nécessaire.

De manière à éliminer les difficultés provenant de l'application elle-même, nous ne considérerons dans ce paragraphe que le cas de la distribution de tâches indépendantes toutes identiques, selon le paradigme maître-esclaves (dans lequel un processeur particulier, le maître, possède initialement toutes les données et distribue les tâches de calcul et les données qui leur sont attachées aux autres processeurs de la plate-forme, les esclaves).

2.1.1 Tâches indépendantes sans communication retour, sans latences

Comme le lecteur a pu s'en apercevoir à la lecture du titre de ce paragraphe, nous avons (déjà) oublié les principes énoncés dans la partie modélisation (toujours, tu prendras les latences en compte), mais rappelons encore que notre but est de montrer que la minimisation du temps de complétion est trop difficile sur de telles plates-formes. Une telle simplification du modèle renforce donc notre propos plutôt qu'elle ne l'affaiblit.

2.1.1.1 Sur un arbre de profondeur 1

Considérons la plate-forme décrite à la Figure 2.1, à gauche. Le maître peut envoyer une tâche au processeur P_i en un temps d_i (on suppose qu'il n'y a pas de latence de communication, de sorte que le temps pour envoyer n tâches à P_i est nd_i). L'esclave P_i peut commencer le traitement d'une tâche une fois celle

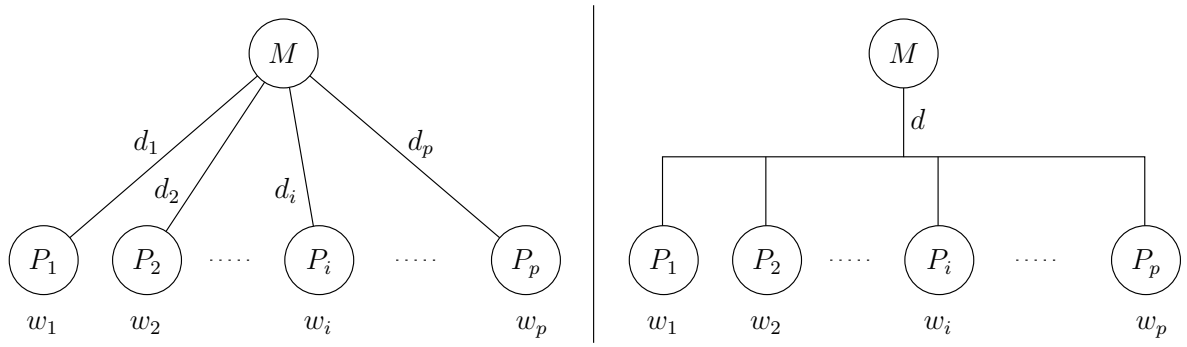


FIG. 2.1 – Architecture maître/esclave sur un arbre de profondeur 1.

ci complètement reçue, et le traitement d'une tâche par P_i prend un temps w_i (pendant ce temps w_i, P_i peut commencer à recevoir une autre tâche, nous supposons un recouvrement complet des calculs par les communications).

Notre objectif est de maximiser le nombre de tâches qui peuvent être traitées en utilisant cette plate-forme en temps T, ce qui peut être exprimé sous la forme suivante :

Définition 2.1. Maître-Esclave$(P_1(d_1, w_1), \ldots, P_p(d_p, w_p), T)$ *Étant données une plate-forme maître-esclave de caractéristiques* $(d_1, w_1), \ldots, (d_p, w_p)$*, et une borne* T *sur le temps d'exécution, quel est le nombre maximal de tâches pouvant être exécutées en temps* T *?*

Une telle formulation n'est cependant pas très réaliste. En effet, on attend d'un algorithme résolvant ce problème, qu'il nous fournisse la liste des tâches traitées par chaque processeur ainsi que leur ordonnancement : même si les tâches ont le même temps d'exécution, elles correspondent certainement à des données distinctes.

De plus, la taille de la description d'un tel ordonnancement est *a priori* proportionnelle à T (puisque le nombre de tâches pouvant être traitées augmente au mieux linéairement avec T). Si les données du problème se résument simplement à la liste des caractéristiques (d_i, w_i) des processeurs et à la valeur de T, la taille d'une telle description n'a donc aucune chance d'être polynomiale en la taille des données (qui est en $O(\log T)$ bits). On va donc faire figurer dans les données du problème la liste des tâches que possède initialement le maître et redéfinir ce problème d'optimisation de la façon suivante. Nous reviendrons sur cette limitation (liée à l'optimisation du temps de complétion) dans le Paragraphe 2.3.3.

Définition 2.2. Maître-Esclave-Tâches$(n, \mathcal{F}, p, \mathcal{W}, \mathcal{D}, T)$ *Étant donnés*
- *un ensemble* $\mathcal{F} = \{F_1, F_2, \ldots, F_n\}$ *de* n *tâches indépendantes et de même taille*
- *un ensemble* $\mathcal{W} = \{w_1, w_2, \ldots, w_p\}$ *de* p *temps d'exécution*
- *un ensemble* $\mathcal{D} = \{d_1, d_2, \ldots, d_p\}$ *de* p *délais de communication*
- *une borne sur le temps de complétion* T

déterminer un sous-ensemble de $\mathcal{F}$ *de cardinal maximum, composé de tâches pouvant toutes être exécutées en* T *unités de temps sur une plate-forme maître-esclaves de* p *processeurs* P_i *de paramètres* w_i *(calcul) et* d_i *(communication), pour* $1 \leqslant i \leqslant p$*.*

Il existe un algorithme polynomial (non trivial) pour résoudre **Maître-Esclave-Tâches**, que le lecteur intéressé pourra trouver dans [17]. Cet algorithme procède en deux étapes

1. on se ramène tout d'abord, en dupliquant les processeurs, au cas où chaque esclave peut traiter au plus une tâche.
2. un algorithme glouton assez sophistiqué permet alors de déterminer quels esclaves doivent traiter des tâches et dans quel ordre le maître doit leur envoyer.

La complexité de l'algorithme obtenu est de l'ordre de $O(n^2p^2)$, donc bien polynomial dans la taille des entrées de **Maître-Esclave-Tâches**. Nous discuterons de la complexité de cet algorithme dans le Paragraphe 2.3.3.

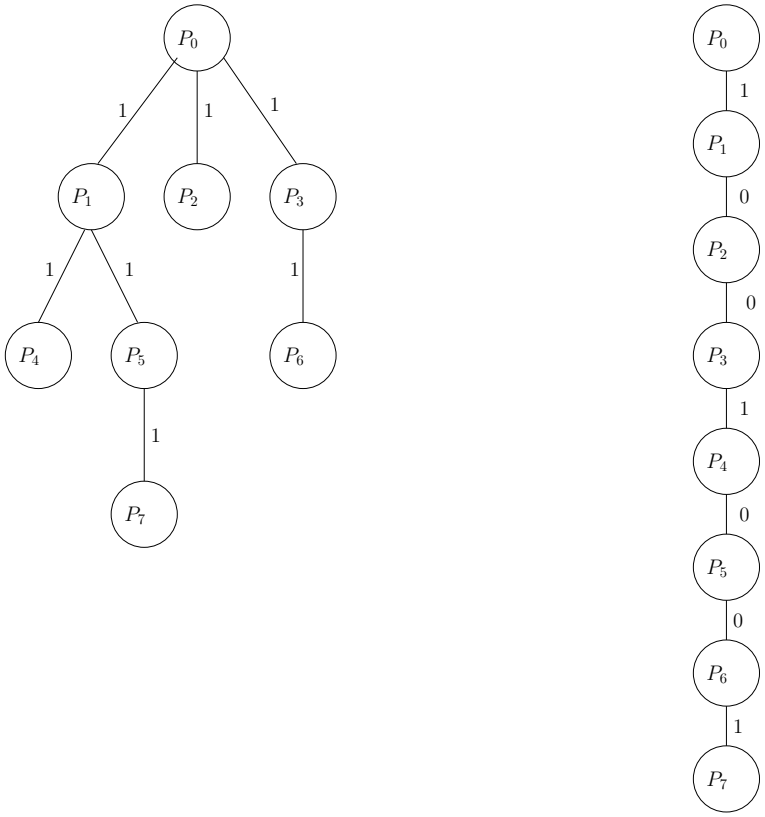


FIG. 2.2 – Solution pour un arbre à liens homogènes.

2.1.1.2 Sur une chaîne, une pieuvre, un arbre

Dutot [43] a étendu le résultat précédent au cas où la plate-forme est une chaîne ou une pieuvre. L'algorithme utilisé pour la chaîne est assez complexe, il consiste à choisir à chaque étape, compte tenu des décisions déjà prises, le processeur de la chaîne le plus loin du maître et qui est susceptible de traiter une tâche en temps T. L'algorithme pour la pieuvre s'appuie à la fois sur les techniques développées dans le cas des arbres de profondeur 1 et des chaînes.

Dutot [42] a également montré (par réduction à 3-Partition) que le problème est NP-Complet au sens fort dans le cas où la plate-forme est un arbre. Il est intéressant de remarquer que dans le cas où la plate-forme est un arbre, la difficulté démontrée par le résultat de NP-Complétude provient de l'hétérogénéité des vitesses des liens de communications et pas de l'hypothèse 1-port sur le mode opératoire des nœuds ou de l'hétérogénéité des vitesses des processeurs.

En effet, dans le cas où les liens sont homogènes (on supposera alors un coût unitaire de transmission d'une tâche), les processeurs sont hétérogènes et opèrent dans le modèle 1-port, le problème peut être résolu en temps polynomial. Dans ce cas, on peut en effet facilement montrer qu'il existe une solution optimale dans laquelle une tâche envoyée par le maître et destinée à un processeur P_i est toujours transmise au plus tôt par les nœuds intermédiaires. On peut alors montrer que, puisque le réseau est homogène, la seule contention possible sur les ports ou les liens de communication se situe au niveau du maître. Il est alors aisé de montrer que l'arbre (dont les liens sont homogènes) représenté à gauche dans la Figure 2.2 et la chaîne (dont les liens sont hétérogènes) représentée à droite sont équivalents pour la maximisation du nombre de tâches traitées en un temps donné. On peut alors utiliser l'algorithme développé par Dutot [43] sur les chaînes pour calculer l'ordonnancement optimal.

On a donc montré sur cet exemple qu'un problème facile, même dans le cas de processeurs hétérogènes opérant sous le modèle 1-port, devient difficile à cause de l'hétérogénéité des liens de communication.

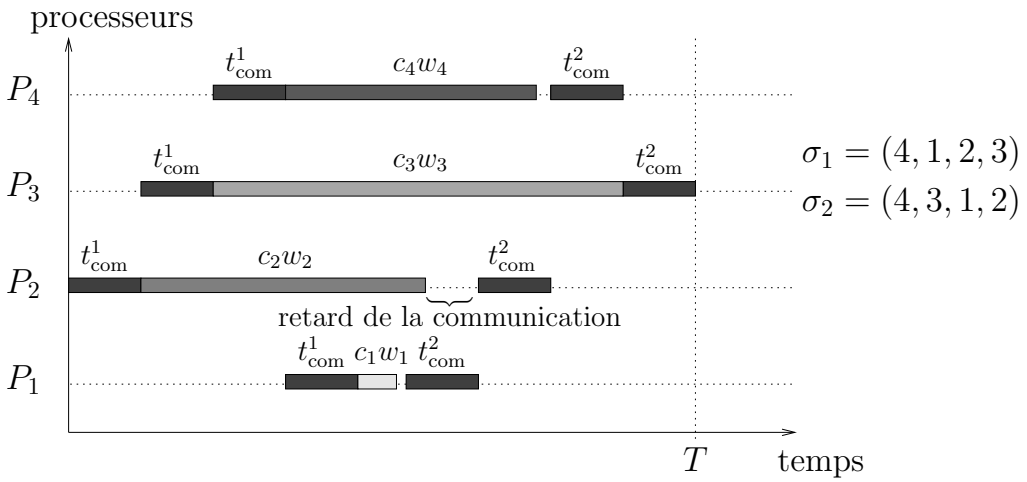


FIG. 2.3 – Retard des messages de retour.

2.1.2 Modèle avec latences et communications retour

Considérons maintenant l'arbre de profondeur 1 décrit à la Figure 2.1, à droite. Par rapport au paragraphe précédent, nous supposons maintenant que le réseau est homogène et que le coût de communication entre le maître et le processeur P_i se réduit à une latence t_{COM}, de sorte que le coût pour envoyer n tâches du maitre à P_i est donné par t_{COM} (les remarques du paragraphe précédent sur la simplification du modèle s'appliquent également ici). Comme précédemment, nous supposons que le temps nécessaire pour traiter une tâche sur P_i est donné par w_i. Par contre, nous supposons qu'une fois les tâches traitées sur les différents esclaves P_i, les résultats doivent être renvoyés au maître. Le coût de cette communication est de nouveau donné par t_{COM}.

Il est assez facile de montrer (comme précédemment, nous invitons le lecteur à consulter [18] pour trouver une preuve des résultats énoncés dans cette partie) que le schéma des communications et des calculs peut être choisi comme indiqué Figure 2.3, de sorte que

- il suffit de déterminer une permutation caractérisant l'ordre d'envoi des communications du maitre aux esclaves, une permutation caractérisant l'ordre d'envoi des communications des esclaves au maître et le nombre c_i de tâches envoyées au processeur P_i.
- il n'est pas indispensable de préciser l'ordonnancement des tâches individuelles, puisque plusieurs tâches sont envoyées simultanément à chacun des esclaves, de sorte que la taille des données de l'algorithme d'ordonnancement doit être en $O(\log n + p + \log T)$.

Le problème d'ordonnancement peut donc s'exprimer comme suit

Définition 2.3. Maître-Esclave-Retour *Étant donnée une borne T, déterminer la meilleure allocation des tâches aux esclaves, c'est-à-dire deux permutations, σ_1 et σ_2, et une allocation $\mathcal{C} = \{c_1, \ldots, c_p\}$ telle que chacun des p processeurs termine l'envoi des résultats avant T et qui maximise le nombre de tâches traitées $C = \sum_{i=1}^{p} c_i$:*

$$\max \left(\sum_{i=1}^{p} c_i \;\middle|\; \begin{array}{c} \sigma_1, \sigma_2 \text{ permutations et} \\ \forall 1 \leqslant i \leqslant p, \quad \sigma_1(i) t_{\text{COM}} + c_i w_i + \sigma_2(i) t_{\text{COM}} \leqslant T \end{array} \right)$$

Le lecteur trouvera dans [18] la preuve que le problème précédent est NP-Complet au sens fort, par une réduction à **RN-3DM**, dont la NP-Complétude a été démontrée par Yu [104].

Comme précédemment, il est intéressant de comprendre la source de la difficulté. Comme nous avons supposé le réseau de communication homogène, considérons le cas où les processeurs sont eux aussi homogènes (i.e. $w_i = w, \ \forall i$). Dans ce cas, l'ordre des envois des tâches aux différents processeurs n'a pas d'importance, puisque tous les processeurs sont identiques, de sorte qu'on peut choisir $\sigma_1 = \text{Id}$. Le seul problème est donc de déterminer la permutation σ_2 utilisée pour renvoyer les résultats. Le principe de l'algorithme est le suivant :

pour une valeur de $\sigma_2(i)$ fixée, le nombre de tâches que le processeur P_i peut exécuter avant de renvoyer les résultats est donné par

$$\lfloor \frac{T - (i + \sigma_2(i))t_{\text{COM}}}{w} \rfloor.$$

On peut alors construire un graphe biparti complet entre les processeurs P_i et les valeurs de σ_2 possibles (cf. Figure 2.4), dont les poids représentent le nombre de tâches que P_i pourrait effectuer pour ce choix de $\sigma_2(i)$. Un couplage de poids maximal dans le graphe biparti fournit alors la permutation $\sigma_2(i)$ qui maximise le nombre de tâches traitées, et donc résout le problème d'optimisation. Un couplage de poids maximal dans le graphe à $2p$ sommets construit peut être calculé en temps $O(p^{\frac{5}{2}})$ [59].

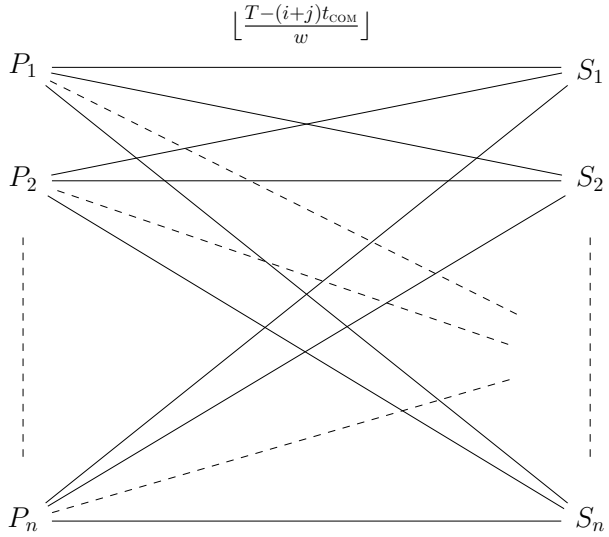


FIG. 2.4 – Graphe biparti permettant de résoudre le problème avec processeurs homogènes.

Nous vérifions donc sur cet exemple que dans ce cas, la source de la difficulté provient de l'hétérogénéité dans la vitesse des processeurs.

2.1.3 Conclusion

Il serait intéressant d'analyser un plus grand nombre de problèmes d'ordonnancement avec des processeurs et des liens hétérogènes, et un modèle de communication 1-port. Néanmoins, dans les paragraphes précédents, nous avons montré que deux problèmes simples (avec un modèle multi-port homogène) deviennent NP-Complet au sens fort avec notre modèle. Nous n'avons pas illustré la complexité introduite par le modèle 1-port, parce que ce fait est bien connu (par exemple, le temps de diffusion d'un message sur une plate-forme 1-port homogène quelconque est NP-Complet au sens fort [48] alors que le problème est trivial avec un modèle multi-port). Par contre, nous avons donné deux exemples où l'hétérogénéité des liens de communication (Paragraphe 2.1.1) où l'hétérogénéité des processeurs (Paragraphe 2.1.2) rendent à elles seules le problème difficile.

Si on souhaite obtenir des algorithmes polynomiaux pour résoudre une large classe de problèmes d'ordonnancement, il est donc nécessaire de tricher un peu (puisqu'on se refuse à modifier la modélisation des communications et des calculs). Dans le Paragraphe 2.2, nous allons étudier la possibilité de simplifier le modèle des applications. Ensuite, dans le Paragraphe 2.3 et en fait dans le reste de ce document, nous considérerons la possibilité de simplifier la métrique utilisée pour définir l'optimalité.

2.2 Simplification du modèle d'applications

Nous avons vu dans le paragraphe précédent que la prise en compte de l'hétérogénéité dans la vitesse des processeurs ou des liens, ainsi que le modèle 1-port, rend NP-complets au sens fort certains problèmes d'ordonnancement élémentaires dans un modèle homogène multi-port. Dans ce paragraphe, nous allons brièvement considérer trois tentatives pour contourner la difficulté intrinsèque liée à ces problèmes.

1. La première consiste à considérer globalement le graphe de tâches et le graphe de plate-forme et à utiliser des techniques de partitionnement de graphes pour allouer les tâches aux processeurs. Cette approche revient en fait à ne pas tenir compte des dépendances entre les tâches, mais de ne retenir que le volume de communication à échanger entre les processeurs, et de se concentrer uniquement sur l'équilibrage de la charge.
2. La seconde consiste à supprimer la prise en compte explicite des communications, en considérant des tâches malléables. Pour chaque tâche malléable, on associe, à chaque nombre possible de processeurs le temps observé ou prédit d'exécution sur ce groupe de processeurs. Ce temps prend donc à la fois en compte le surcoût lié à la parallélisation de la tâche et le surcoût induit par les communications.
3. La troisième consiste à simplifier (encore !) le modèle des tâches indépendantes, en considérant des tâches divisibles, c'est à dire des tâches parallèles idéales qui peuvent être partitionnées et distribuées de manière arbitraire.

2.2.1 Ordonnancement et partitionnement de graphes

Je présente ici rapidement les résultats obtenus dans le cadre du stage de DEA de Frédéric Raoult [83]. L'approche utilisée consiste à considérer simultanément le graphe de tâches (sans les orientations des arêtes), dont les sommets sont pondérés par des volumes de calcul et les arêtes par des volumes de communication, et le graphe de plate-forme (dont les liens de communication sont supposés bi-directionnels), dont les sommets sont pondérés par des vitesses de calcul et les arêtes par des vitesses de communication. Elle est à rapprocher des techniques de clustering développées par Yang et Gerasoulis dans le cadre de PYRROS [103].

Pour déterminer l'allocation des tâches aux processeurs, on utilise un algorithme récursif. À chaque étape, on utilise l'outil de partitionnement de graphes SCOTCH [77] développé par François Pellegrini. Le graphe de plate-forme est partitionné en deux parties (aussi équilibrées que possible du point de vue des vitesses de calcul) en minimisant le poids de la coupe. Le graphe de tâches est également partitionné en deux parties (aussi équilibrées que possible du point de vue des volumes de calcul) en minimisant le poids de la coupe (sans tenir de l'orientation des dépendances). On obtient ainsi deux sous-problèmes, les tâches d'une partition étant associées aux processeurs d'une partition.

Différentes stratégies de partitionnement ont été testées, mais malheureusement aucune d'entre elles ne donne des résultats très supérieurs aux algorithmes de listes adaptés au cas hétérogènes, comme HEFT [97] du point de vue de la minimisation du temps de complétion. Ces résultats assez décevants sont liés au fait que le partitionnement ne tient pas compte de l'ordre des dépendances, ce qui introduit au moment de l'exécution de l'ordonnancement de longues phases séquentielles. Cette approche est plus raisonnable dans le cas où on essaye de maximiser le débit (elle est à rapprocher de l'approche développée par Taura et Chien [95] et sur laquelle nous reviendrons au Paragraphe 4.3). Toutefois, nous verrons également au Chapitre 4 que si on cherche à maximiser le débit, alors il est possible de développer des algorithmes polynomiaux optimaux, pour une large classe de graphes de tâches. Nous avons donc abandonné cette approche, qui ne donne pas de très bons résultats en pratique, et pour laquelle il parait difficile de donner des garanties de performance, étant donnée la difficulté des problèmes de partitionnement sous-jacents.

2.2.2 Tâches malléables

L'utilisation des tâches malléables permet de limiter les difficultés liées à la prise en compte des communications, dans le cas de processeurs et de liens homogènes, quand on considère de longs délais de communication. On parle de longs délais de communication quand le plus petit coût de calcul est inférieur au plus

grand coût de communication. Dans le cas contraire, et dans le cas de processeurs homogènes (en nombre illimité) connectés par un réseau complet dans lequel le mode de communication est multi-port (c'est à dire encore une fois bien loin du modèle vu au Chapitre 1, Paragraphe 1.3), il existe des algorithmes d'approximation garantis [56]. Le lecteur intéressé pourra trouver dans la référence [41] une introduction aux tâches malléables, avec une bibliographie complète.

Avec les tâches malléables, on ne prend pas en compte explicitement les communications entre les tâches, pour ne retenir qu'un comportement plus grossier de l'application. Une tâche malléable est une tâche de calcul qui peut être exécutée sur un nombre arbitraire de processeurs. On suppose donné (ou observé expérimentalement) pour chaque nombre p de processeurs sur lequel la tâche est susceptible de s'exécuter, le temps de la tâche t_p sur p processeurs. t_p contient donc à la fois les informations sur les limites du parallélisme intrinsèque de la tâche et le surcoût introduit par les communications. Deux hypothèses (très réalistes en pratique) sont généralement faites pour diminuer la complexité des problèmes d'ordonnancement

1. t_p est une fonction décroissante en p
2. pt_p (le travail) est une fonction croissante en p.

Dans ce contexte de nombreux algorithmes ont été proposés. Si les cas les plus simples (tâches malléables indépendantes) restent NP-Complet, des algorithmes d'approximation garantis ont été développés pour les tâches indépendantes [27], les arbres de tâches malléables [71] et même les graphes de tâches malléables [71]. Il est à noter que dans le cas où on considère un graphe (arbre ou DAG) de tâches malléables, les coûts de communication entre les tâches malléables sont supposés nuls.

Un modèle encore plus simplifié pour ne pas prendre en compte explicitement les coûts de communication a été proposé par Prasanna et Musicus [80]. Dans ce modèle, le temps t_p pour exécuter une tâche (qu'on appellera malléable continue) sur $p \in \mathbb{Q}$ processeurs est donné sous la forme

$$t_p = \frac{L}{p^\alpha},$$

où L représente le temps de calcul de la tâche sur 1 processeur et α avec $0 \leqslant \alpha \leqslant 1$ est un facteur qui modélise à la fois le parallélisme intrinsèque de la tâche et le surcoût lié aux communications. Il est à noter qu'avec ce modèle, le profil des tâches malléables continues pour les valeurs de p entières vérifie les hypothèses de monotonie des tâches malléables (t_p décroissante et pt_p croissante).

Cette simplification du modèle a permis le développement de (surprenamment peu à mon goût) résultats de complexité. Par exemple, quand toutes les tâches malléables continues ont le même profil d'exécution (i.e. le même facteur α), alors il est possible de calculer un placement et un ordonnancement optimal pour des tâches indépendantes et pour des arbres de tâches [79] (encore une fois sans communications entre les tâches). Une des limitations du modèle des tâches malléables continues est liée à la nature du résultat. On obtient en effet un nombre rationnel de processeurs sur lequel exécuter la tâche malléable continue. Cet inconvénient est en général réglé en affirmant qu'il est possible d'exécuter plusieurs tâches sur le même processeur, avec des priorités adaptées pour que chaque tâche reçoive la fraction de la machine allouée par l'algorithme de placement. Il est sans doute plus raisonnable de considérer qu'il est alors nécessaire d'arrondir les résultats pour que chaque tâche soit allouée à un nombre entier de processeurs, mais la complexité de ce problème n'a (à ma connaissance) jamais été étudiée dans la cadre des tâches malléables continues.

La simplification du modèle de prise en compte des communications en utilisant les tâches malléables (ou malléables continues) est séduisante dans le cas de machines parallèles ou de clusters homogènes. Une des difficultés liée à l'utilisation des tâches malléables est créée par la difficulté d'obtention du profil de chaque tâche sur un nombre quelconque de processeurs. Néanmoins, si la variété des tâches à exécuter est relativement limitée, alors le temps d'exécution peut être obtenu par l'expérience, et j'espère que la lecture du Chapitre 1, Paragraphe 1.4 aura convaincu le lecteur que si une telle démarche est possible, elle peut finalement s'avérer plus fiable que la modélisation... Par contre, il semble très difficile de l'adapter au cas de plates-formes hétérogènes. En effet, le coût de l'expérience pour déterminer le temps d'exécution, pour chaque tâche malléable et chaque p-uplet de processeurs, est clairement rédhibitoire. Le modèle des tâches malléables continues ne semble pas non plus adaptable à des plates-formes hétérogènes. En effet, pour une même puissance agglomérée, le temps de calcul d'une tâche malléable continue sera sans doute très

différent selon qu'on utilise une machine puissante ou une multitude de processeurs de faible puissance et éparpillés géographiquement. Le fait de ne pas prendre explicitement les communications en compte revient malheureusement à supposer implicitement un réseau complet et homogène.

Toutefois, il n'est peut être pas impossible d'adapter le modèle des tâches malléables au cas de plates-formes hétérogènes et distribuées. On pourrait imaginer de représenter le temps d'exécution d'une tâche malléable continue comme une fonction de la puissance agglomérée des machines (l'équivalent de p) et de la dispersion des ressources de calcul (où, pour un groupe de processeurs de la plate-forme, la densité de la puissance de calcul serait donnée, par exemple, par $\sum_i \sum_j \frac{w_i}{v_{i,j}}$, où w_i représente la puissance de calcul du processeur P_i et $v_{i,j}$ l'inverse de la bande passante entre P_i et P_j). Néanmoins, à ma connaissance, de telles études n'ont pas (encore) été menées.

2.2.3 Tâches divisibles

Si les tâches malléables correspondent à une simplification par rapport à la gestion explicite des communications, les tâches divisibles correspondent à une simplification du modèle des applications parallèles. En effet, une tâche divisible est une tâche parallèle idéale, qui peut être découpée de façon arbitraire, chaque partie étant ensuite traitée de manière indépendante. Ce modèle se rapproche donc des tâches indépendantes, mais dans lequel on aurait enlevé les contraintes d'intégrité sur le nombre de tâches envoyées à chaque esclave.

Ce modèle a été largement étudié ces dernières années et a été rendu populaire par le livre de Bharadwaj, Ghose, Mani et Robertazzi [23]. Ce modèle couvre un grand nombre d'applications scientifiques comme le filtrage de Kalman [93], certains traitements d'image [66], la diffusion de flux vidéo et multimédia [2, 3], la recherche d'information dans une base de données [40, 26], ou encore le traitement de gros fichiers distribués [98] (d'autres exemples peuvent être trouvés dans [23]).

Sur le plan pratique, le modèle des tâches divisibles fournit un cadre de travail à la fois simple et réaliste pour étudier la distribution de tâches indépendantes sur une plate-forme hétérogène. La granularité des tâches peut être arbitrairement choisie, ce qui permet une grande flexibilité lors d'une mise en œuvre réelle. Sur le plan théorique, le succès rencontré par ce modèle provient du fait que de nombreux problèmes peuvent être résolus de façon analytique. Des algorithmes optimaux et des formes closes existent pour les instances simples de ce modèle. Le lecteur trouvera dans [19] et [11] les algorithmes que nous avons proposés pour la distribution des tâches divisibles sur des plates-formes en étoile analogues à celles décrites à la Figure 2.1, ainsi que des algorithmes périodiques asymptotiquement optimaux dans le cas où la distribution des tâches se fait en plusieurs fois.

2.2.4 Conclusion

Dans ce paragraphe nous avons analysé trois modèles de représentation des tâches. Le premier modèle consiste à négliger les dépendances en se concentrant sur le volume d'échange de données, le second consiste à ne pas gérer explicitement les communications, en ne retenant qu'un profil plus grossier d'exécution parallèle et le troisième consiste à considérer des tâches parallèles idéales indépendantes et infiniment divisibles. Tous ces modèles ont en commun de simplifier le modèle d'application pour pouvoir concevoir des algorithmes optimaux ou de bons algorithmes d'approximation, pour des problèmes qui en manquent dans le modèle général.

L'utilisation des tâches malléables se prête bien à des plates-formes homogènes mais l'adaptation au cas hétérogène parait difficile. À l'opposé, les tâches divisibles sont bien adaptées aux plates-formes hétérogènes (même si, comme nous le verrons, la prise en compte des latences est encore un problème ouvert), mais ne couvrent qu'un champ relativement réduit d'applications.

Dans la suite, nous allons tenter de convaincre le lecteur que pour concevoir des algorithmes optimaux sur les plates-formes que nous avons décrites dans le Chapitre 1, une solution consiste, non pas à changer le modèle des applications ou la prise en compte des communications, mais à changer la manière de définir l'optimalité !

2.3 Maximisation du débit et optimalité asymptotique

2.3.1 Justification pratique

Comme nous l'avons vu, la motivation du changement de métrique, de la minimisation du temps de complétion à la maximisation du débit, est essentiellement liée à la trop grande difficulté du premier problème. Néanmoins, on peut trouver d'autres justifications, plus pratiques, à cette petite lâcheté.

En effet, le temps de déploiement et la difficulté de la mise au point de programmes sur des environnements de type "grille de calcul" réduit l'utilisation de ces environnements à la résolution de grosses applications. De telles applications présentent souvent une forme de régularité : si elles ne peuvent pas être décomposées en un ensemble de tâches indépendantes, elles génèrent souvent, à un point donné de l'exécution, des problèmes de ce type. L'autre justification pratique est liée aux expériences "réussies" de calcul sur de grosses plates-formes distribuées. De telles expériences sont encore rares (Seti@Home[90], Genome@home [49], Mersenne@Home [81]) et leur degré de complexité en terme de parallélisation est bien bas : toutes les applications citées précédemment correspondent en effet à l'exécution d'un grand nombre de tâches identiques (sur des données différentes). L'étude présentée dans les prochains chapitres fournit un cadre théorique à de telles expériences (Chapitre 3). Comme nous le verrons, elle permet même d'envisager des schémas un peu plus complexes, où chaque tâche elle même peut être distribuée sur plusieurs processeurs (Chapitre 4). Enfin, les schémas de communications collectives étudiées au Chapitre 5 permettent d'imaginer des applications encore plus complexes, les communications collectives pouvant être utilisées lors des synchronisations induites par un code parallèle générant à chaque étape un grand nombre de tâches indépendantes.

2.3.2 Exemple : maximisation du débit pour le routage de paquets

Dans [22], Bertsimas et Gamarnik proposent deux algorithmes asymptotiquement optimaux pour le routage de paquets dans un réseau et pour les problèmes de type "flow-shop". Je vais détailler ici l'approche utilisée pour le routage de paquets, parce qu'elle a eu une grande influence sur la suite des travaux présentés dans ce document. La présentation sera très proche de celle qui est faite dans [22] (quelques généralisations possibles seront signalées au cours de l'exposé). Par contre, la formalisation des notions d'ordonnancement périodique, de motifs, d'optimalité asymptotique (qui n'est pas nécessaire dans le contexte assez simple du routage de paquets dans un modèle multi-port) ne sera présentée qu'au Chapitre 3, Paragraphe 3.3.

Le problème du routage de paquets est le suivant.

- Le réseau est représenté par un graphe $G = (V, E)$, dans lequel les liens sont homogènes (le passage au cas hétérogène n'est pas présenté dans [22], mais il ne pose pas de gros problème).
- A chaque paire de nœuds (V_k, V_l), on associe $n_{k,l}$, qui représente le nombre de messages à router entre V_k et V_l.
- Seule la contention sur les liens est prise en compte : un seul paquet peut circuler à un instant donné sur le lien entre V_i et V_j, mais par contre, V_i peut envoyer des messages à plusieurs de ses voisins (on peut relâcher cette contrainte de fonctionnement multi-port, mais au prix d'un effort plus important).
- Le but n'est pas de minimiser le temps nécessaire au routage de l'ensemble des paquets, mais de concevoir un algorithme asymptotiquement optimal. Plus précisément, les auteurs présentent un algorithme dont le temps de complétion C_H vérifie

$$C_H \leqslant C^* + O(\sqrt{C^*}),$$

où C^* représente un minorant du temps de complétion optimal pour le routage de paquets.

La première étape de l'algorithme (la plus fondamentale) consiste à trouver une borne inférieure sur le temps de complétion du problème de routage de paquets. Pour cela, les auteurs proposent de résoudre un programme linéaire, dans lequel, essentiellement, toutes les contraintes temporelles affirmant qu'un paquet doit être arrivé en un nœud pour pouvoir en repartir ont été supprimées. Le programme linéaire ne retient donc qu'un comportement global, en se concentrant sur les contentions sur les liens du réseau.

Notons $x_{i,j}^{k,l}$ le nombre de paquets qui doivent être routés entre V_k et V_l et qui pour cela, empruntent le lien entre V_i et V_j. Les quatre premières contraintes expriment que

– $n_{k,l}$ paquets doivent quitter V_k pour V_l :

$$\sum_{i \in \mathcal{V}(k)} x_{k,i}^{k,l} = n_{k,l},$$

où $\mathcal{V}(k)$ représente le voisinage de V_k.

– $n_{k,l}$ paquets doivent arriver en V_l depuis V_k :

$$\sum_{i \in \mathcal{V}(l)} x_{i,l}^{k,l} = n_{k,l}.$$

– les paquets routés de V_k à V_l qui arrivent en V_j, $j \neq k, l$ doivent nécessairement en repartir

$$\sum_{i \in \mathcal{V}(j)} x_{i,j}^{k,l} = \sum_{i \in \mathcal{V}(j)} x_{j,i}^{k,l}.$$

– les paquets routés de V_k à V_l qui ont quitté V_k ne peuvent plus y revenir, de même que les paquets qui sont déjà arrivés en V_l ne peuvent plus y revenir

$$\sum_{i \in \mathcal{V}(k)} x_{i,k}^{k,l} = \sum_{i \in \mathcal{V}(l)} x_{l,i}^{k,l} = 0.$$

(il est à noter que cette contrainte n'est pas explicitement présente dans le programme linéaire présenté dans [22], ce qui le rend incorrect.)

Clairement, tout ordonnancement optimal vérifie l'ensemble des contraintes ci dessus. De façon similaire, comme $\sum_{(k,l)} x_{i,j}^{k,l}$ représente le nombre de paquets circulant sur le lien entre V_i et V_j et qu'un seul message peut circuler à un instant donné sur ce lien, le temps de complétion $C_{\max}$ de tout ordonnancement doit vérifier

$$\forall (V_i, V_j) \in E, \quad C_{\max} \leqslant \sum_{(k,l)} x_{i,j}^{k,l}.$$

La valeur optimale du programme linéaire ci-dessous fournit donc un minorant C^* du temps de complétion optimal pour le routage de paquets :

ROUTAGE DE PAQUETS
Minimiser $C_{\max}$,
sous les contraintes

$$\begin{cases} \forall (k,l) \in V^2 & \sum_{i \in \mathcal{V}(k)} x_{k,i}^{k,l} = n_{k,l} \\ \forall (k,l) \in V^2 & \sum_{i \in \mathcal{V}(l)} x_{i,l}^{k,l} = n_{k,l} \\ \forall (k,l) \in V^2, \forall j \neq k,l & \sum_{i \in \mathcal{V}(j)} x_{i,j}^{k,l} = \sum_{i \in \mathcal{V}(j)} x_{j,i}^{k,l} \\ \forall (k,l) \in V^2 & \sum_{i \in \mathcal{V}(k)} x_{i,k}^{k,l} = \sum_{i \in \mathcal{V}(l)} x_{l,i}^{k,l} = 0 \\ \forall (i,j) \in E & C_{\max} \leqslant \sum_{(k,l)} x_{i,j}^{k,l} \\ \forall (k,l) \in V^2, \quad \forall (i,j) \in E & x_{i,j}^{k,l} \geqslant 0 \end{cases}$$

La deuxième étape de l'algorithme consiste à construire un ordonnancement dont le temps de complétion C_H vérifie

$$C_H \leqslant C^* + O(\sqrt{C^*}).$$

Bertsimas et Gamarnik proposent un algorithme d'ordonnancement en deux phases. Pendant la première phase, le temps est découpé en périodes de durée Ω. Si $x_{i,j}^{k,l}$ représente le nombre de paquets routés de V_k à V_l qui empruntent l'arête entre V_i et V_j en temps C^* dans la solution donnée par le programme linéaire, alors on définit $a_{i,j}^{k,l} = \lfloor \frac{\Omega x_{i,j}^{k,l}}{C^*} \rfloor$. Pour atteindre le débit donné par la solution du programme linéaire, il faudrait transmettre $\frac{\Omega x_{i,j}^{k,l}}{C^*}$ paquets routés de V_k à V_l sur l'arête entre V_i et V_j à chaque période de durée Ω. En

fait, pendant la première phase, on va essayer d'envoyer $a_{i,j}^{k,l}$ paquets de type (k,l) sur l'arête (V_i, V_j) à chaque période. Malheureusement, on ne dispose pas nécessairement de $\sum_{k,l} a_{i,j}^{k,l}$ paquets en V_i au début de la période, de sorte que le nombre de paquets envoyés pendant une période peut être inférieur à $a_{i,j}^{k,l}$ (on n'envoie pendant une période que des paquets reçus à des périodes antérieures et si on manque de paquets, l'arête lésée est choisie de façon arbitraire). L'ordonnancement ainsi construit pendant cette première phase n'est donc pas périodique, mais il est découpé en étapes de durées Ω.

Après $\lceil \frac{C^*}{\Omega} \rceil$ périodes de temps (de durée Ω) pendant lesquelles l'algorithme précédent est appliqué, la deuxième phase commence. Tous les paquets qui n'ont pas encore atteint leur destination sont routés séquentiellement, selon un plus court chemin entre leur position courante et leur destination.

Il est montré dans [22] que si on choisit $\Omega = \sqrt{C^*}$, alors
- la durée de la première phase est majorée par $C^* + \sqrt{C^*}$
- le nombre de paquets qui ne sont pas arrivés à destination après la première phase est de l'ordre de $O(\sqrt{C^*})$. Il est donc possible de réaliser la deuxième phase en un temps $O(\sqrt{C^*})$.

On obtient donc finalement un ordonnancement en deux phases dont le temps de complétion C_H vérifie

$$C_H \leqslant C^* + O(\sqrt{C^*}).$$

Dans le Chapitre 3, Paragraphe 3.3, nous développerons une méthodologie générale pour l'ordonnancement asymptotique. Comme nous le verrons, cette méthodologie a pour avantage de permettre de considérer un grand nombre de problèmes d'ordonnancement. Elle a pour second avantage de produire des ordonnancements qui sont effectivement périodiques, ce qui simplifie beaucoup l'exposé des algorithmes et les preuves d'optimalité asymptotique. Néanmoins, le travail de Bertsimas et Gamarnik a eu une grande influence sur les travaux qui suivent, et il était donc juste de le présenter "en l'état".

2.3.3 Retour sur la complexité des tâches indépendantes

Nous avons vu au Paragraphe 2.1 un algorithme polynomial optimal du point de vue de la minimisation du temps de complétion pour la distribution de tâches indépendantes sur un arbre de profondeur 1. La complexité de cet algorithme est en $O(n^2p^2)$. La taille de la description de l'ordonnancement est elle même en $O(n \log n \log p)$, puisque pour chaque tâche, il est nécessaire de donner la date à laquelle la tâche doit être envoyée (et la durée de l'ordonnancement est proportionnelle à n) et le processeur auquel elle va être envoyée. Ainsi, quand n devient très grand, le coût de l'ordonnancement peut devenir grand, même (et surtout) à l'échelle de la vitesse de variation des performances de la plate-forme.

Inversement, le coût de l'algorithme d'ordonnancement proposé par Bertsimas et Gamarnik ne dépend que de la taille de la plate-forme : le coût de l'algorithme tient principalement dans le coût de la résolution du programme linéaire, qui possède $(|E||V|^2 + |E| + 4|V|^2)$ contraintes et $(|E||V|^2)$ variables, où $|E|$ est le nombre de liens et $|V|$ le nombre de nœuds dans la plate-forme (la taille des coefficients du programme linéaire est par contre en $O(\log n)$). De la même manière, il est possible d'écrire le code de l'ordonnancement en chaque nœud de la plate-forme avec de l'ordre de $O(|V|^2|V| \log n)$ sous la forme : pour chaque type de message, pour chaque voisin, envoyer de l'ordre de $a_{i,j}^{k,l}$ messages (toutes les remarques faites ici seront plus amplement justifiées et précisées dans le Paragraphe 3.3).

De ce fait, le passage de la minimisation du temps de complétion à la maximisation du débit et à l'optimalité asymptotique a un autre avantage que la facilité de la résolution. Il est en effet possible de construire, comme nous le verrons au Chapitre 3, des algorithmes d'ordonnancement périodiques dont le temps d'exécution et la taille du code sont logarithmiques en le nombre de tâches à traiter. Dans le cadre de plates-formes dont les performances peuvent varier rapidement, cette propriété est clairement importante.

2.4 Ordonnancement et dynamicité

Nous avons vu au Chapitre 1 que la dynamicité des performances des liens de communication et des ressources de calcul est un point caractéristique essentiel des plates-formes que nous considérons. Une stratégie

usuelle, dans le cas homogène, est de s'appuyer sur des stratégies "dirigées par la demande", où les processeurs demandent une nouvelle tâche, dès qu'ils ont terminé leur travail.

Considérons par exemple une application de traitement d'images, dans laquelle chaque pixel doit être mis à jour avec les valeurs de ses trois voisins du dessus et de son voisin de gauche (la description de l'algorithme et les preuves des résultats énoncés ici peuvent être trouvées dans [8]). On suppose que les pixels sont alloués aux processeurs colonne par colonne, mais on négligera les coûts de communication. On peut facilement montrer que dans le cas homogène, l'algorithme "dirigé par la demande", dans lequel un processeur réclame une nouvelle colonne dès qu'il a terminé les colonnes qui lui ont été allouées, est optimal (et conduit à une distribution cyclique des colonnes aux processeurs). Dans le cas hétérogène, par contre, on peut montrer que la même stratégie conduit à des résultats dramatiques, puisque tous les processeurs se trouvent ralentis à la vitesse du processeur le plus lent (il existe par contre une stratégie très simple d'allocation des colonnes aux processeurs et qui donne le résultat optimal). Cet exemple montre bien que, plus encore que dans le cas homogène, il est nécessaire de faire une allocation statique performante et de ne pas se laisser guider par des stratégies purement dynamiques.

Néanmoins, il ne parait pas raisonnable d'espérer pouvoir maintenir une solution optimale à chaque instant, compte tenu des variations des performances des ressources de la plate-forme. Tout d'abord, il peut être difficile de maintenir les informations sur les performances des liens et des processeurs et de les regrouper pour prendre les décisions d'ordonnancement. D'autre part, recalculer la solution optimale à chaque instant peut induire un coût rédhibitoire de redistribution des données. Par exemple, nous avons étudié différentes stratégies de distribution de matrices sur des processeurs hétérogènes pour le produit de matrices [9, 84]. Nous avons montré que la recherche de la distribution optimale (même dans le cas d'un réseau homogène complet) est un problème NP-Complet, mais nous avons conçu des algorithmes d'approximation garantis. Le calcul de la solution par les algorithmes d'approximation est néanmoins coûteux, et n'est pas robuste à de petites variations des vitesses des processeurs (la solution optimale si les vitesses des processeurs ont toutes varié de ε peut nécessiter une redistribution complète des matrices). Dans ce contexte, nous avons développé dans [16] des techniques de "redistributions légères", dans lesquelles les processeurs ne peuvent s'échanger que des frontières. Cet exemple montre qu'il est illusoire de vouloir adopter la stratégie optimale à chaque instant.

Le tableau est donc bien sombre du point de vue de l'adaptation des stratégies d'ordonnancement aux variations de performances des plates-formes hétérogènes. Il faudrait savoir prendre les décisions "où on en a besoin, au plus près du terrain [...] pour plus de dynam[isme], plus de réactiv[ité] et plus de démocra[tie]" [82]. Dans ce contexte, le lecteur ne sera pas surpris de ne trouver que peu de résultats sur la prise en compte de la dynamicité. L'exemple le plus convaincant d'adaptation sera présenté au Chapitre 4, pour la distribution de tâches indépendantes sur un arbre de processeurs. Nous verrons que dans ce cadre, il est possible d'allouer aux processeurs de la plate-forme des informations de priorité "qualitatives" (un processeur peut être classé dans une catégorie parmi trois). On peut en déduire un algorithme dirigé par la demande (dans lequel la catégorie est utilisée comme priorité), qui donne des résultats très satisfaisants en pratique, mais qui a résisté à toute démonstration d'optimalité, comme nous le verrons au Chapitre 4 !

Chapitre 3

Quelques résultats théoriques pour la suite

3.1 Présentation des résultats

Dans ce chapitre, nous présentons deux résultats théorique essentiels, qui n'apparaissent pas explicitement dans les différentes publications auxquelles les prochains chapitres se réfèrent (le lecteur trouvera la raison de cette absence dans l'ennui qui le saisira sans doute à la lecture des prochains paragraphes...). Ils sont néanmoins essentiels à la correction des algorithmes présentés dans la suite de ce document.

Le premier résultat présente l'équivalence entre deux problèmes de décision fondés sur des programmes linéaires. Il est essentiel puisqu'il justifie la recherche d'algorithmes polynomiaux (ou même d'algorithmes d'approximation garantis) pour des problèmes qui n'appartiennent pas *a priori* à NP. Dans les chapitres suivants, par exemple, nous cherchons un ensemble pondéré d'allocations des tâches aux processeurs et des dépendances aux liens de communication (Chapitre 4), ou un ensemble pondéré d'arbres sur lesquels réaliser une diffusion (Chapitre 5). A priori, le nombre d'allocations possibles, ou le nombre d'arbres de diffusion possibles, n'est pas polynomial en la taille de la plate-forme (et les pondérations associées aux allocations ou aux arbres peuvent être arbitrairement grandes). Le premier résultat exprime que pour une classe de problèmes de décision fondés sur des programmes linéaires (et les problèmes abordés dans les chapitres suivants appartiennent à cette classe), on peut se contenter de chercher des ensembles pondérés où le nombre d'éléments non nuls et leurs poids respectifs sont bornés par des quantités polynomiales en la taille des données, sans changer le résultat du problème de décision.

Le second résultat est également central. Il affirme que si on sait construire un motif d'ordonnancement dans lequel les contraintes de précédence et les contraintes de ressource ont été remplacées par les conditions (faibles) suivantes

- Le processeur P_i a le temps de traiter toutes les tâches qui lui sont allouées en temps T_p,
- Le processeur P_i a le temps d'envoyer tous les messages qui lui sont alloués en temps T_p,
- Le processeur P_i a le temps de recevoir tous les messages qui lui sont envoyés en temps T_p,
- toutes les tâches de calcul et de communication correspondant au traitement de K graphes de tâches sont bien allouées aux processeurs et aux liens de la plate-forme,

alors, il est possible de construire un ordonnancement périodique dont le débit est $\frac{K}{T_p}$ (toutes les définitions seront données au Paragraphe 3.3). Ce résultat est fortement lié au modèle 1-port bidirectionnel défini dans le Chapitre 1 (Paragraphe 1.3) dans lequel un processeur peut simultanément envoyer et recevoir un message, et il n'est pas valide dans le modèle 1-port unidirectionnel (dans lequel un processeur peut recevoir ou envoyer un message à un instant donné), comme nous le verrons au Chapitre 5 (Paragraphe 5.4). Ce résultat nous permettra, dans la suite du document, de nous concentrer uniquement et indépendamment sur l'activité des processeurs et de leurs ports de communication, sans se soucier des contraintes de dépendance temporelle (comme c'est fait dans l'algorithme de Bertsimas et Gamarnik pour le routage de paquets présenté au Paragraphe 2.3))

3.2 Programmation linéaire

Considérons le problème de décision suivant, associé à un problème d'optimisation dont la taille des données est en $O(n^k \log c_{\max}^{k'} \log K)$ pour k et k' fixés, et défini par

DEC-PROG-LIN-1 : $\exists ?\ x \in \mathbb{Q}_m$, tel que

$$\begin{cases} Ax \leqslant b, & A \in \mathbb{Q}_{m,n}, b \in \mathbb{Q}_m \\ x \geqslant 0 & \\ c^t x \geqslant K, & c \in \mathbb{Q}_m \end{cases}$$

où les coefficients de A, b et c sont bornés (en valeur absolue) par $c_{\max}$.
m est *a priori* quelconque (donc en particulier, sa taille peut être de l'ordre de $\exp n$) de sorte que le nombre de contraintes et la taille du résultat x peuvent être exponentiels en la taille des données du problème d'optimisation initial. On suppose par contre qu'on dispose d'un oracle $\mathcal{O}$ qui pour toute donnée x sous la forme $(i_1, x_{i_1} = \frac{a_{i_1}}{b_{i_1}}), \ldots (i_l, x_{i_l} = \frac{a_{i_l}}{b_{i_l}})$ (et $x_i = 0$ sur les autres composantes) permet de déterminer si les contraintes

$$\begin{cases} Ax \leqslant b, & A \in \mathbb{Q}_{m,n}, b \in \mathbb{Q}_m \\ x \geqslant 0 & \\ c^t x \geqslant K, & c \in \mathbb{Q}_m \end{cases}$$

sont vérifiées en temps $O(l^{k''} \log(\max_k(a_{i_k}, b_{i_k}))$ pour k'' fixé.

La formulation de **DEC-PROG-LIN-1** n'appartient donc pas *a priori* à la classe NP. Toutefois, le nombre de contraintes non triviales (i.e. différentes de $x_i \geqslant 0$) est de l'ordre de n. Considérons donc le problème de décision suivant
DEC-PROG-LIN-2 : $\exists ?\ x \in \mathbb{Q}_m$, tel que

- x possède au plus n composantes non nulles
- les composantes non nulles de x, $x_i = \frac{a_i}{b_i}$ vérifient $\log(|a_i|) + \log(|b_i|) \leqslant 2n(\log n + \log c_{\max})$
- x est solution de

$$\begin{cases} Ax \leqslant b, & A \in \mathbb{Q}_{m,n}, b \in \mathbb{Q}_m \\ x \geqslant 0 & \\ c^t x \geqslant K, & c \in \mathbb{Q}_m \end{cases}$$

Théorème 3.1.

- **DEC-PROG-LIN-2** $\in$ *NP*
- $\exists$ *x solution de* **DEC-PROG-LIN-1** $\Rightarrow \exists$ *y solution de* **DEC-PROG-LIN-2** *et y est également solution de* **DEC-PROG-LIN-1**.

Lemme 3.1. DEC-PROG-LIN-2 $\in$ *NP*

Démonstration. Nous utilisons la donnée de x à partir de ses composantes non nulles $(i_1, x_{i_1} = \frac{a_{i_1}}{b_{i_1}}), \ldots (i_l, x_{i_l} = \frac{a_{i_l}}{b_{i_l}})$ comme certificat.

- La vérification des deux premières contraintes ne pose pas de problème (au plus n composantes non nulles et $\log(|a_i|) + \log(|b_i|) \leqslant 2n(\log n + \log c_{\max})$
- On utilise alors l'oracle $\mathcal{O}$ pour vérifier les contraintes linéaires, en temps $O(n^{k''} 2n(\log n + \log c_{\max}))$.

On peut donc vérifier si x est solution de **DEC-PROG-LIN-2** en temps polynomial en la taille des données, ce qui achève la preuve du lemme. ■

Lemme 3.2. $\exists$ *x solution de* **DEC-PROG-LIN-1** $\Rightarrow \exists$ *y solution de* **DEC-PROG-LIN-2** *et y est également solution de* **DEC-PROG-LIN-1**.

Démonstration. Soit x une solution de **DEC-PROG-LIN-1**. Alors, le programme linéaire suivant admet une solution y telle que $c^t y \geqslant c^t x \geqslant K$

$$\begin{array}{l} \textbf{Maximiser } c^t y, \\ \textbf{sous les contraintes} \\ \left\{ \begin{array}{ll} Ax \leqslant b, & A \in \mathbb{Q}_{m,n}, b \in \mathbb{Q}_m \\ x \geqslant 0 \end{array} \right. \end{array}$$

On sait [88] qu'il existe une solution optimale x^{opt} du programme linéaire précédent en un sommet du polyèdre défini par

$$\left\{ \begin{array}{ll} Ax \leqslant b, & A \in \mathbb{Q}_{m,n}, b \in \mathbb{Q}_m \\ x \geqslant 0 \end{array} \right.$$

x^{opt} est donc obtenu en résolvant un système linéaire de taille $m \times m$ de plein rang extrait de $\begin{pmatrix} A \\ I_m \end{pmatrix} x = \begin{pmatrix} b \\ 0_m \end{pmatrix}$ ($\begin{pmatrix} A \\ I_m \end{pmatrix}$ est nécessairement de rang m puisqu'elle contient I_m).
Dans le système linéaire $m \times m$, au plus n lignes de A ont été conservées, de sorte qu'au moins $(m-n)$ lignes sont du type $x_i = 0$, et donc au plus n composantes de x^{opt} sont non nulles.
Considérons le système $n' \times n'$ ($n' \leqslant n$) extrait du système linéaire $m \times m$ dans lequel les variables contraintes à 0 ont été supprimées, et notons y le vecteur des composantes non contraintes à 0 de x^{opt}. y est solution d'un système $A'y = b'$ où A' et b', de taille $n' \times n'$ et n', sont extraits de A et b.

Les coefficients de y peuvent donc être obtenus en utilisant les formules de Cramer [52] sous la forme $y_k = \frac{\det(N^{(k)})}{\det(D^{(k)})}$, où $N^{(k)}$ et $D^{(k)}$ sont des matrices $n' \times n'$ extraites de (A', b').
De plus,

$$\begin{array}{rcll} \det(N^{(k)}) & = & \prod_1^{n'} \lambda_j & \lambda_j \text{ valeur propre de } N^{(k)} \\ & \leqslant & ||N^{(k)}||_2^2 & \\ & \leqslant & (n' \max_{i,j} N_{i,j}^{(k)})^{n'} & \text{cf [52]} \\ & \leqslant & (n' c_{\max})^{n'} & \\ & \leqslant & (n c_{\max})^{n}. & \end{array} .$$

La même propriété est évidemment vérifiée par $\det(N^{(k)})$. On vérifie donc que

$$y_k = \frac{a_k}{b_k}, \quad \text{où } \log(|a_k|) + \log(|b_k|) \leqslant 2n(\log n + \log c_{\max}).$$

Considérons maintenant x^{opt}, vecteur de taille m construit en complétant y par des 0. On vérifie alors que
- x^{opt} possède au plus n composantes non nulles
- Si $x_k^{\text{opt}} = \frac{a_k}{b_k} \neq 0$, alors $\log(|a_k|) + \log(|b_k|) \leqslant 2n(\log n + \log c_{\max})$.
-

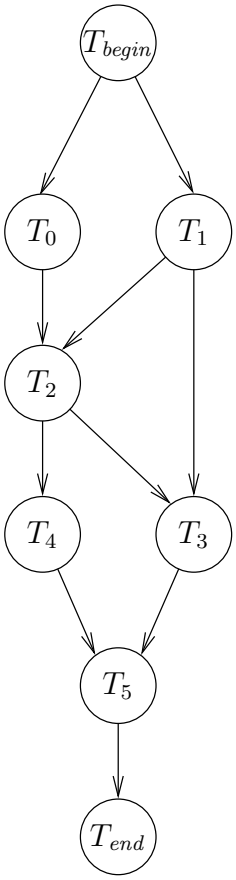
$$\left\{ \begin{array}{l} Ax^{\text{opt}} \leqslant b, \\ x^{\text{opt}} \geqslant 0 \\ c^t x^{\text{opt}} \geqslant K, \end{array} \right.$$

de sorte que x^{opt} est bien une solution de **DEC-PROG-LIN-2** (et par construction également une solution de **DEC-PROG-LIN-1**). ■

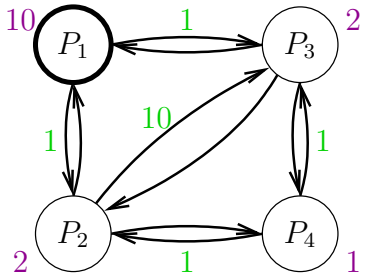
3.3 Ordonnancements périodiques et motifs appauvris

3.3.1 Rappels sur la modélisation des applications et des plates-formes

Nous avons présenté dans le Chapitre 1 la modélisation des plates-formes et des applications qui sera utilisée dans ce document. Nous allons ici définir formellement les notions d'ordonnancement (avec les contraintes de précédence, de ressource et de communication sous le modèle 1-port bidirectionnel correspondant à notre modèle). Dans le Chapitre 2, nous avons tenté de justifier pourquoi nous considérions le problème de la maximisation du débit. Pour cela, nous allons également présenter une adaptation à notre modèle des notions



(a) Graphe de tâches



(b) Graphe de plate-forme : Les fichiers d'entrée $e_{begin,1}$ sont placé sur P_1 à l'origine et les fichiers de sortie $e_{1,end}$ doivent être rassemblés sur P_1

FIG. 3.1 – Exemple simple de modélisation d'une application constituée de tâches indépendantes à exécuter sur une plate-forme hétérogène.

d'ordonnancement cyclique et d'ordonnancement périodique présentés par Alix Munier dans [74]. Il est à noter que le début de ce travail de formalisation a été présenté dans la thèse d'Arnaud Legrand [70].

Comme nous l'avons expliqué, nous nous intéressons à des applications qui se décomposent sous la forme d'un grand nombre n d'applications identiques représentées par le même graphe de tâches $G_A = (V_A, E_A)$. Nous supposons que dans G_A il existe une seule tâche sans prédécesseur, appelée T_{begin}, et une seule tâche sans successeur, appelée T_{end}. Il est évidemment toujours possible de modifier le graphe de tâches pour obtenir cette propriété (en ajoutant au besoin une tâche mère de toutes les tâches sans prédécesseur et une tâche fille de toutes les tâches sans successeur). Comme nous le verrons, ces tâches fictives permettent de modéliser les fichiers d'entrées et de sorties. Un exemple de graphe de tâches est représenté à la Figure 3.1(a). Chaque arête $(T_k, T_l) \in E_A$ est pondérée par le volume de données produites par T_k et nécessaires à l'exécution de T_l et noté $data_{k,l}$ dans la suite.

La plate-forme est représentée par un graphe orienté $G_P = (V_P, E_P)$ appelé *graphe de plate-forme* (un exemple de graphe de plate-forme est représenté à la Figure 3.1(b). Ce graphe est composé de p nœuds $P_1, P_2, \ldots, P_p$ qui représentent les différents processeurs. Chaque arête $P_i \rightarrow P_j$ représente un lien de communication et est étiquetée par $c_{i,j}$, l'inverse de la bande passante (et $L_{i,j}^{\text{comm}}$ la latence) du lien entre P_i et P_j. Notons que dans cette partie, nous allons négliger les latences, mais nous reviendrons sur ce point dans

le Paragraphe 3.3.7. S'il n'est pas possible de communiquer directement entre P_i et P_j, on pose $c_{i,j} = +\infty$. Notons enfin que ce graphe représente l'architecture physique de la plate-forme et que la matrice associée n'est donc pas nécessairement une matrice de distance (c'est-à-dire vérifiant $c_{i,j} \leqslant c_{i,k} + c_{k,j}$).

Concernant le mode opératoire des nœuds, rappelons que dans le modèle bidirectionnel, le processeur P_i peut simultanément

- calculer la tâche $T_k \in G_A$, ce qui lui prend un temps $w_{i,k}$
- envoyer un message (et un seul), qui correspond à une dépendance entre les tâches T_k et T_l de G_A, à un de ses voisins P_j dans G_P, ce qui lui prend un temps $data_{k,l} \times c_{i,j}$ (pendant tout ce temps P_i ne peut commencer un autre envoi)
- recevoir un message (et un seul), qui correspond à une dépendance entre les tâches T_k et T_l de G_A, d'un de ses voisins P_j dans G_P, ce qui lui prend un temps $data_{k,l} \times c_{j,i}$ (pendant tout ce temps P_i ne peut pas recevoir d'autre message)

Les tâches T_{begin} sont fictives et nous posons donc $w_{i,begin} = 0$ pour chaque processeur P_i qui possède des fichiers d'entrée (ou $w_{i,begin} \neq 0$ s'il les génère) et $w_{i,begin} = +\infty$ pour les autres. T_{end}, quant à elle, peut être utilisée pour modéliser deux situations différentes : soit les résultats (les fichiers de sortie) n'ont pas besoin d'être rassemblés à un endroit particulier et peuvent rester sur place, soit ils doivent être rapatriés sur un processeur particulier P_{dest} (pour réaliser une visualisation ou un post-traitement par exemple). Dans la première situation (les fichiers de sortie restent sur place), aucun fichier de sortie ne doit circuler d'un processeur à l'autre et donc on pose $w_{i,end} = 0$. Dans le cas contraire, où les fichiers doivent être rassemblés sur le processeur P_{dest} alors on définit $w_{dest,end}$ comme étant égal à 0 et $w_{i,end} = +\infty$ pour les autres processeurs, ce qui force les fichiers de type $e_{k,end}$ à être envoyés jusqu'à P_{dest}.

3.3.2 Formalisation de la notion d'ordonnancement

Dans ce paragraphe, nous allons formaliser les notions d'allocation et d'ordonnancement, qui sont bien connues, mais qui méritent néanmoins d'être précisées au regard du modèle que nous avons défini.

Définition 3.1 (Allocation). *Une allocation valide est composée d'une application $\pi : V_A \mapsto V_P$ et d'une application $\sigma : E_A \mapsto \{\text{chemin dans } G_P\}$ vérifiant pour tout $e_{k,l} : T_k \to T_l$:*

$$\sigma(e_{k,l}) = (P_{i_1}, P_{i_2}, \ldots, P_{i_p}) \text{ avec } \begin{cases} P_{i_1} = \pi(T_k), P_{i_p} = \pi(T_l) \text{ et} \\ (P_{i_j} \to P_{i_{j+1}}) \in E_P \text{ pour tout } j \in [\![1, p-1]\!] \end{cases}.$$

Définition 3.2 (Ordonnancement). *Un ordonnancement valide associé à une allocation valide (π, σ) est composé d'une application $t_\pi : V_A \mapsto \mathbb{R}$ et d'une application $t_\sigma : E_A \times E_P \mapsto \mathbb{R}$ vérifiant les contraintes suivantes :*

- ***précédence :*** *Pour tout $e_{k,l} : T_k \to T_l$, si $\sigma(e_{k,l}) = (P_{i_1}, P_{i_2}, \ldots, P_{i_p})$ alors*

$$\begin{aligned} t_\pi(T_k) + w_{i_1,k} &\leqslant t_\sigma(e_{k,l}, P_{i_1} \to P_{i_2}) \\ t_\sigma(e_{k,l}, P_{i_1} \to P_{i_2}) + data_{k,l} \times c_{i_1,i_2} &\leqslant t_\sigma(e_{k,l}, P_{i_2} \to P_{i_3}) \\ &\vdots \\ t_\sigma(e_{k,l}, P_{i_{p-1}} \to P_{i_p}) + data_{k,l} \times c_{i_{p-1},i_p} &\leqslant t_\pi(T_l) \end{aligned}$$

- ***ressources de calcul :*** *Un processeur ne peut traiter qu'une seule tâche à la fois. Pour tout $T_k \neq T_l$ on a donc :*

$$\pi(T_k) = \pi(T_l) \Rightarrow [t_\pi(T_k), t_\pi(T_k) + w_{\pi(T_k),k}[\cap [t_\pi(T_l), t_\pi(T_l) + w_{\pi(T_l),l}[= \emptyset$$

- ***ressources de communications :****Il ne peut circuler qu'un seul fichier à la fois entre P_i et P_j. Pour tout $e_{k_1,l_1} \neq e_{k_2,l_2} \in E_A$ et tout $P_i \to P_j$ on a donc :*

$$\begin{aligned} &[t_\sigma(e_{k_1,l_1}, P_i \to P_j), t_\sigma(e_{k_1,l_1}, P_i \to P_j) + data_{k_1,l_1} \times c_{i,j}[\\ &\qquad \cap [t_\sigma(e_{k_2,l_2}, P_i \to P_j), t_\sigma(e_{k_2,l_2}, P_i \to P_j) + data_{k_2,l_2} \times c_{i,j}[= \emptyset \end{aligned}$$

– ***1-port en entrée*** *Un processeur ne peut envoyer de fichier qu'à un seul processeur à la fois. Pour tout* $e_{k_1,l_1}, e_{k_2,l_2} \in E_A$ *et tout* $(P_{i_1} \to P_j) \neq (P_{i_2} \to P_j)$ *on a donc :*

$$[t_\sigma(e_{k_1,l_1}, P_{i_1} \to P_j), t_\sigma(e_{k_1,l_1}, P_{i_1} \to P_j) + data_{k_1,l_1} \times c_{i_1,j}[\\ \cap [t_\sigma(e_{k_2,l_2}, P_{i_2} \to P_j), t_\sigma(e_{k_2,l_2}, P_{i_2} \to P_j) + data_{k_2,l_2} \times c_{i_2,j}[= \emptyset$$

– ***1-port en sortie*** *Un processeur ne peut recevoir de données que d'un seul processeur à la fois. Pour tout* $e_{k_1,l_1}, e_{k_2,l_2} \in E_A$ *et tout* $(P_i \to P_{j_1}) \neq (P_i \to P_{j_2})$ *on a donc :*

$$[t_\sigma(e_{k_1,l_1}, P_i \to P_{j_1}), t_\sigma(e_{k_1,l_1}, P_i \to P_{j_1}) + data_{k_1,l_1} \times c_{i,j_1}[\\ \cap [t_\sigma(e_{k_2,l_2}, P_i \to P_{j_2}), t_\sigma(e_{k_2,l_2}, P_i \to P_{j_2}) + data_{k_2,l_2} \times c_{i,j_2}[= \emptyset$$

Définition 3.3 (Durée d'un ordonnancement). *La durée d'un ordonnancement valide* (t_π, t_σ) *est définie par*

$$\max_{T_k \in V_A} \left(t_\pi(T_k) + w_{\pi(T_k),k}\right) - \min_{T_k \in V_A} t_\pi(T_k)$$

3.3.3 Formalisation de la notion d'ordonnancement cyclique

Ce paragraphe définit formellement les notions d'ordonnancement cyclique et d'ordonnancement périodique (adaptés à notre modélisation) et présentés par Alix Munier dans [74].

Définition 3.4 (Graphe développé). *Soit* $G_A = (V_A, E_A)$ *un graphe de tâches et* S *un ensemble. On notera :*

$$V_A \otimes S = V_A \times S$$
$$E_A \otimes S = \left\{ \left((T_k, n), (T_l, n)\right) \middle| e_{k,l} : T_k \to T_l \in E_A \text{ et } n \in S \right\}$$
$$G_A \otimes S = (V_A \otimes S, E_A \otimes S)$$

$G_A \otimes S$ *est appelé graphe développé de* G_A *selon* S.

Définition 3.5 (Ordonnancement cyclique). *Un ordonnancement cyclique d'un graphe de tâches* G_A *sur une plate-forme* G_P *est un ordonnancement valide de* $G_A \otimes \mathbb{N}$ *sur* G_P.

Définition 3.6 (Durée d'un ordonnancement cyclique). *La durée des* N *premières tâches d'un ordonnancement cyclique est définie par :*

$$D_N = \max_{(T_k,n) \in V_A \times [\![1,N]\!]} \left(t_\pi(T_k, n) + w_{\pi(T_k,n),k}\right) - \min_{(T_k,n) \in V_A \times [\![1,N]\!]} t_\pi(T_k, n)$$

Définition 3.7 (Suite K-périodique). *Une suite* u *est dite* K*-périodique si elle est croissante et s'il existe un entier* n_0 *et un réel* T_p *strictement positif tels que*

$$\forall n \geqslant n_0 : u_{n+K} = u_n + T_p$$

K *est appelé facteur de périodicité et* T_p *représente la période de la suite.* $\frac{K}{T_p}$ *est alors la fréquence de la suite. Une suite telle que* $n_0 = 0$ *est dite* strictement K*-périodique*

Définition 3.8 (Ordonnancement K-périodique). *On identifie* $f(x, n)$ *et* $f(x)(n)$*. Un ordonnancement cyclique* (t_π, t_σ) *est dit* K*-périodique de période* T_p *si pour tout* $T_k \in V_A$, $t_\pi(T_k)$ *est strictement* K*-périodique de période* T_p *et si pour tout* $e_{k,l} \in E_A$ *et tout* $P_i \to P_j$, $t_\sigma(e_{k,l}, P_i \to P_j)$ *est strictement* K*-périodique de période* T_p.

Définition 3.9 (Débit d'un ordonnancement cyclique). *Le débit (ou fréquence) d'un ordonnancement est, sous réserve de son existence, la limite*

$$\lim_{N \to \infty} \frac{N}{D_N}$$

Remarque. *Un ordonnancement* K*-périodique de période* T_p *est entièrement caractérisé par les* K *premières valeurs des* t_π (T_k) *et des* t_σ $(e_{k,l}, P_i \to P_j)$.

Le débit d'un ordonnancement K*-périodique de période* T_p *est égal à* $\frac{K}{T_p}$.

3.3.4 Motifs et ordonnancements périodiques

Définition 3.10 (K-motif de longueur T_p). *Un K-motif de longueur T_p est une allocation (π, σ) de $G_A \otimes [\![1, K]\!]$ sur G_P et deux applications $\tilde{t}_\pi$ de $V_A \otimes [\![1, K]\!]$ dans $[0, T_p[$ et $\tilde{t}_\sigma$ de $(E_A \otimes [\![1, K]\!]) \times E_P$ dans $[0, T_p[$ vérifiant les contraintes de ressource et les contraintes 1-port.*

Intuitivement, un motif est donc une tranche de temps de longueur T_p pendant laquelle les tâches de calcul et de communication correspondant au traitement de K copies de G_A sont allouées. Cette allocation vérifie les contraintes de ressource (à chaque instant de la tranche de temps de durée T_p, un processeur traite au plus une tâche et est engagé dans au plus une communication sortante et au plus une communication entrante). Par contre, les contraintes temporelles de précédence ne sont pas *a priori* respectées dans le motif (une tâche T_l peut être traitée dans la tranche de temps avant une tâche T_k alors qu'il existe un chemin orienté de T_k vers T_l dans G_A).

Théorème 3.2. *: Soit $T_p \in \mathbb{R}_+^*$. Soit (π, σ), une allocation de G_A sur G_P. Étant donnés $\tilde{t}_\pi$ de $V_A \otimes [\![1, K]\!]$ dans $[0, T_p[$ et $\tilde{t}_\sigma$ de $E_A \times E_P \otimes [\![1, K]\!]$ dans $[0, T_p[$ vérifiant les contraintes de ressource et les contraintes 1-port, il existe un ordonnancement K-périodique (t_π, t_σ) de G_A sur G_P (et donc vérifiant également les contraintes de précédence), de période T_p, tel que*

$$\forall T_k \in V_A, \forall n \in \mathbb{N} : t_\pi(T_k, n) = \tilde{t}_\pi(T_k, n \mod K) \mod T_p$$
$$\forall e_{k,l} \in E_A, \forall P_i \to P_j \in E_P, \forall n \in \mathbb{N} : t_\sigma((e_{k,l}, n), P_i \to P_j) = \tilde{t}_\sigma(e_{k,l}, P_i \to P_j, n \mod K) \mod T_p$$

Démonstration. Le résultat du théorème est essentiel puisqu'il permet de reconstruire, à partir d'un K-motif de longueur T_p (dans lequel les contraintes de précédence ne sont pas respectées), un ordonnancement périodique valide (respectant à la fois les contraintes de ressource et de précédence) dont la fréquence est égale à $\frac{K}{T_p}$.

L'idée pour reconstruire l'algorithme périodique est la suivante : on va décaler les différentes tâches de calcul et de communication (sans changer les processeurs et les liens auxquelles elles sont associées dans le motif) jusqu'à respecter les contraintes de précédence. Plus précisément, on va décaler chaque tâche d'un certain multiple de T_p, afin de vérifier facilement les contraintes de ressource.

On définit π et σ à partir de $\tilde{\pi}$ et $\tilde{\sigma}$ par

$$\forall T_k \in V_A, \forall n : \pi(T_k, n) = \tilde{\pi}(T_k, n \mod K)$$
$$\forall e_{k,l} \in E_A, \forall P_i \to P_j \in E_P, \forall n : \sigma(e_{k,l}, n, P_i \to P_j) = \tilde{\sigma}(e_{k,l}, n, P_i \to P_j),$$

de sorte que dans la suite (pour alléger des notations déjà un peu lourdes) on identifiera un peu abusivement π et $\tilde{\pi}$, σ et $\tilde{\sigma}$.

On cherche donc t_π et t_σ sous la forme

$$t_\pi(T_k, n) = \tilde{t}_\pi(T_k, n \mod K) + (n + \Delta_\pi(T_k, n \mod K))T_p$$
$$t_\sigma((e_{k,l}, n), P_i \to P_j) = \tilde{t}_\sigma(e_{k,l}, P_i \to P_j) + (n + \Delta_\sigma(e_{k,l}, P_i \to P_j, n \mod K))T_p,$$

avec Δ_σ et Δ_π à valeurs dans $\mathbb{Z}$. Donnons les conditions nécessaires et suffisantes sur Δ_σ et Δ_π pour que (t_π, t_σ) soit bien un ordonnancement K-périodique de G_A sur G_P.

- **K-périodicité de période T_p :** par construction, pour tout T_k, $t_\pi(T_k)$ est clairement K-périodique de période T_p et pour tout $e_{k,l}$, $t_\sigma(e_{k,l})$ est également clairement K-périodique de période T_p.
- **contraintes de ressource :** Supposons que deux tâches (T_k, n) et (T_l, n') soient allouées au même processeur au même instant t. Alors, par construction, elles sont également allouées au même processeur dans le motif. En effet, par définition du motif,

$$[\tilde{t}_\pi(T_k, n \mod K), \tilde{t}_\pi(T_k, n \mod K) + w_{\pi(T_k, n \mod K), k}] \subset [0, T_p] \text{ et}$$

$$[\tilde{t}_\pi(T_l, n' \mod K), \tilde{t}_\pi(T_l, n' \mod K) + w_{\pi(T_l,n' \mod K,l}] \subset [0, T_p],$$

donc si

$$[t_\pi(T_k, n), t_\pi(T_k, n) + w_{\pi(T_k,n),k}] \bigcap [t_\pi(T_l, n'), t_\pi(T_l, n') + w_{\pi(T_l,n'),k}] \neq \emptyset,$$

alors, nécessairement

$$n + \Delta_\pi(T_k, n \mod K) = n' + \Delta_\pi(T_l, n' \mod K).$$

On vérifie alors que

$$\begin{aligned}[\tilde{t}_\pi(T_k, n \mod K), \tilde{t}_\pi(T_k, & \ n \mod K) + w_{\pi(T_k,n \mod K),k}] \\ & \bigcap [\tilde{t}_\pi(T_l, n' \mod K), \tilde{t}_\pi(T_l, n' \mod K) + w_{\pi(T_l,n' \mod K,l}] = \emptyset\end{aligned}$$

impose $k = l$ et $n = n' \mod K$ puisque le motif respecte les contraintes de ressource. Puisque de plus,

$$n + \Delta_\pi(T_k, n \mod K) = n' + \Delta_\pi(T_l, n' \mod K),$$

la seule solution est donnée par $(n, k) = (n', l)$.
On démontre d'une façon similaire que les contraintes de ressource sur les liens de communication sont également vérifiées.
L'ordonnancement construit est donc K-périodique de période T_p, et satisfait les contraintes de ressource, pour tout choix des valeurs de $\Delta_\pi(T_k, n)$ et $\Delta_\sigma(e_{k,l}, P_i \to P_j, n))$. Il nous reste à démontrer qu'en choisissant astucieusement ces valeurs, on peut assurer que les contraintes de précédence sont également vérifiées.

– **Contraintes de précédence :** Soit $e_{k,l} \in E_A$. Notons $\sigma(e_{k,l}) = (P_{i_1}, P_{i_2}, \ldots, P_{i_p})$. Dans la suite, on posera $n' = n \mod K$

$$\forall n \in \mathbb{N} : t_\sigma((e_{k,l}, n), P_{i_{p-1}} \to P_{i_p}) + data_{k,l} \times c_{i_{p-1},i_p} \leqslant t_\pi(T_l, n)$$
$$\Leftrightarrow$$
$$\tilde{t}_\sigma(e_{k,l}, P_{i_{p-1}} \to P_{i_p}, n') + \Delta_\sigma(e_{k,l}, P_{i_{p-1}} \to P_{i_p}, n')T_p + data_{k,l} \times c_{i_{p-1},i_p} \leqslant \tilde{t}_\pi(T_l, n') + \Delta_\pi(T_l, n')T_p$$
$$\Leftrightarrow$$
$$\Delta_\sigma(e_{k,l}, P_{i_{p-1}} \to P_{i_p}, n') \leqslant \left\lfloor \frac{\tilde{t}_\pi(T_l, n') - \tilde{t}_\sigma(e_{k,l}, P_{i_{p-1}} \to P_{i_p}, n') - data_{k,l} \times c_{i_{p-1},i_p}}{T_p} \right\rfloor + \Delta_\pi(T_l, n')$$

De même on a

$$\forall n \in \mathbb{N} : t_\sigma((e_{k,l}, n), P_{i_1} \to P_{i_2}) + data_{k,l} \times c_{i_1,i_2} \leqslant t_\sigma((e_{k,l}, n), P_{i_2} \to P_{i_3})$$
$$\Leftrightarrow$$
$$\begin{aligned}\Delta_\sigma(e_{k,l}, P_{i_1} \to P_{i_2}, n') \leqslant & \left\lfloor \frac{\tilde{t}_\sigma(e_{k,l}, P_{i_2} \to P_{i_3}, n') - \tilde{t}_\sigma(e_{k,l}, P_{i_1} \to P_{i_2}, n') - data_{k,l} \times c_{i_1,i_2}}{T_p} \right\rfloor \\ & + \Delta_\sigma(e_{k,l}, P_{i_2} \to P_{i_3}, n'),\end{aligned}$$

et

$$\forall n \in \mathbb{N} : t_\pi(T_k) + w_{i_1,k} \leqslant t_\sigma(e_{k,l}, P_{i_1} \to P_{i_2})$$
$$\Leftrightarrow$$
$$\Delta_\pi(T_k, n') \leqslant \left\lfloor \frac{\tilde{t}_\sigma(e_{k,l}, P_{i_1} \to P_{i_2}, n') - \tilde{t}_\pi(T_k, n') - w_{i_1,k}}{T_p} \right\rfloor + \Delta_\sigma(e_{k,l}, P_{i_1} \to P_{i_2}, n')$$

(t_π, t_σ) vérifie donc les contraintes de précédence si et seulement si le système de contraintes de potentiel suivant admet une solution :

$$\forall e_{k,l} : T_k \to T_l,\ \forall n' \in [1,K],$$
$$\begin{cases} \Delta_\pi(T_k,n') - \Delta_\sigma(e_{k,l},P_{i_1}\to P_{i_2},n') \leqslant \left\lfloor \dfrac{\tilde{t}_\sigma(e_{k,l},P_{i_1}\to P_{i_2},n') - \tilde{t}_\pi(T_k,n') - w_{i_1,k}}{T_p} \right\rfloor \\ \Delta_\sigma(e_{k,l},P_{i_1}\to P_{i_2},n') - \Delta_\sigma(e_{k,l},P_{i_2}\to P_{i_3},n') \leqslant \left\lfloor \dfrac{\tilde{t}_\sigma(e_{k,l},P_{i_2}\to P_{i_3},n') - \tilde{t}_\sigma(e_{k,l},P_{i_1}\to P_{i_2},n') - data_{k,l}\times c_{i_1,i_2}}{T_p} \right\rfloor \\ \vdots \\ \Delta_\sigma(e_{k,l},P_{i_{p-1}}\to P_{i_p},n') - \Delta_\pi(T_l,n') \leqslant \left\lfloor \dfrac{\tilde{t}_\pi(T_l,n') - \tilde{t}_\sigma(e_{k,l},P_{i_{p-1}}\to P_{i_p},n') - data_{k,l}\times c_{i_{p-1},i_p}}{T_p} \right\rfloor \end{cases}$$

G_A étant un graphe sans cycle, le graphe de potentiel associé à ces équations ne comporte pas non plus de cycle et donc, en particulier ne contient aucun cycle de poids négatif. Il a donc une solution qui peut être calculée en temps polynomial [37]. ■

3.3.5 Motifs appauvris et ordonnancements périodiques

Définition 3.11 (K-motif appauvri de longueur T_p). *Un K-motif appauvri de longueur T_p est une allocation (π,σ) de $G_A \otimes [\![1,K]\!]$ sur G_P vérifiant globalement les contraintes de ressource et les contraintes 1-port, i.e.*

- *P_i a le temps de traiter toutes les tâches qui lui sont allouées*

$$\forall i, \sum_{(k,r),r\in[1,K],\ \pi(T_k,r)=P_i} w_{i,k} \leqslant T_p.$$

- *P_i a le temps de recevoir toutes les communications entrantes*

$$\forall i, \sum_{(k,l,r,j),r\in[1,K],\ \sigma(e_{k,l},r)\ni P_j\to P_i} data_{k,l}\times c_{j,i} \leqslant T_p.$$

- *P_i a le temps d'envoyer toutes les communications sortantes*

$$\forall i, \sum_{(k,l,r,j),r\in[1,K],\ \sigma(e_{k,l},r)\ni P_i\to P_j} data_{k,l}\times c_{i,j} \leqslant T_p.$$

Théorème 3.3. *À partir de tout K-motif appauvri de longueur T_p, il est possible de construire un $A\times K$-motif de longueur $A\times T_p$, où $\log A$ est majoré par $|E_P|\log T_p$.*

Démonstration. Ce théorème nous sera essentiel dans le reste du document. Il permet en effet de négliger la construction explicite du motif, pour se concentrer uniquement sur l'activité des différents processeurs en calcul et communication. Les trois conditions données pour un motif appauvri sont en effet beaucoup plus simples à vérifier que les contraintes associées à un motif (qui nécessitent de vérifier les contraintes 1-port à chaque instant).

La valeur de A sera précisée dans la suite. Notre but est de construire, à partir du motif appauvri, deux applications $\tilde{t}_\pi$ de $V_A \otimes [\![1,AK]\!]$ dans $[0,AT_p[$ et $\tilde{t}_\sigma$ de $(E_A\otimes[\![1,AK]\!])\times E_P$ dans $[0,AT_p[$ vérifiant les contraintes de ressource et les contraintes 1-port. On construit $\tilde{\pi}$ et $\tilde{\sigma}$, associées au motif, à partir de π et σ, associées au motif appauvri, en posant

$$\begin{array}{ll} \forall n\in[1,KA], & \tilde{\pi}(T_k,n) = \pi(T_k, n \mod K) \\ \text{et } \forall n\in[1,KA], & \tilde{\sigma}(e_{k,l},n) = \sigma(e_{k,l}, n \mod K). \end{array}$$

Comme précédemment, on identifiera (et tout aussi abusivement que précédemment) π avec $\tilde{\pi}$ et σ avec $\tilde{\sigma}$

– **Construction de $\tilde{t}_\pi$:**
La construction de l'allocation des tâches de calcul au processeur P_i est relativement simple, puisque l'activité de calcul des tâches ne fait intervenir que le processeur lui-même et aucune autre ressource. On trie donc les tâches telles que $\pi(T_k, n \mod A) = P_i,\ n \in [1, KA]$ par ordre lexicographique en $(n \mod A, k)$ et on alloue ces tâches, par groupes de A tâches, dans cet ordre, au processeur P_i. Le temps de calcul de P_i dans le motif de longueur AT_p est donc borné par A fois le temps de calcul de P_i dans le motif appauvri de longueur T_p (et donc inférieur à AT_p).
On peut donc construire facilement une allocation des tâches de calcul correspondant aux KA copies de G_A.
– **Construction de $\tilde{t}_\sigma$:**
La construction de l'allocation des tâches de communication au processeur P_i est beaucoup plus délicate. En effet, pendant que P_i communique avec P_j, P_i ne doit pas être engagé dans une autre communication sortante, de même que P_j ne peut pas être engagé dans une autre communication entrante, de sorte que la construction de l'ordonnancement des communications ne peut être que globale.
Pour organiser les communications, on va construire un graphe biparti, comme celui représenté à la Figure 3.2. Les nœuds à gauche représentent les ports de communications sortantes des processeurs

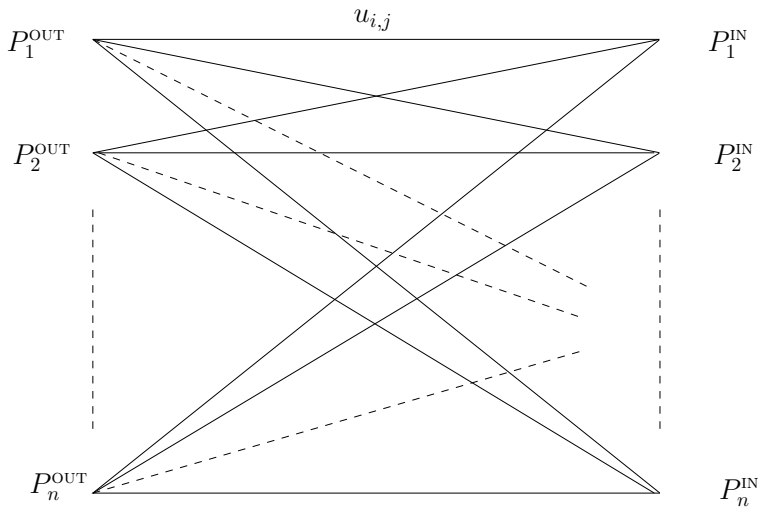


FIG. 3.2 – Graphe biparti permettant de résoudre le problème avec processeurs homogènes.

$P_1, \ldots, P_p$ et les nœuds à droite représentent les ports de communications entrantes des processeurs $P_1, \ldots, P_p$. L'arête entre P_i^{out} et P_j^{in} est pondérée par $u_{i,j}$ le volume total des communications échangées entre P_i et P_j pendant la durée totale d'un motif **appauvri** :

$$u_{i,j} = \sum_{(k,l,r),r\in[1,K],\ \sigma(e_{k,l},r)\ni P_j\to P_i} data_{k,l} \times c_{j,i},$$

de sorte que

$$\forall i,\ \sum_j u_{i,j} \leqslant T_p \text{ et } \sum_j u_{j,i} \leqslant T_p$$

par définition du motif appauvri.
Un couplage dans le graphe biparti correspond à un ensemble de communications indépendantes dans le modèle 1-port bidirectionnel. De plus, on sait [89, vol.A chapitre 20], qu'il est possible de décomposer le graphe biparti en une somme pondérée d'au plus $|E|$ couplages $(\mu_1, M_1), \ldots, (\mu_{|E|}, M_{|E|})$, où $|E|$ représente le nombre d'arêtes dans le graphe biparti (donc $|E| \leqslant |E_P|$). De plus, le poids cumulé des couplages (donc le temps nécessaire pour réaliser l'ensemble des communications) vérifie

$$\sum \mu_i = \max_i(\sum_j u_{i,j}, \sum_j u_{j,i}) \leqslant T_p.$$

Cette décomposition du graphe biparti en somme de couplages permet donc d'organiser le volume total de communication dans un motif de longueur T_p. Toutefois, si on construit gloutonnement le motif des communications (comme nous l'avons fait pour l'allocation des tâches aux processeurs), certaines communications peuvent se dérouler sur plusieurs couplages. Le motif construit serait alors incorrect, ce qui justifie l'introduction du facteur A. On va construire un motif de longueur AT_p, décomposé en une somme de couplages pondérés $(A\mu_1, M_1), \ldots, (A\mu_{|E|}, M_{|E|})$, en choisissant A pour assurer que pendant chaque couplage, entre chaque couple de processeurs P_i et P_j participant à ce couplage, un nombre entier de messages circulent entre P_i et P_j.
Pour cela, posons pour toute arête (P_i, P_j) du graphe biparti

$$\chi_{i,j} = \{m, \quad (P_i, P_j) \in M_m\},$$

de sorte que toutes les communications entre P_i et P_j doivent avoir lieu pendant les couplages de $\chi_{i,j}$. Posons également, pour i et j fixés,

$$n_{i,j}^{k,l} = |\{r \in [1, AK], \quad (P_i, P_j) \in \sigma(e_{k,l}, r)\}|,$$

c'est à dire le nombre total de messages de type (T_k, T_l) qui circulent sur l'arête entre P_i et P_j. Définissons enfin A avec

$$A = \text{ppcm}_m(\mu_m).$$

Pendant le couplage $(A\mu_m, M_m)$ on va transférer exactement

$$\frac{\mu_m}{\sum_{k \in \chi_{i,j}} \mu_k} A n_{i,j}^{k,l}$$

messages de type (T_k, T_l) entre P_i et P_j. Par construction $\frac{A}{\sum_{k \in \chi_{i,j}} \mu_k} \in \mathbb{N}$, de sorte que le nombre de messages envoyés pendant chaque couplage est bien entier et on vérifie sans difficulté que tous les messages de type (T_k, T_l) entre P_i et P_j sont bien envoyés pendant le motif de longueur AT_p.

On a donc construit un AK motif de longueur AT_p, où $A = \text{ppcm}_m(\mu_m)$. Avec cette définition, on vérifie que

$$\begin{array}{rcl} A & = & \text{ppcm}_m(\mu_m) \\ & \leqslant & \prod_m \mu_m \\ & \leqslant & T_p^{|E_P|} \end{array}$$

et donc $\log A \leqslant |E_P| \log T_p$. ■

Nous allons maintenant donner la description précise du AK-motif de longueur AT_p à partir de la description du K-motif appauvri de longueur T_p. Le K-motif appauvri de longueur T_p est décrit par

- $n_{i,k}$: le nombre de tâches de calcul T_k allouées au processeur P_i dans le K-motif appauvri de longueur T_p.
- $n_{i,j}^{k,l}$: le nombre de tâches de communication de type T_k, T_l transmises par le processeur P_i au processeur P_j dans le K-motif appauvri de longueur T_p.

Avec ces notations, l'algorithme de construction du K-motif de longueur T_p est donné par

On peut vérifier que le temps de communication de P_i pendant le couplage M_m est donné par

$$\frac{\mu_m}{\sum_{m' \in \chi_{i,j}} \mu_{m'}} \sum_{k,l} A n_{i,j}^{k,l} data_{k,l} \times c_{i,j} = \frac{A\mu_m}{\sum_{m' \in \chi_{i,j}} \mu_{m'}} \sum_{k,l} n_{i,j}^{k,l} data_{k,l} c_{i,j}$$

où

$$\sum_{k,l} n_{i,j}^{k,l} data_{k,l} c_{i,j}$$

représente le temps total de communication entre P_i et P_j, c'est à dire le poids total des couplages dans lesquels (P_i, P_j) apparaît, et qui est donc égal à $\sum_{m' \in \chi_{i,j}} \mu_{m'}$. On vérifie donc que le temps de communication de P_i pendant le couplage M_m est égal à $A\mu_m$.

Construire le graphe biparti
Le décomposer en somme pondérée de couplages (μ_i, M_i)
$\forall i, j \in E_P$, calculer $\chi_{i,j}$ et $\sum_{m \in \chi_{i,j}} \mu_m$

Description de l'activité de calcul de P_i
Pour $k = 1$ à $|V_A|$ **:**
  Calculer $An_{i,k}$ tâches de type T_k.
Attendre $AT_p - \sum_k An_{i,k} w_{i,k}$

Description de l'activité de communication de P_i
Pour chaque m tel que m (on notera P_j l'éventuel voisin de P_i dans M_m) **:**
  Pour chaque dépendance de type $(T_k, T_l) \in G_A$ **:**
    Envoyer $\frac{\mu_m}{\sum_{m' \in \chi_{i,j}} \mu_{m'}} An_{i,j}^{k,l}$ messages de type (T_k, T_l) à P_j
  Attendre $A\mu_m - \frac{\mu_m}{\sum_{m' \in \chi_{i,j}} \mu_{m'}} \sum_{k,l} An_{i,j}^{k,l} data_{k,l} \times c_{i,j}$

FIG. 3.3 – Description du AK-motif de longueur AT_p

3.3.6 Motifs appauvris compacts et ordonnancements périodiques

Le temps de calcul (et la taille de la description) de l'ordonnancement périodique associé au motif calculé au paragraphe précédent peut être grand, puisqu'*a priori*, pour chaque tâche (de calcul ou de communication) du motif, il est nécessaire de calculer le décalage associé à cette tâche (Δ_π ou Δ_σ) dans l'ordonnancement périodique. Or, si la valeur de A peut être codée de manière polynomiale en la taille du graphe de plate-forme et du graphe d'application, il n'est pas possible de calculer en temps polynomial les décalages associés à toutes les AK tâches du motif. Dans cette partie, nous allons prouver que si le schéma d'allocation des tâches aux processeurs peut être décrit de manière compacte dans le motif appauvri, alors il est possible de regrouper les différentes instances du graphe de tâches dans le motif en un nombre de groupes qui ne dépend que des caractéristiques de la plate-forme et de l'application. Nous montrerons également qu'il est possible d'associer à tous les membres du même groupe les mêmes valeurs de décalage. Ainsi, nous pourrons construire un ordonnancement périodique dont le temps de calcul et la taille d'encodage sont polynomiaux en $|V_A|, |E_A|, |V_P|, |E_P|, \log(\max c_{i,j}), \log(\max w_{i,k}), \log(\max data_{k,l}), \log K, \log T_p$ et Q le nombre de schémas d'allocations utilisés dans le motif appauvri.

Nous supposons que le motif appauvri est donné sous la forme $(\mathcal{A}_1, \alpha_1), \ldots, (\mathcal{A}_Q, \alpha_Q)$ où $\mathcal{A}_q$ est un schéma d'allocation et α_q le nombre de tâches allouées selon ce schéma dans le motif appauvri (avec $\sum_1^Q \alpha_i = K$). Toutes les tâches de calcul T_k sont allouées au même processeur et les tâches de communications (T_k, T_l) sont allouées à la même suite de liens (P_i, P_j) dans un schéma d'allocation donné $\mathcal{A}_q$, ce qui permet de définir $\pi(T_k, \mathcal{A}_q)$, le processeur qui traite les tâches de type T_k dans $\mathcal{A}_q$ et $\sigma(e_{k,l}, \mathcal{A}_q)$, la suite des liens (P_i, P_j) utilisés pour le transfert de la tâche de type (T_k, T_l) dans le schéma d'allocation. L'algorithme présenté à la Figure 3.4 permet de construire un motif de longueur AK dans lequel $A\alpha_i$ tâches sont allouées selon le même schéma $\mathcal{A}_i$, en utilisant la technique vue au paragraphe précédent.

Lemme 3.3. *Considérons $\mathcal{G}(\mathcal{A}_q)$ un groupe de g tâches allouées selon le même schéma d'allocation $\mathcal{A}_q$, de poids α_q, et dont toutes les tâches (de calcul et de communication) sont allouées consécutivement (et toujours dans le même ordre) sur les ressources dans le motif. Alors, il existe des valeurs de décalage $\Delta_\pi(T_k, \mathcal{G}(\mathcal{A}_q))$ et $\Delta_\sigma(e_{k,l}, P_i \to P_j, \mathcal{G}(\mathcal{A}_q))$ valides pour tous les éléments de $\mathcal{G}(\mathcal{A}_q)$.*

Démonstration. On note $\tilde{t}_\sigma(e_{k,l}, P_i \to P_j, n)$ la date de début dans le motif, pour le $n-$ième élément du groupe, de la communication entre P_i et P_j associée à la dépendance (T_k, T_l) et $\tilde{t}_\pi(T_k, p)$ la date de début dans le motif, pour le $n-$ième élément du groupe, du traitement de la tâche T_k.

Construire le graphe biparti
Le décomposer en somme pondérée de couplages (μ_i, M_i)
$\forall i, j \in E_P$, calculer $\chi_{i,j}$ et $\sum_{m \in \chi_{i,j}} \mu_m$

Description de l'activité de calcul de P_i
Pour $k = 1$ à $|V_A|$ **:**
  Pour $q = 1$ à Q tel que $\pi(T_k, \mathcal{A}_q) = P_i$ **:**
    Calculer $A\alpha_q$ tâches de type T_k.
Attendre $AT_p - \sum_k An_{i,k} w_{i,k}$

Description de l'activité de communication de P_i
Pour chaque m (notons P_j l'éventuel voisin de P_i dans M_m) **:**
  Pour chaque dépendance de type $(T_k, T_l) \in G_A$ **:**
    Pour $q = 1$ à Q tel que $\sigma(e_{k,l}, \mathcal{A}_q) \ni (P_i, P_j)$ **:**
      Envoyer $\frac{\mu_m}{\sum_{m' \in \chi_{i,j}} \mu_{m'}} A\alpha_q$ messages de type (T_k, T_l) à P_j
  Attendre $A\mu_m - \frac{\mu_m}{\sum_{m' \in \chi_{i,j}} \mu_{m'}} \sum_{k,l} An_{i,j}^{k,l} data_{k,l} \times c_{i,j}$

FIG. 3.4 – Description du AK-motif de longueur AT_p en utilisant les schémas d'allocation

On cherche des valeurs de $\Delta_\pi(T_k, \mathcal{A}_q)$ et $\Delta_\sigma(e_{k,l}, P_i \to P_j, \mathcal{A}_q)$, telles que

$$\forall e_{k,l} : T_k \to T_l,\ \forall n \in \mathcal{G}(\mathcal{A}_q),$$
$$\begin{cases} \Delta_\pi(T_k, \mathcal{G}(\mathcal{A}_q)) - \Delta_\sigma(e_{k,l}, P_{i_1} \to P_{i_2}, \mathcal{G}(\mathcal{A}_q)) \leqslant \left\lfloor \dfrac{\tilde{t}_\sigma(e_{k,l}, P_{i_1} \to P_{i_2}, n) - \tilde{t}_\pi(T_k, n) - w_{i_1,k}}{T_p} \right\rfloor \\ \Delta_\sigma(e_{k,l}, P_{i_1} \to P_{i_2}, \mathcal{G}(\mathcal{A}_q)) - \Delta_\sigma(e_{k,l}, P_{i_2} \to P_{i_3}, \mathcal{G}(\mathcal{A}_q)) \leqslant \left\lfloor \dfrac{\tilde{t}_\sigma(e_{k,l}, P_{i_2} \to P_{i_3}, n) - \tilde{t}_\sigma(e_{k,l}, P_{i_1} \to P_{i_2}, n) - data_{k,l} \times c_{i_1,i_2}}{T_p} \right\rfloor \\ \vdots \\ \Delta_\sigma(e_{k,l}, P_{i_{p-1}} \to P_{i_p}, \mathcal{G}(\mathcal{A}_q)) - \Delta_\pi(T_l, \mathcal{G}(\mathcal{A}_q)) \leqslant \left\lfloor \dfrac{\tilde{t}_\pi(T_l, n) - \tilde{t}_\sigma(e_{k,l}, P_{i_{p-1}} \to P_{i_p}, n) - data_{k,l} \times c_{i_{p-1},i_p}}{T_p} \right\rfloor \end{cases}$$

Par construction du groupe, on a

$$\begin{array}{lcll} \tilde{t}_\sigma(e_{k,l}, P_i \to P_j, n) & = & \tilde{t}_\sigma(e_{k,l}, P_i \to P_j, 0) + n data_{k,l} c_{i,j} & \text{et} \\ \tilde{t}_\pi(T_k, n) & = & \tilde{t}_\pi(T_k, 0) + n w_{i,k}. & \end{array}$$

On vérifie donc (par exemple) que

$$\left\lfloor \frac{\tilde{t}_\sigma(e_{k,l}, P_i \to P_j, n) - \tilde{t}_\pi(T_k, n) - w_{i,k}}{T_p} \right\rfloor = \left\lfloor \frac{\tilde{t}_\sigma(e_{k,l}, P_i \to P_j, 0) - \tilde{t}_\pi(T_k, 0) - w_{i,k} + n(data_{k,l} c_{i,j} - w_{i,k})}{T_p} \right\rfloor,$$

de sorte que $\forall p$,

$$\left\lfloor \frac{\tilde{t}_\sigma(e_{k,l}, P_i \to P_j, n) - \tilde{t}_\pi(T_k, n) - w_{i,k}}{T_p} \right\rfloor \leqslant \left\lfloor \frac{\tilde{t}_\sigma(e_{k,l}, P_i \to P_j, 0) - \tilde{t}_\pi(T_k, 0) - w_{i,k} + n(\max(0, data_{k,l} c_{i,j} - w_{i,k}))}{T_p} \right\rfloor.$$

Si on procède de la même manière avec tous les types de contraintes, on vérifie donc que les conditions pour que les décalages soient valides pour tous les éléments du groupe (noté $\mathcal{G}$ dans les équations suivantes pour

cause de place) sont données par

$\forall e_{k,l} : T_k \to T_l,\ \forall n \in \mathcal{G},$

$$\begin{cases} \Delta_\pi(T_k,\mathcal{G}) - \Delta_\sigma(e_{k,l}, P_{i_1} \to P_{i_2}, \mathcal{G}) \leqslant \left\lfloor \dfrac{\tilde{t}_\sigma(e_{k,l}, P_{i_1} \to P_{i_2}, n) - \tilde{t}_\pi(T_k, n) - w_{i_1,k} + n(\min(0, data_{k,l}c_{i_1,i_2} - w_{i_1,k}))}{T_p} \right\rfloor \\ \Delta_\sigma(e_{k,l}, P_{i_1} \to P_{i_2}, \mathcal{G}) - \Delta_\sigma(e_{k,l}, P_{i_2} \to P_{i_3}, \mathcal{G}) \leqslant \left\lfloor \dfrac{\tilde{t}_\sigma(e_{k,l}, P_{i_2} \to P_{i_3}, n) - \tilde{t}_\sigma(e_{k,l}, P_{i_1} \to P_{i_2}, n) - data_{k,l}c_{i_1,i_2} + n data_{k,l}(\min(0, c_{i_2,i_3} - c_{i_1,i_2}))}{T_p} \right\rfloor \\ \vdots \\ \Delta_\sigma(e_{k,l}, P_{i_{p-1}} \to P_{i_p}, \mathcal{G}) - \Delta_\pi(T_l, \mathcal{G}) \leqslant \left\lfloor \dfrac{\tilde{t}_\pi(T_l, n) - \tilde{t}_\sigma(e_{k,l}, P_{i_{p-1}} \to P_{i_p}, n) - data_{k,l}c_{i_{p-1},i_p} + n(\min(0, w_{i_p,l} - data_{k,l}c_{i_{p-1},i_p}))}{T_p} \right\rfloor \end{cases}$$

La résolution du système de potentiel ci-dessus donne donc des valeurs de décalage dans l'ordonnancement périodique valides pour tous les éléments du groupe $\mathcal{G}(\mathcal{A}_q)$. ■

Nous avons vu qu'il était possible de donner des valeurs de décalage communes à tous les éléments du groupe. Il nous reste à montrer comment construire les groupes dans le motif décrit à la Figure 3.4.

Lemme 3.4. *À toute allocation $\mathcal{A}_q$, on peut associer P groupes de tâches, où P est majoré par $|G_A||E_P|^2$.*

Démonstration. Dans le motif présenté dans la Figure 3.4, toutes les tâches de calcul associées à la même allocation $\mathcal{A}_q$ sont allouées de manière consécutive. Par contre, les tâches de communication entre P_i et P_j associées à une dépendance de type (T_k, T_l) sont distribuées entre tous les couplages de $\chi_{i,j}$, de sorte que toutes les tâches allouées selon le même schéma ne peuvent être toutes mises dans le même groupe.

– Notons
$$\forall P_i \to P_j \in \sigma(e_{k,l}, \mathcal{A}_q), \forall m \in \chi_{i,j}, \quad A(e_{k,l}, P_i \to P_j, m)$$
le nombre de messages transférés pendant le couplage M_m entre P_i et P_j et correspondant à une dépendance de type (T_k, T_l). On a donc
$$A(e_{k,l}, P_i \to P_j, m) = \frac{\mu_m}{\sum_{m' \in \chi_{i,j}} \mu_{m'}} A\alpha_q.$$

– Pour $\pi(T_k, \mathcal{A}_q) = P_i$, $A(T_k, P_i)$ représente le nombre de tâches T_k exécutées par P_i. On a donc
$$A(T_k, P_i) = A\alpha_q.$$

L'algorithme de construction des groupes associés à l'allocation $\mathcal{A}_q$ est décrit à la Figure 3.5.

1: $P = 0$
2: **Tant que** $\max(A(e_{k,l}, P_i \to P_j, m)) > 0$ **:**
3: $P = P + 1$;
4: $g_P = \min(A(e_{k,l}, P_i \to P_j, m))$
5: Constituer un groupe de g_P éléments et retirer g_P tâches à chaque ressource utilisée dans $\mathcal{A}_q$:
6: $\forall i, j, k, l, m$, tel que $A(e_{k,l}, P_i \to P_j, m) > 0$, $A(e_{k,l}, P_i \to P_j, m) = A(e_{k,l}, P_i \to P_j, m) - g_P$
7: $A(T_k, P_i) = A(T_k, P_i) - g_P$.

FIG. 3.5 – Constitution des groupes associés à l'allocation $\mathcal{A}_q$.

À chacune des P étapes on construit donc un groupe $\mathcal{G}_p(\mathcal{A}_q)$ de g_p éléments dont toutes les tâches sont toutes allouées consécutivement sur toutes les ressources de calcul et de communication. À chaque étape, au moins une des valeurs $A(e_{k,l}, P_i \to P_j, m)$ est annulée. Comme il existe au plus $|E_A||E_P|^2$ valeurs possibles non nulles de $A(e_{k,l}, P_i \to P_j, m)$, on vérifie donc que

$$P \leqslant |E_A||E_P|^2.$$

■

En utilisant les deux lemmes précédents, on montre donc que
- il est possible de constituer, pour chaque allocation, P groupes d'éléments dont toutes les tâches de calcul et de communication sont allouées de manière consécutive sur les ressources de calcul et de communication. On notera $\mathcal{G}_1(\mathcal{A}_q), \ldots, \mathcal{G}_P(\mathcal{A}_q)$ les groupes associés à l'allocation $\mathcal{A}_q$.
- il est possible d'associer à chacun de ces groupes $\mathcal{G}_p(\mathcal{A}_q)$ des valeurs de décalage dans l'ordonnancement périodique $\Delta_\pi(T_k, \mathcal{G}_p(\mathcal{A}_q))$ et $\Delta_\sigma(e_{k,l}, P_i \rightarrow P_j, \mathcal{G}_p(\mathcal{A}_q))$ qui sont valides pour tous les éléments de $\mathcal{G}(\mathcal{A}_q)$.

On obtient donc (enfin...) à la Figure 3.6 la description de l'ordonnancement périodique (sur $\mathrm{NB}^{\mathrm{iter}}$ itérations) et le théorème suivant, sur lequel la plus grande partie des résultats des chapitres suivants s'appuient

Théorème 3.4. *Soit un K-motif appauvri de longueur T_p donné sous la forme de Q schémas d'allocations pondérés $(\mathcal{A}_1, \alpha_1), \ldots, (\mathcal{A}_Q, \alpha_Q)$ (avec $\sum \alpha_q = K$). Il est possible de construire un ordonnancement $A.K$-périodique de période $A.T_p$ (où A est majoré par $|E_P| \log T_p$) à partir du motif appauvri en temps polynomial en $|V_A|, |E_A|, |V_P|, |E_P|, \log(\max c_{i,j}), \log(\max w_{i,k}), \log(\max data_{k,l}), \log K, \log T_p$ et Q. De plus, la taille du code de l'ordonnancement obtenu est elle même polynomiale en $|V_A|, |E_A|, |V_P|, |E_P|, \log(\max c_{i,j}), \log(\max w_{i,k}), \log(\max data_{k,l}), \log K, \log T_p$ et Q.*

1: Construire le motif en utilisant l'algorithme décrit à la Figure 3.4.
2: Constituer les groupes $\mathcal{G}_p(\mathcal{A}_q)$ en utilisant l'algorithme décrit à la Figure 3.5.
3: Calculer, dans le motif, pour chaque P_i et chaque groupe $\mathcal{G}_p(\mathcal{A}_q)$, pour chaque tâche T_k telle que $\pi(T_k, \mathcal{A}_q) = P_i$, la valeur de $t(P_i, T_k, \mathcal{A}_q))$, date de début du traitement par P_i des tâches T_k des éléments du groupe $\mathcal{G}_p(\mathcal{A}_q)$
4: Calculer, dans le motif, pour chaque P_i et chaque groupe $\mathcal{G}_p(\mathcal{A}_q)$, pour chaque dépendance $e_{k,l}$ telle que $\sigma(e_{k,l}, \mathcal{A}_q) \ni (P_i, P_j)$, la valeur de $t(P_i, e_{k,l}, \mathcal{G}_p(\mathcal{A}_q))$, date de début du transfert par P_i des tâches T_k, T_l des éléments du groupe $\mathcal{G}_p(\mathcal{A}_q)$
5: Calculer, pour chaque P_i et chaque groupe $\mathcal{G}_p(\mathcal{A}_q)$, pour chaque tâche T_k telle que $\pi(T_k, \mathcal{A}_q) = P_i$, la valeur de $\Delta_\pi(T_k, \mathcal{G}_p(\mathcal{A}_q))$
6: Calculer, pour chaque P_i et chaque groupe $\mathcal{G}_p(\mathcal{A}_q)$, pour chaque dépendance $e_{k,l}$ telle que $\sigma(e_{k,l}, \mathcal{A}_q) \ni (P_i, P_j)$, la valeur de $\Delta_\sigma(e_{k,l}, P_i \rightarrow P_j, \mathcal{G}_p(\mathcal{A}_q))$
7:
8:
9:
10: **Description de l'activité de calcul de P_i**
11: **Pour** $N^{\mathrm{iter}} = 1$ à $\mathrm{NB}^{\mathrm{iter}}$ **:**
12: **Pour** $k = 1$ à $|V_A|$ **:**
13: **Pour** $q = 1$ à Q tel que $\pi(T_k, \mathcal{A}_q) = P_i$ **:**
14: **Pour** chaque groupe $\mathcal{G}_p(\mathcal{A}_q)$ **:**
15: Calculer $A|\mathcal{G}_p(\mathcal{A}_q)|$ tâches de type T_k à la date $t(P_i, T_k, \mathcal{G}_p(\mathcal{A}_q)) + AT_p(\mathrm{N}^{\mathrm{iter}} + \Delta_\pi(T_k, \mathcal{G}_p(\mathcal{A}_q))$.
16:
17:
18:
19: **Description de l'activité de communication de P_i**
20: **Pour** $N^{\mathrm{iter}} = 1$ à $\mathrm{NB}^{\mathrm{iter}}$ **:**
21: **Pour** chaque m (notons P_j l'éventuel voisin de P_i dans M_m) **:**
22: **Pour** chaque dépendance de type $(T_k, T_l) \in G_A$ **:**
23: **Pour** $q = 1$ à p tel que $\sigma(e_{k,l}, \mathcal{A}_q) \ni (P_i, P_j)$ **:**
24: **Pour** chaque groupe $\mathcal{G}_p(\mathcal{A}_q)$ **:**
25: Envoyer $\frac{\mu_m}{\sum_{m' \in \chi_{i,j}} \mu_{m'}} A|\mathcal{G}_p(\mathcal{A}_q)|$ messages de type (T_k, T_l) à P_j, à la date $t(P_i, e_{k,l}, \mathcal{G}_p(\mathcal{A}_q)) + AT_p(\mathrm{N}^{\mathrm{iter}} + \Delta_\sigma(e_{k,l}, P_i \rightarrow P_j, \mathcal{G}_p(\mathcal{A}_q)))$.

FIG. 3.6 – Description compacte de l'ordonnancement AK-périodique de période AT_p

3.3.7 Avec des latences

Supposons maintenant que le motif appauvri ait été construit en négligeant les latences (c'est d'ailleurs ce qui sera fait dans la suite de ce document). Pour introduire les latences, l'idée est de construire, à partir du K-motif appauvri de longueur T_p, un NAK-motif dans lequel on prend en compte les latences de calcul et de communication et dans lequel on regroupe les communications et les calculs de manière à absorber le sur-coût introduit par les latences (en prenant N suffisamment grand).

Théorème 3.5. *À partir de tout K-motif appauvri de longueur T_p, il est possible de construire un NAK-motif de longueur $NAT_p + |E_P|L$, où $L = \max(\max_i L_i^{calc}, \max_{i,j} L_{i,j}^{comm})$.*

La description du motif avec latences est donnée à la Figure 3.7.

$L = \max(\max_i L_i^{\text{calc}}, \max_{i,j} L_{i,j}^{\text{comm}})$
Description de l'activité de calcul de P_i
Pour $k = 1$ à $|V_A|$ **:**
  Calculer $ANn_{i,k}$ tâches de type T_k.
Attendre $ANT_p + |E_P||E_A|L - (L_i^{\text{calc}} + \sum_k ANn_{i,k}w_{i,k})$

Description de l'activité de communication de P_i
Pour chaque m (on notera P_j l'éventuel voisin de P_i dans M_m) **:**
  Pour chaque dépendance de type $(T_k, T_l) \in G_A$ **:**
    Envoyer $\frac{\mu_m}{\sum_{m' \in \chi_{i,j}} \mu_{m'}} NAn_{i,j}^{k,l}$ messages de type (T_k, T_l) à P_j
  Attendre $NA\mu_m + |E_A|L - \sum_{k,l}(L_{i,j}^{\text{comm}} + \frac{\mu_m}{\sum_{m' \in \chi_{i,j}} \mu_{m'}} NAn_{i,j}^{k,l} data_{k,l} \times c_{i,j})$

FIG. 3.7 – Description du NAK-motif de longueur $NAT_p + |E_P||E_A|L$

Vérifions que le motif décrit ci-dessus est bien valide.

- **Activité de calcul :** En tenant compte des latences, le temps de calcul total du processeur P_i est donné par
$$L_i^{\text{calc}} + \sum_k ANn_{i,k}w_{i,k},$$
qui est bien inférieur à
$$ANT_p + |E_P||E_A|L$$
- **Activité de communication :** Soit M_m un couplage dans lequel P_i est impliqué en emission avec P_j. P_i fait un envoi à P_j pour potentiellement chaque type de tâche de communication (T_k, T_l). Le temps total de communication entre P_i et P_j est donc donné par
$$\sum_{k,l}(L_{i,j}^{\text{comm}} + \frac{\mu_m}{\sum_{m' \in \chi_{i,j}} \mu_{m'}} An_{i,j}^{k,l} data_{k,l} \times c_{i,j}),$$
qui est bien inférieur à
$$NA\mu_m + |E_A|L.$$
Comme le nombre de couplages est majoré par $|E_P|$, on vérifie donc que la durée totale du motif est bien inférieure à
$$NA\sum_m \mu_m + |E_A||E_P|L = NAT_p + |E_A||E_P|L.$$

Comme dans le paragraphe précédent, si le K-motif appauvri de longueur T_p est donné sous la forme de Q schémas d'allocation pondérés $(\mathcal{A}_1, \alpha_1), \ldots, (\mathcal{A}_Q, \alpha_Q)$ (avec $\sum \alpha_q = K$), on peut montrer qu'on peut

reconstruire en temps polynomial en $|V_A|$, $|E_A|$, $|V_P|$, $|E_P|$, $\log(\max c_{i,j})$, $\log(\max w_{i,k})$, $\log(\max data_{k,l})$, $\log K$, $\log T_p$ et Q, un ordonnancement périodique dont le débit est

$$\frac{ANK}{NAT_p + |E_A||E_P|L}.$$

En choisissant N suffisamment grand, on peut donc construire un ordonnancement périodique dont le débit est arbitrairement proche de $\frac{K}{T_p}$, de sorte que la prise en compte des latences de calcul et de communication ne pose pas de difficulté si on est intéressé par la construction d'ordonnancements périodiques (et si on oublie les problèmes liés à la limitation des capacités de mémorisation des nœuds, comme nous le verrons au Paragraphe 4.3).

3.4 Conclusion

La démonstration des résultats de ce chapitre est un peu laborieuse. Les résultats que nous avons démontrés sont néanmoins essentiels pour concevoir des ordonnancements périodiques de débit optimal. Dans la suite du document, la méthodologie employée sera en effet (grossièrement) la suivante :

- Pour un problème d'ordonnancement, on cherchera un K motif appauvri de longueur T_p et de débit $\frac{K}{T_p}$ optimal sous la forme d'un ensemble pondéré d'allocations $(\mathcal{A}_1, \alpha_1), \ldots, (\mathcal{A}_Q, \alpha_Q)$. Pour cela, nous nous concentrerons uniquement sur l'activité des processeurs (en calcul et en communication sous le modèle 1-port bidirectionnel). Nous serons alors amenés, comme nous le verrons, à écrire des programmes linéaires avec relativement peu de contraintes (une pour chaque type d'activité par processeur) mais beaucoup de variables (une pour chaque schéma d'allocation possible). Le premier résultat (Paragraphe 3.2) nous permet de restreindre la recherche à des motifs constitués de relativement peu d'allocations avec des poids raisonnables (le peu et le raisonnable sont évidemment à préciser en fonction du problème, on cherchera en général un nombre d'allocations polynomial en la taille du graphe de plate-forme et du graphe d'application, dont les poids peuvent être également encodés de manière polynomiale en la taille du graphe de plate-forme et du graphe d'application).
- Une fois un motif appauvri de débit optimal obtenu sous la forme $(\mathcal{A}_1, \alpha_1), \ldots, (\mathcal{A}_Q, \alpha_Q)$, alors, en utilisant les résultats du Paragraphe 3.3, on peut construire en temps polynomial en la taille de la plate-forme et la taille de la description du motif appauvri, un ordonnancement périodique qui réalise le débit optimal $\frac{K}{T_p}$ si on néglige les latences, et un ordonnancement périodique de débit arbitrairement proche de $\frac{K}{T_p}$ si on prend en compte les latences.

Chapitre 4

Distribution de tâches indépendantes

4.1 Tâches indépendantes élémentaires

4.1.1 Introduction

Dans ce paragraphe, on considère un ensemble de tâches élémentaires, qui doivent être exécutées sur une plate-forme quelconque, selon le paradigme maître-esclaves. Comme dans le Chapitre 3, Paragraphe 3.3, on ajoute deux tâches fictives T_{begin} et T_{end} qui modélisent les échanges de fichiers, avant et après le traitement de chaque tâche (voir Figure 4.1). La modélisation de la plate-forme est également la même que dans le Chapitre 3, Paragraphe 3.3. Dans ce paragraphe, on supposera que les fichiers correspondant aux données des tâches sont initialement sur P_1, le maître (i.e. $w_{T_{begin},P_1} = 0$ et $w_{T_{begin},P_i} = +\infty$ pour $i \neq 1$), et que les résultats doivent être regroupés chez le maître (i.e. $w_{T_{end},P_1} = 0$ et $w_{T_{end},P_i} = +\infty$ pour $i \neq 1$).

4.1.2 Formalisation

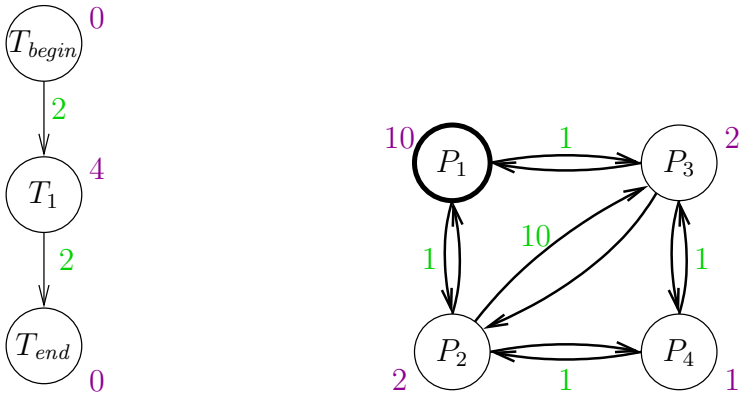
Dans le contexte des tâches élémentaires dont les fichiers d'entrée sont regroupés chez le maître et dont les fichiers de sortie doivent être regroupés chez le maître, une allocation $\mathcal{A}_q$ est simplement définie par

- $\pi(\mathcal{A}_q)$, le processeur P_i qui traite la tâche de calcul
- $\sigma(\mathcal{A}_q, e_{begin,l})$, le chemin entre P_1 et P_i pour acheminer les fichiers d'entrée (de volume $data_{begin,1}$)
- $\sigma(\mathcal{A}_q, e_{1,end})$, le chemin entre P_i et P_1 pour acheminer les fichiers de sortie (de volume $data_{1,end}$)

On sait qu'à partir d'un K-motif appauvri de longueur T_p, il est possible de construire un ordonnancement périodique de débit $\frac{K}{T_p}$ (Théorème 3.4). Dans un motif appauvri, les seules contraintes à vérifier sont les contraintes de ressource sous le modèle 1-port. Maximiser le débit du motif appauvri est donc équivalent à résoudre le programme linéaire suivant, où α_k représente le poids de l'allocation $\mathcal{A}_k$:

Maximiser $\frac{\sum_q \alpha_q}{T_p}$,
sous les contraintes

$$\begin{cases} \forall P_i, & \displaystyle\sum_{q,\ \pi(\mathcal{A}_q)=P_i} \alpha_q w_i \leqslant T_p & & \text{calcul sur} P_i \\ \forall P_i, & \displaystyle\sum_{j,\ (P_i,P_j)\in E_P} \sum_{e_{k,l}\in E_A} \sum_{q,\ \sigma(\mathcal{A}_q,e_{k,l})\ni(P_i,P_j)} \alpha_q c_{i,j} data_{k,l} \leqslant T_p & & \text{communications sortantes de } P_i \\ \forall P_i, & \displaystyle\sum_{j,\ (P_j,P_i)\in E_P} \sum_{e_{k,l}\in E_A} \sum_{q,\ \sigma(\mathcal{A}_q,e_{k,l})\ni(P_j,P_i)} \alpha_q c_{j,i} data_{k,l} \leqslant T_p & & \text{communications entrantes en } P_i \\ \alpha_q \geqslant 0 & & & \text{poids des allocations} \end{cases}$$



(a) Graphe de tâches

(b) Graphe de plate-forme : Les fichiers d'entrée $e_{begin,1}$ sont placé sur P_1 à l'origine et les fichiers de sortie $e_{1,end}$ doivent être rassemblés sur P_1

FIG. 4.1 – Exemple simple de modélisation d'une application constituée de tâches indépendantes à exécuter sur une plate-forme hétérogène.

ou, si on pose $\beta_q = \frac{\alpha_q}{T_p}$,

Maximiser $\sum_q \beta_q$,
sous les contraintes

$$\begin{cases} \forall P_i, & \displaystyle\sum_{q,\ \pi(\mathcal{A}_q)=P_i} \beta_q w_i \leqslant 1 & & \text{calcul sur} P_i \\ \forall P_i, & \displaystyle\sum_{j,\ (P_i,P_j)\in E_P} \sum_{e_{k,l}\in E_A} \sum_{q,\ \sigma(\mathcal{A}_q,e_{k,l})\ni(P_i,P_j)} \beta_q c_{i,j}\, data_{k,l} \leqslant 1 & & \text{communications sortantes de } P_i \\ \forall P_i, & \displaystyle\sum_{j,\ (P_j,P_i)\in E_P} \sum_{e_{k,l}\in E_A} \sum_{q,\ \sigma(\mathcal{A}_q,e_{k,l})\ni(P_j,P_i)} \beta_q c_{j,i}\, data_{k,l} \leqslant 1 & & \text{communications entrantes en } P_i \\ \beta_q \geqslant 0 & & & \text{poids des allocations} \end{cases}$$

Remarque. *Dans la suite, on considérera systématiquement les allocations sous forme fractionnaire et on cherchera le nombre maximal de tâches qui peut être traité en temps 1, ce qui revient à faire le passage de* α_q *à* β_q.

Le programme linéaire ci-dessus n'est certainement pas facile à résoudre, mais néanmoins, il nous permet d'affirmer qu'il est raisonnable de se poser la question de la maximisation du débit. En effet, le programme linéaire contient $3|V_P|$ contraintes différentes de $\beta_q \geqslant 0$, et à partir d'une allocation pondérée donnée sous la forme $(\beta_1, \mathcal{A}_1), \ldots, (\beta_Q, \mathcal{A}_Q)$, il est facile de vérifier les contraintes sur l'utilisation des ressources (en calcul, communications sortantes et communications entrantes) en temps polynomial en Q (et la taille des coefficients des β_q). On est donc dans les conditions d'applications du Théorème 3.1, qui nous permet d'affirmer qu'il existe une solution optimale composée d'au plus $3|V_P|$ allocations, dont les poids $\beta_k = \frac{a_k}{b_k}$ vérifient $\log(|a_i|) + \log(|b_i|) \leqslant 6|V_P|(\log(3|V_P|) + \log c_{\max})$, où $c_{\max} = \max(\max_{i,j,k,l}(c_{i,j}\, data_{k,l}), \max_i w_i)$.

De plus, si on obtient la solution optimale sous cette forme, comme le nombre d'allocations (et la taille de leurs poids) est polynomial en la taille de la plate-forme, alors le Théorème 3.4 nous permet de reconstruire, en temps polynomial en la taille de la plate-forme, un ordonnancement périodique de débit optimal.

4.1.3 Calcul du débit optimal

La formulation de la solution avec le programme linéaire que nous avons vu au paragraphe précédent n'est pas très pratique (même si ce programme linéaire peut être résolu en utilisant la méthode des ellipsoïdes, mais nous essaierons de garder celle-ci pour les cas désespérés, comme nous le verrons au Chapitre 5, Paragraphe 5.4). Pour trouver le débit optimal, nous allons procéder en deux étapes :

- on va chercher à déterminer le débit optimal en décrivant l'activité des processeurs, mais en remplaçant les allocations par de simples lois de conservation (ce qui nous conduira à la résolution d'un programme linéaire).
- à partir de la solution du programme linéaire, nous construirons un ensemble pondéré d'allocations qui correspond à l'activité des processeurs
- enfin, comme précédemment, nous utiliserons le Théorème 3.4 pour reconstruire un ordonnancement optimal.

4.1.3.1 Obtention du programme linéaire

Pour toute arête $e_{k,l} : T_k \to T_l$,

$$e_{k,l} : T_k \to T_l \in \{e_{begin,1} : T_{begin} \to T_1, e_{1,end} : T_1 \to T_{end}\}$$

du graphe de tâches et pour chaque paire de processeurs $P_i \to P_j$, on note $s(P_i \to P_j, e_{k,l})$ le temps moyen passé par P_i à envoyer des données de type $e_{k,l}$ au processeur P_j. Le nombre $s(P_i \to P_j, e_{k,l})$ est donc un rationnel positif. On note également $sent(P_i \to P_j, e_{k,l})$ le nombre moyen de fichiers de type $e_{k,l}$ envoyés de P_i à P_j par unité de temps. On obtient donc la relation

$$s(P_i \to P_j, e_{k,l}) = sent(P_i \to P_j, e_{k,l}) \times (data_{k,l} \times c_{i,j}) . \tag{4.1}$$

Pour chaque tâche T_k,

$$T_k \in \{T_{begin}, T_1, T_{end}\}$$

et chaque processeur P_i, on note $\alpha(P_i, T_k)$ le temps moyen passé à traiter des tâches de type T_k sur le processeur P_i et $cons(P_i, T_k)$ le nombre moyen de tâches de type T_k traitées par unité de temps sur le processeur P_i. $\alpha(P_i, T_k)$ et $cons(P_i, T_k)$ sont des nombres positifs et on a la relation suivante

$$\alpha(P_i, T_k) = cons(P_i, T_k) \times w_{i,k}. \tag{4.2}$$

Les quantités introduites ci-dessus doivent vérifier les contraintes suivantes, pour assurer les contraintes de ressource dans le motif appauvri.

Activité sur une période unitaire Les fractions de temps passées par les différents processeurs à calculer ou à communiquer, doivent appartenir à l'intervalle $[0, 1]$ puisque ces quantités correspondent à une activité moyenne pendant une unité de temps :

$$\forall P_i, \forall T_k \in V_A, 0 \leqslant \alpha(P_i, T_k) \leqslant 1 \tag{4.3}$$

$$\forall P_i, P_j, \forall e_{k,l} \in E_A, 0 \leqslant s(P_i \to P_j, e_{k,l}) \leqslant 1 \tag{4.4}$$

Modèle 1-port en sortie Les émissions du processeur P_i devant être effectuées séquentiellement vers ses voisins, on a l'équation suivante :

$$\forall P_i, \sum_{P_i \to P_j} \sum_{e_{k,l} \in E_A} s(P_i \to P_j, e_{k,l}) \leqslant 1, \tag{4.5}$$

Modèle 1-port en entrée Les réceptions du processeur P_i devant être effectuées séquentiellement, on a l'équation suivante :

$$\forall P_i, \sum_{P_j \to P_i} \sum_{e_{k,l} \in E_A} s(P_j \to P_i, e_{k,l}) \leqslant 1 \tag{4.6}$$

Recouvrement des calculs et des communications En raison des hypothèses faites sur le recouvrement, il n'y a pas d'autres contraintes sur $\alpha(P_i, T_k)$ que

$$\forall P_i, \sum_{T_k \in V_A} \alpha(P_i, T_k) \leqslant 1 \tag{4.7}$$

Pour assurer que le motif appauvri permet bien de traiter des tâches, il est nécessaire d'ajouter des lois de conservation, qui permettent d'assurer qu'à toute tâche traitée correspond un fichier d'entrée transmis depuis P_1 et un fichier de sortie transmis à P_1.

Soit P_i un processeur et $e_{k,l}$ une arête du graphe de tâches. En une unité de temps, P_i reçoit de ses voisins un certain nombre de fichiers de type $e_{k,l}$: $\sum_{P_j \to P_i} sent(P_j \to P_i, e_{k,l})$ exactement. Le processeur P_i exécute certaines tâches T_k lui même et génère donc autant de fichiers de type $e_{k,l}$. Qu'arrive-t-il à ces fichiers ? Certains sont envoyés aux voisins de P_i et d'autres sont utilisés par P_i pour traiter des tâches de type T_l. On en déduit l'équation suivante :

$$\forall P_i, \forall e_{k,l} : T_k \to T_l \in E_A,$$
$$\sum_{P_j \to P_i} sent(P_j \to P_i, e_{k,l}) + cons(P_i, T_k) = \sum_{P_i \to P_j} sent(P_i \to P_j, e_{k,l}) + cons(P_i, T_l) \tag{4.8}$$

Les équations précédentes forment un programme linéaire dont l'objectif est de maximiser le débit de la plate-forme, c'est-à-dire le nombre de tâches T_{end} consommées par unité de temps :

$$\begin{array}{l}
\text{MAXIMISER } \rho = \sum_{i=1}^{p} cons(P_i, T_{end}), \\
\text{SOUS LES CONTRAINTES} \\
\left\{ \begin{array}{ll}
\text{(4.9a)} & \forall P_i, \forall T_k \in V_A, 0 \leqslant \alpha(P_i, T_k) \leqslant 1 \\
\text{(4.9b)} & \forall P_i \to P_j, \forall e_{k,l} \in E_A, 0 \leqslant s(P_i \to P_j, e_{k,l}) \leqslant 1 \\
\text{(4.9c)} & \forall P_i \to P_j, \forall e_{k,l} \in E_A, s(P_i \to P_j, e_{k,l}) = sent(P_i \to P_j, e_{k,l}) \times (data_{k,l} \times c_{i,j}) \\
\text{(4.9d)} & \forall P_i, \forall T_k \in V_A, \alpha(P_i, T_k) = cons(P_i, T_k) \times w_{i,k} \\
\text{(4.9e)} & \forall P_i, \sum_{P_j \to P_i} \sum_{e_{k,l} \in E_A} s(P_j \to P_i, e_{k,l}) \leqslant 1 \\
\text{(4.9f)} & \forall P_i, \sum_{P_i \to P_j} \sum_{e_{k,l} \in E_A} s(P_i \to P_j, e_{k,l}) \leqslant 1 \\
\text{(4.9g)} & \forall P_i, \sum_{T_k \in V_A} \alpha(P_i, T_k) \leqslant 1 \\
\text{(4.9h)} & \forall P_i, \forall e_{k,l} \in E_A : T_k \to T_l, \\
& \qquad \sum_{P_j \to P_i} sent(P_j \to P_i, e_{k,l}) + cons(P_i, T_k) = \\
& \qquad\qquad \sum_{P_i \to P_j} sent(P_i \to P_j, e_{k,l}) + cons(P_i, T_l)
\end{array} \right.
\end{array} \tag{4.9}$$

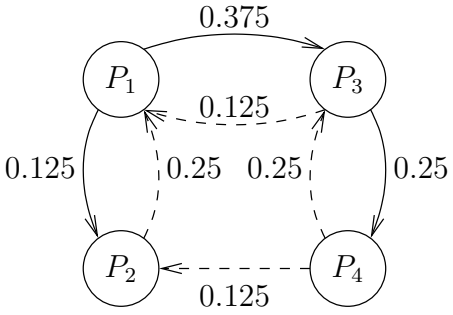
4.1.3.2 Construction des allocations

Le programme linéaire (4.9) nous fournit, pour chaque processeur et chaque type de tâche, le nombre fractionnaire de tâches traitées par ce processeur ou envoyées à ses voisins. Considérons l'exemple de la figure 4.1.

La solution du programme linéaire (4.9) est résumée à la Figure 4.2. Le débit $\rho = 0.525$ pourra être atteint si on arrive à exhiber une description compacte d'un K-motif appauvri de longueur T_p tel que $\rho = K/T_p$, ce

	$\alpha(P_i, T_1)$	$cons(P_i, T_1)$
P_1	100%	0.025
P_2	100%	0.125
P_3	100%	0.125
P_4	100%	0.250
Total	21 tâches toutes les 40 secondes	

(a) Répartition des calculs



(b) Communications

FIG. 4.2 – Solution du programme linéaire : le graphe de plate-forme est annoté avec les valeurs non-nulles des $sent(P_i \to P_j, e_{k,l})$; les flèches continues représentent des transferts de type $e_{begin,1}$ et les flèches pointillées des transferts de type $e_{1,end}$.

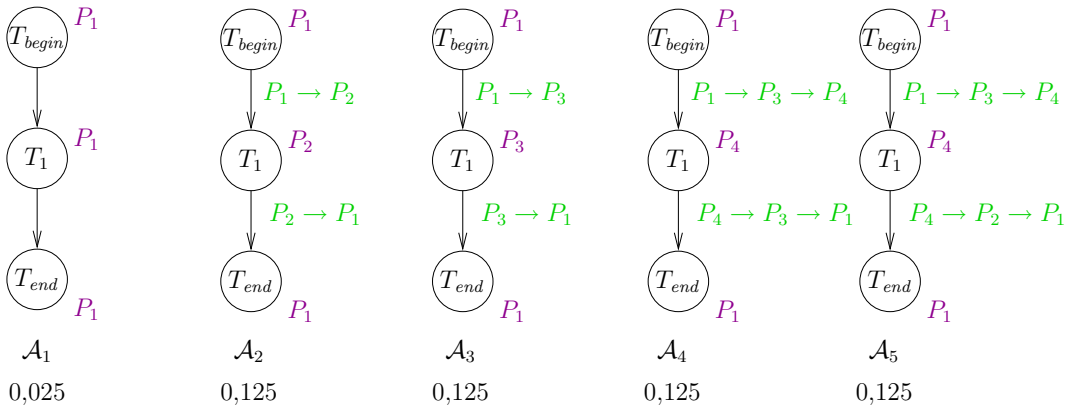


FIG. 4.3 – Décomposition de la solution du programme linéaire en 5 allocations $\mathcal{A}_1$,...,$\mathcal{A}_5$. Chacune de ces allocations participe pour une certaine fraction au régime permanent et la somme de leur contribution $(0.025 + 0.125 + 0.125 + 0.125 + 0.125)$ est égale au débit total $(0.525 = \frac{21}{40})$.

qui nécessite de décomposer la solution du programme linéaire en une somme pondérée d'allocations $(\alpha_i, \mathcal{A}_i)$ (voir figure 4.3).

Dans le cas où G_A est simplement constitué d'une tâche principale, d'une tâche T_{begin} et d'une tâche T_{end}, on peut montrer [70] que la décomposition en allocations peut être obtenue très facilement en "épluchant" le graphe (voir Algorithme 4.1).

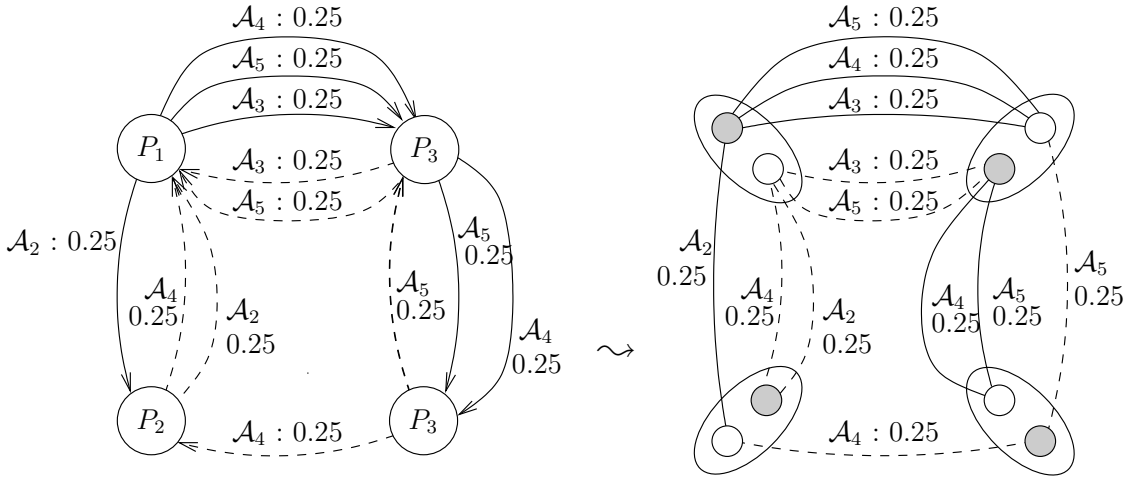
Decomposition en Allocations :
Soit P_i tel que $cons(P_i, T_{begin}) = 0$
$\pi(T_{begin}) \leftarrow P_i$.
$src \leftarrow i$
$\gamma_1 \leftarrow \emptyset$
Tant que $cons(P_{src}, T_1) = 0$ **:**
  Soit P_j tel que $sent(P_{src} \rightarrow P_j, e_{begin,1}) > 0$
  $\gamma_1 \leftarrow \gamma_1 \cup (P_{src} \rightarrow P_j)$
  $src \leftarrow j$
$\sigma(e_{begin,1}) \leftarrow \gamma_1$
$\pi(T_1) \leftarrow P_{src}$
$\gamma_2 \leftarrow \emptyset$
Tant que $cons(P_{src}, T_{end}) = 0$ **:**
  Soit P_j tel que $sent(P_{src} \rightarrow P_j, e_{1,end}) > 0$
  $\gamma_2 \leftarrow \gamma_2 \cup (P_{src} \rightarrow P_j)$
  $src \leftarrow j$
$\sigma(e_{1,end}) \leftarrow \gamma_2$
$\pi(T_{end}) \leftarrow P_{src}$
$\alpha \leftarrow \min\big(cons(P_{begin}, T_{\pi(T_{begin})}), cons(P_1, T_{\pi(T_1)}), cons(P_{end}, T_{\pi(T_{end})}),$
$\{sent(P_i \rightarrow P_j, e_{begin,1}) | P_i \rightarrow P_j \in \sigma(e_{begin,1})\},$
$\{sent(P_i \rightarrow P_j, e_{1,end}) | P_i \rightarrow P_j \in \sigma(e_{1,end})\}\big)$
Renvoyer(α, π, σ)

Algorithme 4.1: Algorithme d'extraction d'une allocation

L'Algorithme 4.1 parcourt G_A en profondeur et sélectionne gloutonnement les processeurs capables d'effectuer les différentes tâches. Les équations de conservation nous garantissent qu'une telle allocation peut être trouvée puisque si une tâche est consommée, elle produit un fichier de sortie qui est transmis à un autre processeur. Une fois une allocation valide déterminée, on détermine son poids en prenant le minimum des différentes quantités $cons(P_i, T_k)$ et $sent(P_i \rightarrow P_j, e_{k,l})$ impliquées dans l'allocation. En soustrayant ce poids à ces différentes quantités, on obtient des variables qui vérifient toujours les contraintes (4.9) et on peut donc recommencer jusqu'à ce que plus aucune tâche ne soit consommée.

Il est facile de montrer (en utilisant les formules de Cramer comme dans la preuve du Théorème 3.1) que la taille des coefficients $cons(P_i, T_k)$ et $sent(P_i \rightarrow P_j, e_{k,l})$ est polynomiale en la taille de la plate-forme. De plus, dans l'Algorithme 4.1, au moins une valeur non nulle de $cons(P_i, T_k)$ et $sent(P_i \rightarrow P_j, e_{k,l})$ est annulée à chaque étape (i.e. à chaque nouvelle allocation). On peut donc facilement vérifier que le nombre d'allocations est inférieur à $|V_P| + 2 + 2|E_P|$ (puisque les seules valeurs qui peuvent être non nulles de $cons(P_i, T_k)$ et $sent(P_i \rightarrow P_j, e_{k,l})$ sont $cons(P_1, T_{begin})$, $cons(P_1, T_{end})$, $cons(P_1, T_1)$, $cons(P_i, T_1)$, $sent(P_i \rightarrow P_j, e_{begin,1})$ et $sent(P_i \rightarrow P_j, e_{1,end})$).

Il est intéressant de noter que cette solution peut conduire à utiliser plus de schémas d'allocations ($|V_P| + 2 + 2|E_P|$) que dans la solution optimale ($3|V_P|$). Toutefois, le nombre d'allocations obtenu est polynomial en la taille du graphe de plate-forme, ce qui suffisant pour notre propos, comme nous le verrons au paragraphe suivant.



(a) Solution du programme linéaire

(b) Graphe biparti associé

FIG. 4.4 – Graphe des communications et graphe biparti associé.

4.1.3.3 Construction de l'ordonnancement périodique

On a obtenu une décomposition du motif-appauvri de débit optimal ρ comme une somme pondérée d'allocations. Le nombre d'allocations (et la taille de leurs coefficients) est polynomial en la taille de la plate-forme, donc on peut utiliser le Théorème 3.4 pour construire un ordonnancement périodique de débit optimal. Les différentes étapes (construction du graphe biparti, décomposition en couplage et construction de l'ordonnancement périodique) sont présentées sur l'exemple aux Figures 4.4, 4.5 et 4.6.

4.1.4 Tâches indépendantes élémentaires sur un arbre

Le lecteur peut être étonné de trouver ici un paragraphe consacré aux arbres, alors que le problème de la distribution de tâches élémentaires a été résolu au paragraphe précédent. En fait, dans le cas des arbres, on peut utiliser des techniques beaucoup plus élémentaires pour déterminer un ordonnancement périodique de débit optimal. La simplicité de ces techniques a permis de déterminer un algorithme dirigé par la demande, donc décentralisé et potentiellement plus robuste vis à vis de petites perturbations. Nous insisterons plus particulièrement sur ce point, qui est d'un grand intérêt pratique, mais pour lequel nous n'avons pas su prouver des propriétés de d'optimalité ou de robustesse. Pour la démonstration du reste des résultats, nous invitons le lecteur à se reporter à [6].

Considérons tout d'abord le cas de la plate-forme en étoile représentée à la Figure 4.7.

Dans le cas d'une étoile, on peut résoudre explicitement le programme linéaire exprimant les contraintes de ressource pour le motif. Le résultat est exprimé dans le lemme suivant.

Lemme 4.1. *Pour obtenir le débit optimal sur une étoile*

1. *Trier les esclaves par temps de communication croissant et les numéroter de façon à ce que* $c_1 \leqslant c_2 \ldots \leqslant c_k$.
2. *Soit* q *le plus grand indice tel que* $\sum_{i=1}^{q} \frac{c_i}{w_i} \leqslant 1$. *Les esclaves* $P_1, \ldots, P_q$ *calculeront en permanence. Si* $q < k$ *définissons* $\varepsilon = 1 - \sum_{i=1}^{q} \frac{c_i}{w_i}$, *la fraction de bande passante non utilisée par les* q *premiers processeurs, et sinon posons* $\varepsilon = 0$.

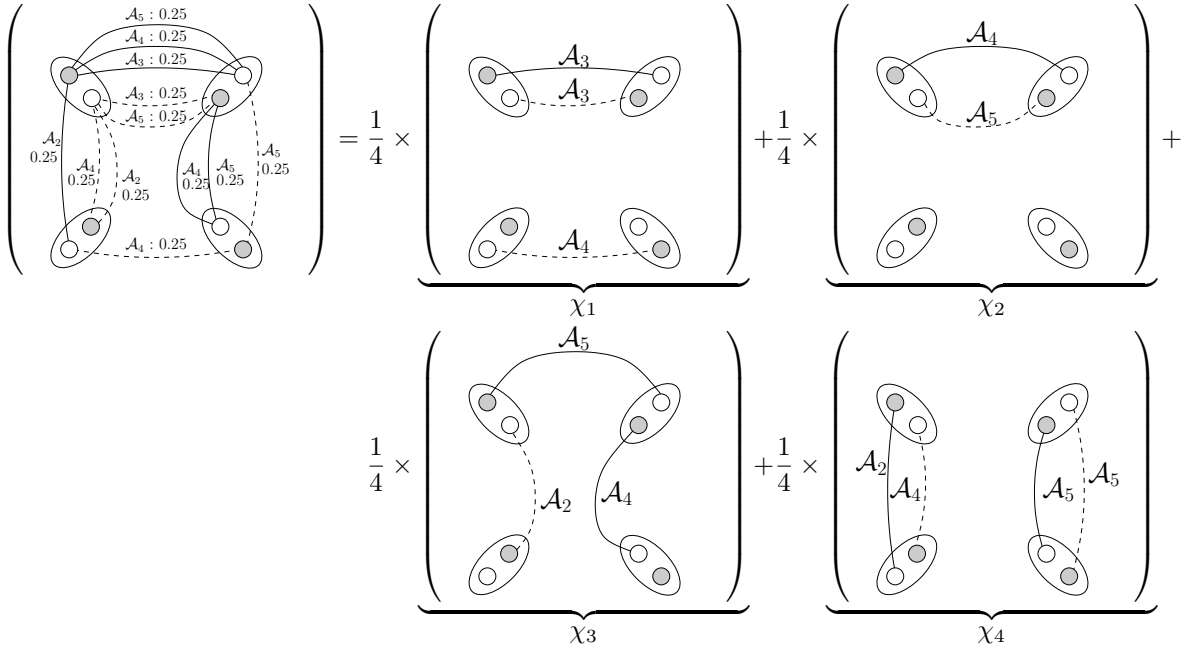


FIG. 4.5 – Décomposition du graphe biparti en somme de couplages pondérés

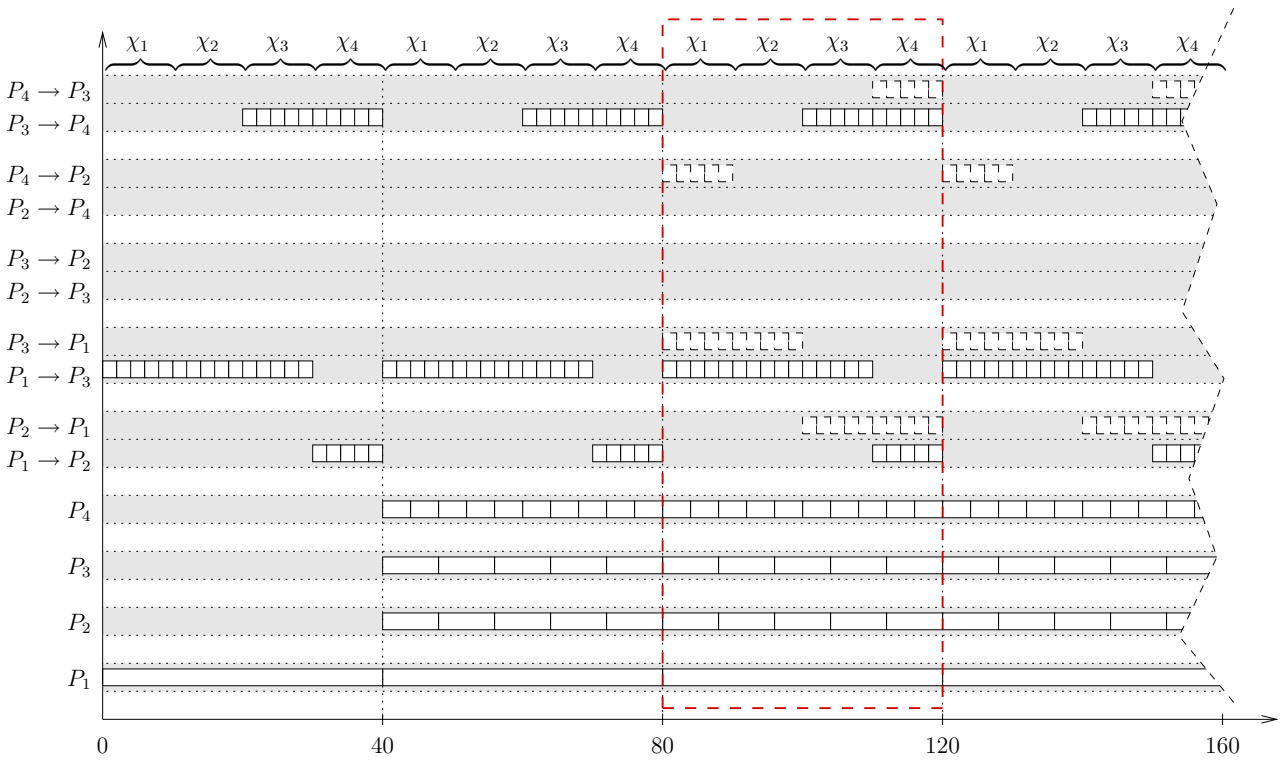


FIG. 4.6 – Ordonnancement atteignant le régime permanent optimal

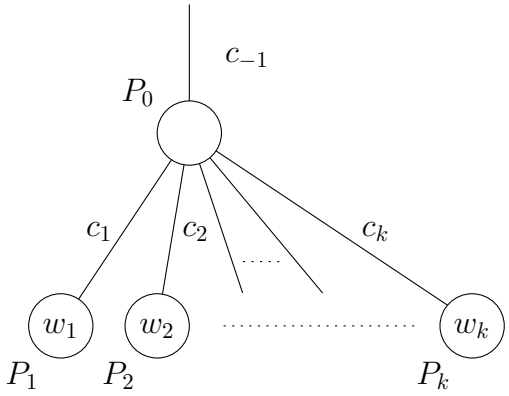


FIG. 4.7 – Graphe en étoile.

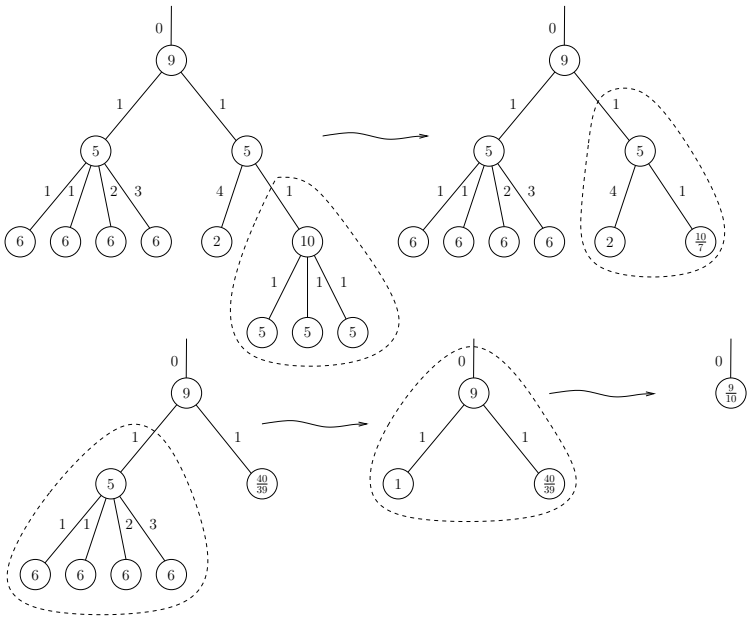


FIG. 4.8 – Calcul du régime permanent optimal par réductions successives

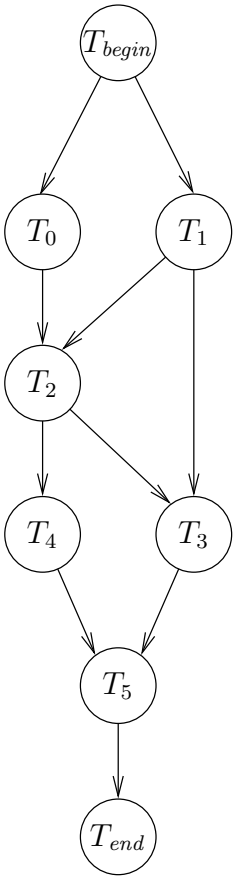
3. Alors

$$\rho = \min\left(\frac{1}{c_{-1}},\ \frac{1}{w_0} + \sum_{i=1}^{q} \frac{1}{w_i} + \frac{\varepsilon}{c_{q+1}}\right) \tag{4.10}$$

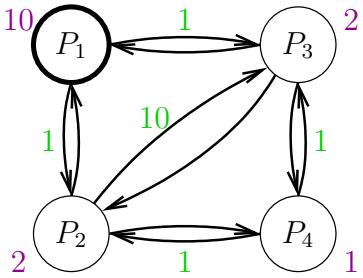
Le Lemme 4.1 conduit à trois propriétés essentielles

1. Le maître doit choisir les esclaves en fonction de leur proximité (c_i) et pas en fonction de leur vitesse de calcul (w_i).
2. Il n'existe que trois catégories d'esclaves : ceux qui travaillent tout le temps ($i \leqslant q$), ceux qui ne travaillent jamais ($i > q+1$) et celui (éventuellement) qui travaille partiellement.
3. L'étoile peut être remplacée, du point de vue de la puissance de calcul, par un seul nœud, relié à son père par un lien de débit c_{-1} et de puissance $\frac{1}{\rho}$.

La troisième propriété permet facilement de calculer, par réductions successives le débit optimal de la plate-forme (voir Figure 4.8), ainsi que l'ensemble des processeurs de la plate-forme qui doivent être utilisés en permanence, jamais ou partiellement. Nous verrons dans le Paragraphe 4.3 que la deuxième propriété permet de construire des ordonnancements dynamiques.



(a) Graphe de tâches



(b) Graphe de plate-forme : Les fichiers d'entrée $e_{begin,1}$ sont placé sur P_1 à l'origine et les fichiers de sortie $e_{1,end}$ doivent être rassemblés sur P_1

FIG. 4.9 – Exemple simple de modélisation d'une application constituée de tâches indépendantes complexes à exécuter sur une plate-forme hétérogène.

4.2 Problèmes indépendants complexes

4.2.1 Introduction

Nous considérons maintenant le problème de déterminer le débit optimal, quand les problèmes indépendants sont donnés sous la forme d'un graphe de tâche général (voir Figure 4.9).

Dans ce cas, les différentes tâches d'un même problème peuvent être allouées à des processeurs différents. Taura et Chien [95] ont étudié le même problème en essayant de déterminer, pour chaque type de tâche T_k, un processeur P_i sur lequel exécuter toutes les tâches de ce type. Ils ont montré que ce problème est NP-Complet. Dans les paragraphes qui suivent, nous considérons le problème différemment, en cherchant une solution sous la forme d'une somme pondérée d'allocations (comme définies au Paragraphe 3.3). Dans ces conditions, il est nécessaire de vérifier, comme nous l'avons fait au Paragraphe 4.1.2 que le problème a du sens, c'est à dire qu'il admet une solution optimale encodable de façon polynomiale en la taille des données. Comme précédemment, si on cherche la solution sous la forme d'allocations pondérées $(\beta_1, \mathcal{A}_1), \ldots, (\beta_Q, \mathcal{A}_Q)$,

alors un motif appauvri de débit optimal et de longueur 1 est donné par la solution du programme linéaire

Maximiser $\sum_q \beta_q$,
sous les contraintes

$$\begin{cases} \forall P_i, & \displaystyle\sum_q \sum_{k,\ \pi(T_k,\mathcal{A}_q)=P_i} \beta_k w_{i_k} \leqslant 1 & & \text{calcul sur} P_i \\ \forall P_i, & \displaystyle\sum_{j,\ (P_i,P_j)\in E_P} \Big(\sum_q \sum_{(T_k,T_l),\ \sigma(e_{k,l},\mathcal{A}_q)\ni(P_i,P_j)} \beta_q c_{i,j}\, data_{k,l} \leqslant 1 & & \text{communications entrantes en } P_i \\ \forall P_i, & \displaystyle\sum_{j,\ (P_j,P_i)\in E_P} \Big(\sum_q \sum_{(T_k,T_l),\ \sigma(e_{k,l},\mathcal{A}_q)\ni(P_j,P_i)} \beta_q c_{j,i}\, data_{k,l} \leqslant 1 & & \text{communications sortantes de } P_i \\ \beta_k \geqslant 0 & & & \text{poids des allocations} \end{cases}$$

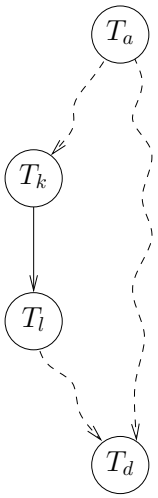
En appliquant le Théorème 3.1, on vérifie qu'il existe une solution optimale sous la forme $(\beta_1, \mathcal{A}_1), \ldots, (\beta_Q, \mathcal{A}_Q)$, composée d'au plus $3|V_P|$ allocations, dont les poids $\beta_q = \frac{a_q}{b_q}$ vérifient $\log(|a_q|) + \log(|b_q|) \leqslant 6|V_P|(\log(3|V_P|) + \log c_{\max})$, où $c_{\max} = \max(\max_{i,j,k,l}(c_{i,j}\, data_{k,l}), \max_i w_i)$.
En appliquant le Théorème 3.4 sur une telle solution, on vérifie qu'il est possible de déterminer en temps polynomial en la taille des données un ordonnancement périodique réalisant ce débit optimal.

Malheureusement, contrairement au cas de graphes de tâches élémentaires (constitués d'une seule tâche de calcul), il n'est pas possible de reformuler le programme linéaire ci dessus pour le résoudre en temps polynomial (nous verrons en effet que la détermination du débit optimal est un problème NP-Complet au Paragraphe 4.2.3).

4.2.2 Cas des graphes de tâches de profondeur bornée

Dans le cas des graphes de tâches élémentaires, nous avons montré que le programme linéaire 4.9 fournissait (en temps polynomial) le débit optimal d'un motif appauvri de longueur 1. Pour comprendre la difficulté induite par les graphes généraux, appliquons le programme linéaire 4.9 au graphe de tâches et au graphe de plate-forme décrits à la Figure 4.10. Dans cet exemple, tous les temps de calcul sont unitaires et les temps de communication sont égaux à $\frac{1}{2}$, mais chaque processeur ne peut traiter qu'un seul type de tâches (indiqué sur le graphe de plate-forme) et les liens sont unidirectionnels (comme indiqué sur le graphe de plate-forme).

La résolution du programme linéaire 4.9 fournit un débit égal à 1. Ce résultat est très optimiste puisqu'on peut facilement remarquer (à moins de dupliquer les tâches, ce qui sort du cadre de cet étude), qu'on ne peut en fait traiter aucun problème avec cette plate-forme ! En effet, il n'existe pas de plongement du graphe de tâches dans le graphe de plate-forme. En fait, le programme linéaire est incorrect puisqu'il accepte de traiter des tâches T_4 sur P_7, à partir de fichiers de sortie de tâches T_3 traitées sur P_3 et de tâches T_2 traitées sur P_6. Or ces tâches sont incompatibles puisqu'elles proviennent nécessairement de fichiers de sortie de tâches T_1 traitées respectivement sur P_1 et P_2. Il est donc nécessaire de se souvenir des lieux de traitement de certaines tâches, pour déterminer si deux fichiers de sortie sont compatibles. J'invite le lecteur à consulter [12], où il trouvera la preuve des résultats suivants.

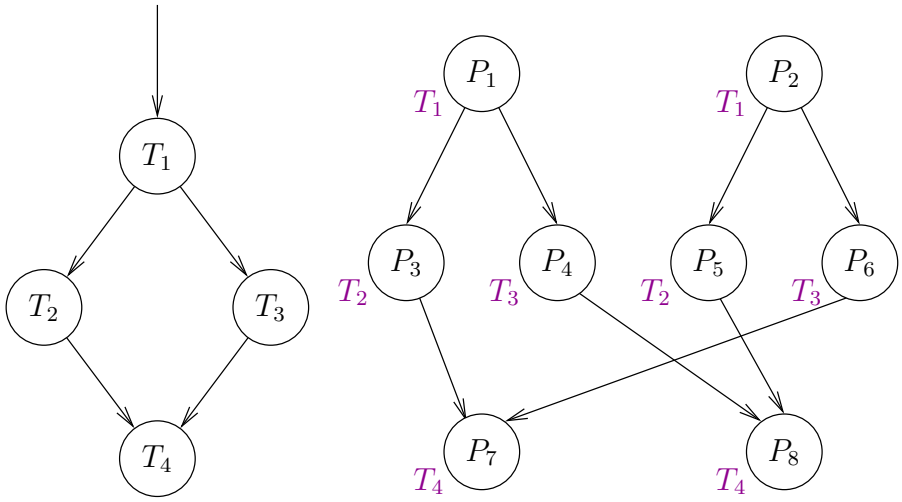


Pour assurer la compatibilité des fichiers de sortie, on annote les variables $sent(P_i \to P_j, e_{k,l})$, $s_i$jkl, $\alpha(P_i, T_k)$ et $cons(P_i, T_k)$ par une liste de contraintes, qui représente les lieux d'exécution de certains ancêtres dans le graphe de tâches.

Plus précisément, pour toute dépendance $e_{k,l}$ on dira qu'une tâche T_a est contraignante pour $e_{k,l}$ si T_a est un ancêtre de T_l (ce peut donc être T_k) et s'il existe un T_d (qui peut être T_l) tel que :

- T_d est un descendant de T_l,
- il y a un chemin de T_a à T_d sommets-disjoint (exceptés T_a et T_d) de celui allant de T_a à T_d en passant par T_l.

Une fois le calcul des tâches contraignantes réalisé, il est possible de calculer le débit optimal en résolvant un programme linéaire analogue à 4.9 dans lequel les variables $sent(P_i \to P_j, e_{k,l})$, $s_i$jkl, $\alpha(P_i, T_k)$ et $cons(P_i, T_k)$ sont annotées par des listes de contraintes, et dans lequel la compatibilité des fichiers de sortie est garantie. Il est alors possible de décomposer



(a) Graphe de l'application

(b) Graphe de plate-forme : chaque processeur ne peut traiter qu'un seul type de tâche

FIG. 4.10 – Contre-exemple

la solution du programme linéaire en une somme pondérée d'allocations, et l'application du Théorème 3.4 nous fournit un ordonnancement périodique de débit optimal.

Malheureusement, pour un graphe de tâches général, la taille des listes de contraintes peut être grande. Considérons par exemple le graphe de tâches représenté à la Figure 4.11. Chaque arête du graphe de tâches est annotée par son nombre de tâches contraignantes. Dans ce cas, le nombre de variables du programme linéaire ($p^2n^2p^n$) à résoudre n'est pas polynomial en la taille des données.

Néanmoins, si on définit la profondeur de dépendance comme le nombre maximal de tâches contraignantes d'une dépendance $e_{k,l}$ (sur l'ensemble des $e_{k,l}$), on peut démontrer le théorème suivant.

Définition 4.1. *Débit-Opt-Profondeur-Bornée$_d$(G_a, G_p) : Étant donnés un graphe de tâches G_a dont la profondeur de dépendance est bornée par d et un graphe de plate-forme G_p, trouver un ordonnancement périodique de débit optimal.*

Théorème 4.1. *Pour tout $d \in \mathbb{N}$, Débit-Opt-Profondeur-Bornée$_d$ peut être résolu en temps polynomial.*

En effet, si on borne la taille des listes de contraintes, alors la taille du programme linéaire est polynomiale en la taille des données (le nombre de variables est alors borné par $p^2n^2p^d$) et on peut décomposer la solution en allocations, puis appliquer le Théorème 3.4 pour obtenir un ordonnancement périodique de débit optimal.

4.2.3 Cas général

Considérons le problème général de la maximisation du débit pour un graphe de tâches et un graphe de plate-forme quelconques :

Définition 4.2. *Débit-Opt(G_a, G_p, ρ) : Étant donnés un graphe de tâches G_a, un graphe de plate-forme G_p, et une constante ρ, existe-t-il un ordonnancement périodique de débit supérieur à ρ ?*

Une telle formulation n'appartient pas *a priori* à NP, puisque le nombre d'allocations (et leurs poids) utilisées dans la solution optimale peut être exponentiel en la taille des données.

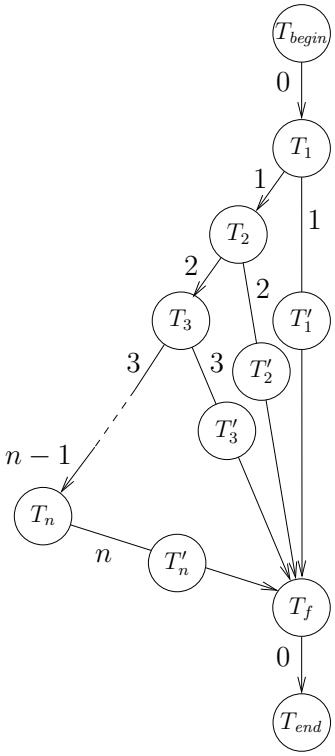


FIG. 4.11 – Tous les lieux de traitement des tâches T_1,...,T_n doivent être mémorisées pour assurer la compatibilité en T_f.

Néanmoins, nous avons démontré au Paragraphe 4.2.1 (en utilisant le Théorème 3.1) que s'il existe une solution à Débit-Opt(G_a, G_p, ρ), alors il existe une solution sous la forme $(\beta_1, \mathcal{A}_1), \ldots, (\beta_Q, \mathcal{A}_Q)$, composée d'au plus $3|V_P|$ allocations, dont les poids $\beta_q = \frac{a_q}{b_q}$ vérifient $\log(|a_q|)+\log(|b_q|) \leqslant 6|V_P|(\log(3|V_P|)+\log c_{\max})$, où $c_{\max} = \max(\max_{i,j,k,l}(c_{i,j}\, data_{k,l}), \max_i w_i)$.

On peut donc considérer le problème de décision suivant, qui appartient à NP, dont toute solution est également une solution de Débit-Opt(G_a, G_p, ρ), et qui admet une solution si et seulement si Débit-Opt(G_a, G_p, ρ) admet une solution (d'après le Théorème 3.1)

Définition 4.3. *Débit-Opt-Compact(G_a, G_p, ρ) : Étant donnés un graphe de tâches G_a, un graphe de plate-forme G_p, et une constante ρ, existe-t-il un ordonnancement périodique de débit supérieur à ρ, composé d'au plus $3|V_P|$ allocations, dont les poids $\beta_q = \frac{a_q}{b_q}$ vérifient $\log(|a_q|)+\log(|b_q|) \leqslant 6|V_P|(\log(3|V_P|)+\log c_{\max})$, où $c_{\max} = \max(\max_{i,j,k,l}(c_{i,j}\, data_{k,l}), \max_i w_i)$?*

Théorème 4.2. *Débit-Opt-Compact(G_a, G_p, K) est NP-Complet au sens fort.*

La preuve de ce résultat peut être trouvée dans [12]. Elle s'appuie sur une réduction à MINIMUM-MULTIWAY-CUT, qui est NP-complet (et même APX-complet) [4].

Définition 4.4 (MINIMUM-MULTIWAY-CUT(G_M, S, t, B)). *Étant donnés un graphe $G_M = (V_M, E_M)$, un ensemble $S \subset V_M$ de sommets distingués, une fonction de poids t sur les arêtes et une borne $B \in \mathbb{Q}$, existe-t-il une coupe multiple, i.e. un ensemble $E'_M \subset E_M$ tel que dans $G_M = (V_M, E_M \backslash E_M{}')$ chaque nœud distingué est déconnecté des autres nœuds distingués et*

$$\sum_{e \in E'_M} t(e) \leqslant B \ ?$$

4.3 Tâches indépendantes : conclusions, perspectives et problèmes ouverts

4.3.1 Résultats de complexité

Nous avons justifié le changement de métrique, de la minimisation du temps de complétion à la maximisation du débit, en arguant qu'il pourrait rendre les problèmes d'ordonnancement plus faciles. De ce point de vue, ce chapitre montre que les résultats sont très satisfaisants.

Dans le cas des tâches élémentaires, nous avons vu qu'il était possible de construire en temps polynomial un ordonnancement périodique de débit optimal sur une plate-forme quelconque, alors que Dutot [42] (voir Paragraphe 2.1) a démontré que le problème est NP-Complet au sens fort sur des arbres de profondeur 2, quand on considère la minimisation du temps de complétion (même sans communications retour). Il est également à noter que la taille du code produit pour organiser l'ordonnancement des tâches de calcul et de communication dépend de manière polynomiale de la taille de la plate-forme mais de manière logarithmique du nombre de tâches à traiter (NB^{iter} dans l'Algorithme 3.6).

Dans le cas des tâches complexes, nous avons également montré qu'il est possible de construire en temps polynomial un ordonnancement de débit optimal pour une large classe de graphes de tâches (les graphes de profondeur de dépendance bornée), résultat qui se compare très favorablement aux résultats de complexité associés à la minimisation du temps de complétion.

Il est également intéressant de noter que pour que les problèmes d'ordonnancement deviennent faciles quand on considère la maximisation du débit, il est nécessaire de chercher la solution sous la forme d'une combinaison linéaire d'allocations. Par exemple, dans le cas des tâches complexes, nous avons vu que Taura et Chien [95] ont montré que la recherche du meilleur placement de chaque tâche du graphe de tâche sur un processeur du graphe de plate-forme est un problème NP-Complet au sens fort du point de vue de la maximisation du débit, même dans le cas des graphes de tâche de profondeur bornée. Avec les notations du Chapitre 3, la recherche de ce placement correspond à la recherche du meilleur schéma d'allocation $\mathcal{A}$. Nous avons par contre démontré que la recherche de la meilleure combinaison linéaire d'allocation (qui produit un ordonnancement dont le débit est par construction supérieur) est un problème qui peut être résolu en temps polynomial. De façon analogue, dans le cadre de la maximisation du débit avec des tâches simples, on peut rechercher le meilleur arbre couvrant, pour lui appliquer les techniques que nous avons décrites au Paragraphe 4.1. Nous avons montré que la recherche du meilleur arbre est un problème NP-Complet (une preuve par réduction à 2-Partition peut être trouvée dans [6] et une autre, suggérée par Claire Hanen, par réduction à 3-Partition, dans la thèse d'Arnaud Legrand [70]). Ces résultats n'ont en fait rien de surprenant si on les compare aux résultats de théorie des graphes, où de nombreux problèmes NP-Complets (colorations...) peuvent être résolus en temps polynomial si on considère leur version fractionnaire (on pourra consulter [87, 78] pour trouver de nombreuses illustrations de cette propriété).

4.3.2 Vers des versions dynamiques ?

4.3.2.1 Tâches élémentaires sur des arbres

Dans le cas des tâches élémentaires sur des plates-formes en arbre, nous avons vu qu'il était possible de calculer une puissance équivalente à chaque sous arbre et qu'il était possible de classifier les nœuds de la plate-forme en trois catégories "qualitatives" (Théorème 4.1) : les nœuds qui doivent être utilisés tout le temps, ceux qui ne doivent jamais recevoir de tâches et ceux (celui en fait pour chaque arbre de profondeur 1) qui doivent être utilisés en complément.

Ces deux propriétés rendent le problème séduisant pour construire une version dynamique (et donc décentralisée, comme nous l'avons vu au Paragraphe 2.4) de l'ordonnancement. En effet, il est facile pour un nœud de la plate-forme de mesurer au cours du temps la distance à ses fils dans l'arbre et d'obtenir une évaluation de la puissance de leur sous-arbre (en observant la fréquence à laquelle ils demandent de nouvelles

tâches). Par ailleurs, la classification "qualitative" est naturellement assez stable par rapport à de petites variations dans les performances des ressources de calcul et de communication de la plate-forme.

Ces idées ont été analysées dans la thèse de Barbara Kreaseck [64]. Elle a proposé deux algorithmes "autonomes" de distribution des tâches élémentaires sur une plate-forme en arbre. Elle a montré que deux problèmes intéressants se posent quand on essaye d'adapter l'algorithme d'ordonnancement que nous avons vu à des schémas d'ordonnancement dans lesquels les processeurs doivent prendre les décisions de façon autonome (c'est à dire en ne communiquant qu'avec leurs voisins directs) :

- pour qu'une forme de régime permanent s'installe, il est nécessaire qu'en amont de l'algorithme autonome, chaque processeur possède un certain nombre de tâches.
- pour que le processeur qui travaille partiellement ne ralentisse pas le schéma de distribution, il est nécessaire de supposer qu'on peut interrompre les communications.

Le premier problème pose également la question de la mémoire nécessaire à l'application du protocole autonome (sur lequel nous reviendrons au Paragraphe 4.3.4). Intuitivement, si un des fils d'un processeur possède un lien relativement lent, pendant le déroulement de cette communication, il est nécessaire que les autres fils possède suffisamment de tâches pour pouvoir travailler tout le temps. Barbara Kreaseck a également démontré que ce problème peut être partiellement résolu si on suppose que les communications sont interruptibles, c'est à dire qu'elles peuvent se dérouler en plusieurs étapes.

Néanmoins, les schémas proposés manquent de fondements théoriques, même s'ils donnent de très bons résultats en pratique. En effet, la détermination du nombre minimal de tâches nécessaires à l'initialisation du protocole autonome, et de la quantité de mémoire nécessaire en chaque nœud sont pour l'instant des problèmes ouverts.

4.3.2.2 Tâches élémentaires sur des plates-formes quelconques

Quand on considère des plates-formes générales, il est beaucoup plus difficile de concevoir des algorithmes décentralisés. En effet, dans le cas d'une plate-forme en arbre, chaque nœud reçoit des tâches d'au plus un autre nœud, de sorte qu'il n'est pas nécessaire d'organiser l'ordonnancement des communications, puisque le récepteur est nécessairement prêt au moment de l'envoi. Dans le cas d'une plate-forme générale, la situation est très différente, puisqu'il est nécessaire de s'assurer, avant chaque communication que l'émetteur et le récepteur sont disponibles. Nous avons vu que théoriquement (Paragraphe 3.3), cette propriété est facile à assurer en décomposant le graphe de communications en somme pondérée de couplages. À notre connaissance, il n'existe pas d'algorithme décentralisé pour calculer une telle décomposition, et la décomposition en couplages paraît très sensible à des variations dans les performances des liens de communication. La conception d'un algorithme d'ordonnancement décentralisé et robuste à des variations de performance des ressources reste donc un problème ouvert.

Toutefois, on peut proposer certaines pistes possibles pour une implantation réelle de la distribution de tâches indépendantes sur une plate-forme générale (ces deux solutions nécessitent une résolution initiale du problème de la distribution sur une plate-forme idéale) :

- La première solution consiste à s'appuyer sur la décomposition de la solution en somme pondérée d'allocations. Chaque processeur connaît la fraction du temps qu'il doit consacrer à chacune des allocations et il utilise l'algorithme d'équilibrage 1D décrit dans [28] pour assurer qu'il consacre la fraction juste de son temps à chaque allocation. Pour organiser les communications, on peut s'appuyer sur un schéma de négociation, où émetteur et destinataire se mettent d'accord sur une plage pendant laquelle échanger leurs messages (cette organisation ne peut pas *a priori* conduire à une décomposition optimale en couplage, à moins de concevoir une version décentralisée de l'algorithme proposé dans [89, vol.A chapitre 20], ce qui ne paraît pas facile).
- La deuxième solution consiste à s'appuyer sur la décomposition en couplages du graphe de communication. En imposant des synchronisations fortes à la fin de chaque couplage (c'est à dire en imposant que chaque processeur renonce aux communications qu'il n'a pas eu le temps d'effectuer), on peut assurer que si les performances des liens ne changent pas, l'algorithme d'ordonnancement aura des performances satisfaisantes. Chaque processeur peut en outre, à partir du nombre de tâches qu'il n'a

pu transmettre à la fin de la période, estimer le nombre de tâches qu'il doit demander selon chacun des schémas d'allocation.
- On peut aussi (plus modestement) utiliser la décomposition en somme pondérée d'allocations comme heuristique pour déterminer le meilleur arbre de diffusion sur la plate-forme. Malheureusement, outre le fait que le problème de la détermination du meilleur arbre est NP-Complet, nous avons démontré [6] que le débit du meilleur arbre peut être arbitrairement mauvais par rapport au débit optimal (en utilisant toute la plate-forme).

Une solution élégante et totalement différente a été proposée par Bo Hong [58]. La modélisation des communications est très différente de celle que nous avons vu au Chapitre 1, puisque dans [58], un processeur peut émettre (et recevoir) un message vers (de) plusieurs processeurs au même instant. Dans ce cadre, il est possible d'exprimer la maximisation du débit comme un problème de flot maximal sur le graphe de la plate-forme et un algorithme décentralisé de calcul du flot et de distribution des tâches a également été proposé. Cette approche a l'avantage de conduire à un algorithme très élégant, mais l'inconvénient de s'appuyer sur une modélisation des communications très discutable. Toutefois, sur des plates-formes complexes, le problème de la distribution de tâches élémentaires est difficile, du point de vue d'une implantation réelle décentralisée et robuste et il serait intéressant de comparer, par la simulation les approches décrites dans ce paragraphe avec celle proposée dans [58] (malheureusement, ce travail n'a pas encore été réalisé).

4.3.3 Plusieurs maîtres, plusieurs types de tâches

Si on considère que le maître produit plusieurs types de fichiers d'entrée (disons des tâches de types T_1 et T_2 si on considère des tâches élémentaires ou des graphes de tâches G_{A1} ou G_{A2} si on considère des tâches complexes) dans des proportions fixées, il est facile de montrer que l'adaptation des algorithmes que nous avons vu dans ce chapitre ne pose pas de problème. Il suffit en effet, pour chaque ressource de calcul, de communications entrantes et de communications sortantes, d'écrire que le temps d'occupation de chaque ressource est la somme des temps consacrés à chaque type de problème, et que la fraction de problèmes de chaque type respecte les proportions initiales.

Par contre, même dans le cas de tâches simples sur une plate-forme en arbres, l'adaptation des protocoles autonomes développés par Barbara Kreaseck [64] n'a pas encore été réalisée. En effet, il n'est plus possible de représenter par un simple nombre la puissance équivalente d'un sous-arbre. En effet, il faut que celle ci exprime le temps de traitement pour toute combinaison de chaque type de tâches.

L'adaptation au cas où plusieurs maîtres produisent des fichiers d'entrée ne pose également pas de problème d'un point de vue théorique, puisqu'il suffit de dupliquer, dans les programmes linéaires, le nombre de nœuds maîtres particuliers. Du point de vue de l'implantation pratique, même dans le cas où la plate-forme est un arbre, le problème est plus complexe. En effet, dans ce cas plusieurs arbres (enracinés en chacun des maîtres) sont utilisés pour la distribution des tâches. Dans ce cas, chaque nœud de la plate-forme peut recevoir des tâches de plusieurs autres nœuds et il est alors nécessaire, comme nous l'avons vu au paragraphe précédent, d'organiser ces communications à l'aide de décompositions en couplages.

4.3.4 Et la mémoire ?

Dans le Chapitre 1, nous avons proposé une modélisation assez fine des capacités de calcul et de communication des différents processeurs de la plate-forme, mais nous avons complètement négligé leur capacité mémoire. Malheureusement, la construction des ordonnancements périodiques que nous avons proposé au Paragraphe 3.3 est très consommatrice de mémoire. En effet, nous avons vu qu'il est nécessaire d'augmenter la longueur du motif appauvri
- pour assurer qu'un nombre entier de communications se déroule pendant chaque couplage (introduction du facteur A)
- pour assurer qu'on peut obtenir un débit arbitrairement proche du débit optimal, même en présence de latences (introduction du facteur N)

Rappelons que dans les ordonnancements périodiques que nous avons construits au Paragraphe 3.3, les fichiers et les tâches reçus à l'étape i ne sont transmis ou traités (souvent, selon les valeurs de décalage calculées

	Arbres de profondeur 1 et pieuvres	Plates-formes quelconques
Min. Makespan avec contrainte mém.	NP Complet ([15])	NP Complet ([15])
Min. Makespan sans contrainte mém.	Polynomial [43]	NP-Complet [42]
Max débit avec contrainte mém.	NP-Complet ([15])	NP-Complete ([15])
Max débit sans contrainte mém.	Polynomial [10]	Polynomial [5]

FIG. 4.12 – Résultats de complexité en présence de contraintes mémoires

au Paragraphe 3.3) qu'à l'étape $i+1$, de sorte que l'augmentation de la taille de la période conduit à une augmentation des besoins en mémorisation.

Dans ce cadre, la limitation de la taille de la mémoire peut donc devenir un problème important. Il est donc intéressant de considérer les problèmes de maximisation du débit en présence de contraintes mémoires. Le lecteur trouvera dans [15] les travaux que nous avons mené sur ce problème. Malheureusement, tous les résultats obtenus sont négatifs, même sur des arbres de profondeur 1. Les résultats sont résumés dans le tableau de la Figure 4.12.

En présence de contraintes mémoires, il est donc nécessaire de concevoir des algorithmes d'approximation. Une solution simple consiste à limiter la taille de la période jusqu'à ce que les contraintes mémoires soient satisfaites (mais on perdra alors la propriété de débit optimal, puisque les latences ne seront plus négligeables et que la décomposition en couplages ne sera plus optimale). Le lecteur trouvera également dans[15] un algorithme d'approximation garanti à un facteur 2 dans le cas des arbres de profondeur 1, mais la prise en compte des contraintes mémoire reste (également) un problème largement ouvert.

Chapitre 5

Communications collectives

5.1 Introduction

Dans ce chapitre, nous considérons quelques opérations de communications collectives sur des plates-formes hétérogènes. Nous utiliserons le mode opératoire décrit dans le Chapitre 1, Paragraphe 1.3 pour les communications (modèle 1-port bidirectionnel), excepté au Paragraphe 5.4, dans lequel nous utiliserons le modèle 1-port unidirectionnel (dans lequel, à un instant donné, un processeur peut soit émettre, soit recevoir un message).

Comme dans le chapitre précédent, nous nous intéressons à la maximisation du débit et à la conception d'ordonnancements périodiques de débit optimal (nous considérons donc implicitement que les messages à transférer sont de très grande taille). Dans ce chapitre, nous considérerons la diffusion de messages (broadcast), dans lequel un nœud particulier envoie le même message à tous les autres nœuds (aux paragraphes 5.2 et 5.4) et la diffusion partielle (multicast), dans lequel un nœud particulier envoie le même message à un sous ensemble des nœuds de la plate-forme (au Paragraphe 5.3). D'autres opérations de communications collectives, comme la diffusion personnalisée (scatter) ou le regroupement (gather) peuvent s'exprimer en utilisant des graphes de tâches (voir Figure 5.1).

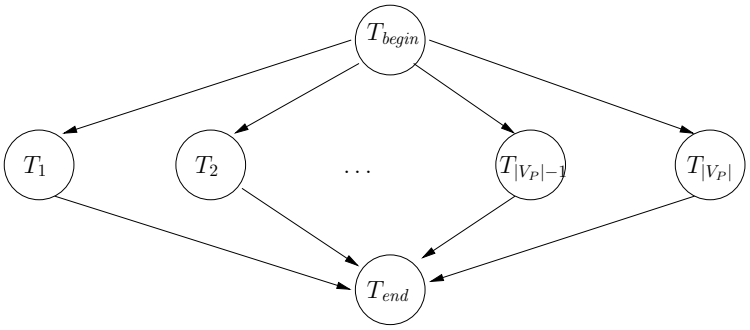


FIG. 5.1 – Graphe de tâches utilisés pour Scatter et Gather

Dans le cas de la diffusion personnalisée dans lequel le processeur P_0 doit envoyer un message de taille s_i à chacun de P_i, $\forall i$, on posera (pour le graphe de tâches décrit à la Figure 5.1) :

$$\begin{cases} \forall 1 \leqslant i \leqslant |V_P|, & data_{begin,i} = s_i \\ \forall 1 \leqslant i \leqslant |V_P|, & data_{i,end} = 0 \\ \forall 1 \leqslant i \leqslant |V_P|, & w_{i,begin} = +\infty, \quad w_{i,i} = w_{i,end} = 0, \quad w_{i,j\neq i} = +\infty \\ & w_{begin,0} = 0 \end{cases}$$

Ainsi, on impose une communication d'un message de taille s_i entre P_0 et P_i. Comme le graphe de tâches correspondant à l'opération de diffusion personnalisée est de profondeur de dépendance bornée (1), on peut

appliquer le Théorème 4.1 pour construire un ordonnancement périodique de débit optimal pour la diffusion personnalisée.

De la même manière, si on pose (pour le graphe de tâches décrit à la Figure 5.1)

$$\begin{cases} \forall 1 \leqslant i \leqslant |V_P|, & data_{begin,i} = 0 \\ \forall 1 \leqslant i \leqslant |V_P|, & data_{i,end} = s_i \\ \forall 1 \leqslant i \leqslant |V_P|, & w_{end,i} = +\infty, \quad w_{i,i} = w_{begin,i} = 0, \quad w_{i,j\neq i} = +\infty \\ & w_{end,0} = 0 \end{cases}$$

on peut appliquer le Théorème 4.1 pour construire un ordonnancement périodique de débit optimal pour le regroupement.

La situation est très différente dans le cas de la diffusion totale (broadcast) ou partielle (multicast), puisqu'il est nécessaire de tenir compte du fait que le même message est envoyé à tous les destinataires, ce qui ne peut pas être exprimé sous la forme d'un graphe de tâches. Il est intéressant de noter qu'en retour, les techniques développées dans ce chapitre peuvent être utilisées pour modéliser la situation où les fichiers de sortie correspondant à plusieurs dépendances $(T_k, T_{l_1}), \ldots, (T_k, T_{l_i})$ sont pour tout ou partie identiques. Malheureusement, il faudrait utiliser un schéma de diffusion partielle (multicast) pour communiquer avec les nœuds auxquels sont allouées les tâches $T_{l_1}, \ldots, T_{l_i}$ et nous verrons au Paragraphe 5.3 que ce problème est NP-Complet et mal approximable (APX-Complet).

5.2 Diffusion sous le modèle 1-port bidirectionnel

5.2.1 Formalisation

Dans le cadre de la diffusion, chaque bout de message doit être envoyé à tous les nœuds de la plate-forme. À partir de tout ordonnancement dans lequel une portion du message est reçue plusieurs fois en un même nœud, on peut construire un ordonnancement (de débit supérieur ou égal) dans lequel chaque portion du message n'est reçue qu'une seule fois en chaque nœud, de sorte qu'on peut se contenter de chercher la solution de débit optimale sous la forme d'un ensemble pondéré d'arbres (couvrants la plate-forme) de diffusion $(\alpha_1, \mathcal{T}_1), \ldots, (\alpha_Q, \mathcal{T}_Q)$.

Considérons un schéma quelconque (i.e. pas nécessairement périodique) de diffusion de messages, et notons $T(N)$ le temps nécessaire pour diffuser un message de taille N. Chacune des N parties (bits) du message est diffusée, comme nous venons de le voir, selon un arbre de diffusion. Pour chaque arbre de diffusion possible $\mathcal{T}_q$ (le nombre d'arbres possibles est fini), on note $\beta_q^{(N)}$ le nombre de messages élémentaires qui sont diffusés en utilisant l'arbre $\mathcal{T}_q$ parmi les N premiers messages (on a donc $\sum_q \beta_q^{(N)} = N$).

Les contraintes suivantes expriment les contraintes 1-port pour chacun des processeurs P_i :

$$\begin{cases} \forall P_i, & \displaystyle\sum_{j,\ (P_i,P_j)\in E_P} \sum_{q, \mathcal{T}_q \ni (P_i,P_j)} \beta_q^{(N)} c_i j \leqslant T(N) & \text{communications sortantes de } P_i \\ \forall P_i, & \displaystyle\sum_{j,\ (P_j,P_i)\in E_P} \sum_{q, \mathcal{T}_q \ni (P_j,P_i)} \beta_q^{(N)} c_j i \leqslant T(N) & \text{communications entrantes en } P_i \end{cases},$$

et donc

$$\begin{cases} \forall P_i, & \displaystyle\sum_{j,\ (P_i,P_j)\in E_P} \sum_{q, \mathcal{T}_q \ni (P_i,P_j)} \frac{\beta_q^{(N)}}{T(n)} c_i j \leqslant 1 & \text{communications sortantes de } P_i \\ \forall P_i, & \displaystyle\sum_{j,\ (P_j,P_i)\in E_P} \sum_{q, \mathcal{T}_q \ni (P_j,P_i)} \frac{\beta_q^{(N)}}{T(N)} c_j i \leqslant 1 & \text{communications entrantes en } P_i \end{cases}.$$

Comme au Paragraphe 4.1.2, considérons le programme linéaire suivant :

Diffusion-Max-Débit
$\text{Débit}_{\text{Opt}} = \textbf{Maximiser } \sum_q \alpha_q$,
sous les contraintes

$$\begin{cases} \forall P_i, & \displaystyle\sum_{j,\ (P_i,P_j)\in E_P} \ \sum_{q, \mathcal{T}_q \ni (P_i,P_j)} \alpha_q c_i j \leqslant 1 & \text{communications sortantes de } P_i \\ \forall P_i, & \displaystyle\sum_{j,\ (P_j,P_i)\in E_P} \ \sum_{q, \mathcal{T}_q \ni (P_j,P_i)} \alpha_q c_j i \leqslant 1 & \text{communications entrantes en } P_i \\ \alpha_q \geqslant 0 & & \text{poids des allocations} \end{cases}$$

Les variables $\frac{\beta_q^{(N)}}{T(N)}$ vérifient les conditions du programme linéaire ci dessus, de sorte que pour tout ordonnancement et pour toute taille de message N, on vérifie que

$$\sum_q \frac{\beta_q^{(N)}}{T(N)} = \frac{N}{T(N)} \leqslant \text{Débit}_{\text{Opt}}.$$

Le programme linéaire permet donc d'obtenir une borne supérieure sur le débit d'une opération de diffusion sur E_P.

Comme au Paragraphe 4.1.2, le programme linéaire ci-dessus n'est certainement pas facile à résoudre, mais néanmoins, il nous permet d'affirmer qu'il est raisonnable de se poser la question de la maximisation du débit. En effet, le programme linéaire contient $2|V_P|$ contraintes différentes de $\alpha_q \geqslant 0$, et à partir d'une allocation pondérée donnée sous la forme $(\alpha_1, \mathcal{T}_1), \ldots, (\alpha_Q, \mathcal{T}_Q)$, il est facile de vérifier en temps polynomial en Q (et la taille des coefficients des α_q) que les contraintes sur les communications sortantes et entrantes de chaque processeur sont bien vérifiées. On est donc dans les conditions d'application du Théorème 3.1, qui nous permet d'affirmer qu'il existe une solution optimale composée d'au plus $2|V_P|$ allocations, dont les poids $\alpha_q = \frac{a_q}{b_q}$ vérifient $\log(|a_q|) + \log(|b_q|) \leqslant 4|V_P|(\log(2|V_P|) + \log c_{\max})$, où $c_{\max} = \max_{i,j} c_{i,j}$.

Il est un peu plus délicat d'utiliser le Théorème 3.4, qui permet de construire un ordonnancement de débit optimal à partir du motif appauvri, puisqu'il n'y a pas de graphe de tâches qui correspond à l'opération de diffusion. Néanmoins, il est facile de construire un graphe de tâches pour lequel la solution optimale du programme linéaire correspond à une allocation valide. Considérons en effet le graphe de tâches G_A représenté à la Figure 5.2, construit à partir de l'ensemble pondéré d'arbres, solution du programme linéaire $(\alpha_1, \mathcal{T}_1), \ldots, (\alpha_Q, \mathcal{T}_Q)$

La taille de G_A est polynomiale en la taille de la plate-forme (rappelons que $Q \leqslant 2|V_P|$ et $\alpha_q = \frac{a_q}{b_q}$, où $\log(|a_q|) + \log(|b_q|) \leqslant 4|V_P|(\log(2|V_P|) + \log c_{\max})$). En effet, si on note $T_i^{(q)}$ la tâche qui correspond au processeur P_i dans l'arbre $\mathcal{T}_q$, alors

- Le nombre de tâche est exactement $Q|V_P| + 2$
- Le nombre de dépendances est $Q + Q(|V_P| - 1) + \sum_q \text{Nb feuilles}(\mathcal{T}_q) \leqslant Q(2|V_P|)$
- Le poids des dépendances $data_{k,l}$ est 0 ou α_i
- On pose $w_{T_i^{(q)},j} = 0$ si $j = i$ et $+\infty$ sinon, $w_{begin,j} = 0$ si $j = 0$ et $+\infty$ sinon, et $w_{end,j} = 0$ si $j = 0$ et $+\infty$ sinon.

On construit l'allocation comme suit

- $\pi(T_i^{(q)}) = P_i$, $\pi(T_{begin}) = \pi(T_{end}) = P_0$
- $\sigma(T_i^{(q)}, T_j^{(q)}) = (P_i, P_j)$, $\sigma(T_{begin}, T_i^{(q)}) = \emptyset$, $\sigma(T_i^{(q)}, T_{end}) = (P_i, P_0)$

Avec ces définitions, on vérifie que π et σ définissent un $\sum_q \alpha_q$-motif appauvri de longueur 1 valide pour le graphe de tâches G_A. On peut donc appliquer le Théorème 3.4 pour construire un ordonnancement périodique de débit de G_A de $\text{Débit}_{\text{Opt}}$, et donc un schéma périodique de diffusion de débit optimal.

Dans ce paragraphe, nous avons donc montré (en utilisant le Théorème 3.1) qu'il existe une formulation du problème de la maximisation du débit pour une diffusion qui appartient à NP, et nous avons montré (en utilisant le Théorème 3.4) qu'à partir d'une solution sous la forme d'un ensemble (de taille polynomiale) pondéré d'arbres de diffusion $(\alpha_1, \mathcal{T}_1), \ldots, (\alpha_Q, \mathcal{T}_Q)$ respectant les contraintes 1-port, il est possible de construire un ordonnancement périodique de débit $\sum_q \alpha_q$.

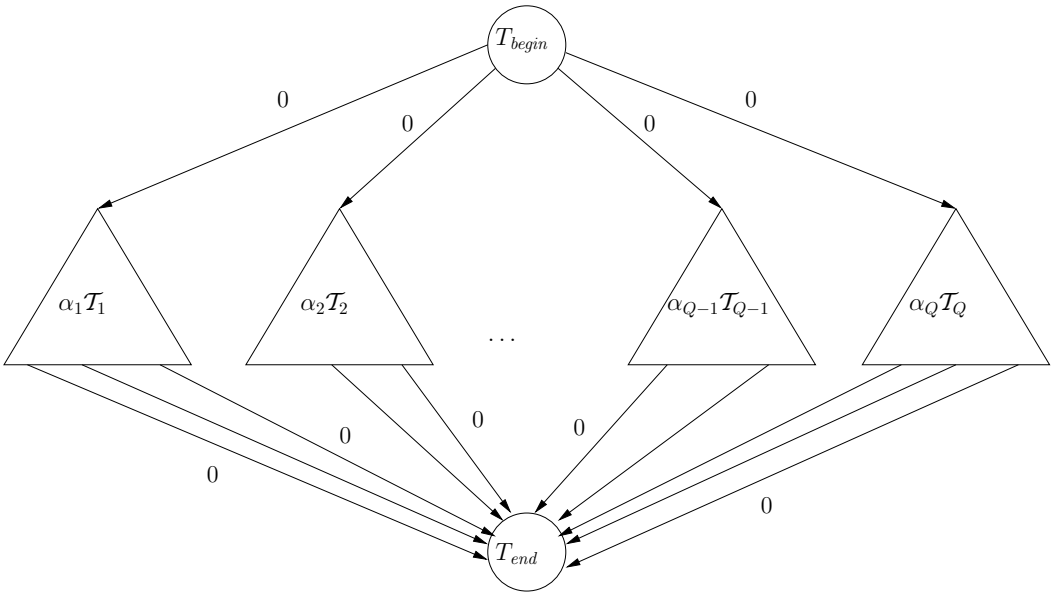


FIG. 5.2 – G_A Graphe de tâches correspond à l'ensemble pondéré d'arbres

Néanmoins, le programme linéaire proposé pour déterminer l'ensemble des arbres n'est pas pratique, même si, en utilisant les techniques développées au Paragraphe 5.3, il peut être résolu en temps polynomial en utilisant la méthode des ellipsoïdes [63]. Nous verrons dans le prochain paragraphe une méthode beaucoup plus rapide pour résoudre le problème, en utilisant un autre programme linéaire, plus proche de ceux utilisés dans le Chapitre 4, paragraphes 4.1 et 4.2.

5.2.2 Diffusion sur plusieurs arbres : Méthodologie

Comme dans le Chapitre 4, nous ne présenterons pas ici le détail des preuves, que le lecteur pourra trouver dans [14] et nous ne présenterons ici que les résultats principaux. Le lecteur pourra également trouver dans [14], s'il n'en est pas déjà convaincu, la preuve de l'utilité de considérer plusieurs arbres de diffusion. Sur une plate-forme simple, il y est montré que le débit en utilisant plusieurs arbres de diffusion est strictement supérieur au meilleur débit obtenu en utilisant le meilleur arbre de diffusion possible. Il y est également prouvé que, de façon analogue à ce que nous avons déjà noté au Paragraphe 4.3, la recherche du meilleur arbre est un problème NP-Complet (pour la maximisation du débit), alors que la recherche de l'ensemble optimal pondéré d'arbres peut être résolu en temps polynomial, comme nous allons le montrer dans ce paragraphe.

Pour déterminer l'ensemble optimal d'arbres de diffusion, nous allons utiliser une démarche proche de celle proposée par Bertsimas et Gamarnik [22], que nous avons décrite au Paragraphe 2.3.2. Nous suivrons donc la démarche suivante

1. La première étape consiste à chercher une borne supérieure sur le débit qui peut être atteint en utilisant la plate-forme décrite par G_P.
2. La deuxième étape consiste à trouver un ensemble pondéré d'arbres de diffusion qui permet d'atteindre le débit annoncé à la première étape.
3. La dernière étape consiste à donner un ordonnancement périodique, dont le débit est optimal (en utilisant la Théorème 3.4, comme au Chapitre 4 et au Paragraphe 5.2.1).

5.2.3 Borne supérieure sur le débit

Comme dans le paragraphe précédent, considérons un schéma quelconque (i.e. pas nécessairement périodique) de diffusion de messages, et notons $T(N)$ le temps nécessaire pour diffuser un message de taille N. Notons également $\beta_q^{(N)}$ le nombre de messages qui sont diffusés en utilisant l'arbre $\mathcal{T}_q$ parmi les N premiers messages (on a donc $\sum_q \beta_q^{(N)} = N$). Dans la suite, nous présentons un ensemble de contraintes linéaires qui doivent être vérifiées par toute opération de diffusion.

Pour la diffusion du message de taille N (pour des raisons de place, nous omettrons dans la suite des indices en N), notons $n_{j,k}$ le nombre moyen (normalisé par rapport à N) de messages circulant sur l'arête $(P_j, P_k) \in E_P$ et $t_{j,k}$ le temps moyen (normalisé par rapport à N d'occupation de l'arête $(P_j, P_k) \in E_P$. Nous noterons également $x_i^{j,k}$, $\forall P_i \in V$, $\forall (P_j, P_k) \in E_P$ le nombre moyen (normalisé par rapport à N) des messages envoyés de P_0 à P_i et qui empruntent l'arête $(P_j, P_k) \in E_P$.

Pour chaque nœud P_j, $\mathcal{N}^{out}(P_j)$ représente l'ensemble des voisins "sortants" de P_j, c'est à dire l'ensemble des nœuds P_k tels que $(P_j, P_k) \in E_P$; de façon similaire, $\mathcal{N}^{in}(P_j)$ représente l'ensemble des voisins "entrants" de P_j, i.e. les nœuds P_k tels que $(P_k, P_j) \in E_P$.

Par construction, on a donc

$$\begin{cases} \forall P_i, \ \ \forall (P_j, P_k) \in E_P, & x_i^{j,k} = \dfrac{1}{N} \displaystyle\sum_{q,\ (P_j,P_k) \in \mathrm{Chem}(P_0,P_i,\mathcal{T}_q)} \beta_q^{(N)} \\ \forall (P_j, P_k) \in E_P, & n_{j,k} = \dfrac{1}{N} \displaystyle\sum_{q,\ (P_j,P_k) \in \mathcal{T}_q} c_{j,k} \beta_q^{(N)} \\ \forall (P_j, P_k) \in E_P, & t_{j,k} = \dfrac{1}{N} \displaystyle\sum_{q,\ (P_j,P_k) \in \mathcal{T}_q} \beta_q^{(N)} \end{cases},$$

où $\mathrm{Chem}(P_0, P_i, \mathcal{T}_q)$ représente le chemin de P_0 à P_i dans l'arbre $\mathcal{T}_q$.

On peut alors vérifier les égalités suivantes

– $\forall P_i$,

$$\sum_{k,\ P_k \in \mathcal{N}^{out}(P_0)} x_i^{0,k} = \frac{1}{N} \sum_{k,\ P_k \in \mathcal{N}^{out}(P_0)\ q,\ (P_0,P_k) \in \mathrm{Chem}(P_0,P_i,\mathcal{T}_q)} \beta_q^{(N)} = 1$$

puisque pour chaque $(P_i, \mathcal{T}_q)$, $\exists$ un et un seul $P_k \in \mathcal{N}^{out}(P_0)$ tel que $(P_0, P_k) \in \mathrm{Chem}(P_0, P_i, \mathcal{T}_q)$.

– $\forall P_i$,

$$\sum_{k,\ P_k \in \mathcal{N}^{out}(P_i)} x_i^{k,i} = \frac{1}{N} \sum_{k,\ P_k \in \mathcal{N}^{in}(P_i)\ q,\ (P_0,P_k) \in \mathrm{Chem}(P_0,P_i,\mathcal{T}_q)} \beta_q^{(N)} = 1$$

puisque pour chaque $(P_i, \mathcal{T}_q)$, $\exists$ un et un seul $P_k \in \mathcal{N}^{in}(P_i)$ tel que $(P_k, P_i) \in \mathrm{Chem}(P_0, P_i, \mathcal{T}_q)$.

– $\forall P_i,\ \forall P_j,\ j \neq 0, i$

$$\begin{aligned} \sum_{k,\ P_k \in \mathcal{N}^{in}(P_j)} x_i^{k,j} &= \frac{1}{N} \sum_{k,\ P_k \in \mathcal{N}^{in}(P_j)\ q,\ (P_k,P_j) \in \mathrm{Chem}(P_0,P_i,\mathcal{T}_q)} \beta_q^{(N)} \\ &= \frac{1}{N} \sum_{k,\ P_k \in \mathcal{N}^{out}(P_j)\ q,\ (P_j,P_k) \in \mathrm{Chem}(P_0,P_i,\mathcal{T}_q)} \beta_q^{(N)} = \sum_{k,\ P_k \in \mathcal{N}^{out}(P_j)} x_i^{j,k} \end{aligned},$$

puisque pour chaque $(P_i, \mathcal{T}_q)$ et chaque P_j tel que $P_j \in \mathrm{Chem}(P_0, P_i, \mathcal{T}_q)$ il existe une et une seule arête entrante en P_j dans $\mathcal{T}_q$ et une et une seule arête sortante de P_j dans $\mathcal{T}_q$.

En outre, on peut vérifier que par construction,

$$\forall P_i, \quad n_{j,k} = \frac{1}{N} \sum_{q,\ (P_j,P_k) \in \mathcal{T}_q} \beta_q^{(N)} \geqslant x_i^{j,k}$$

$$\text{et } t_{j,k} = c_{j,k} n_{j,k}.$$

Définissons enfin $t_j^{(in)}$ le temps moyen (normalisé par rapport à N) passé par P_j à recevoir des messages et $t_j^{(out)}$ le temps moyen (normalisé par rapport à N) à envoyer des messages. On vérifie alors que

$$\forall P_j, \quad t_j^{(in)} = \sum_{P_k \in \mathcal{N}^{in}(P_j)} t_{k,j} \text{ et } t_j^{(out)} = \sum_{P_k \in \mathcal{N}^{out}(P_j)} t_{j,k}.$$

Enfin, les contraintes 1-port s'écrivent

$$\forall P_j, \quad t_j^{(in)} \leqslant \frac{T(N)}{N} \text{ et } t_j^{(out)} \leqslant \frac{T(N)}{N}.$$

Finalement, considérons le programme linéaire suivant :

Diffusion-Temps-Min-Message-Unitaire
$\text{Temps-Min}_{\text{Opt}} = \text{MINIMISER } \tilde{t}_\sigma,$
SUBJECT TO

$$\begin{cases}
\forall i, & \displaystyle\sum_{P_j \in \mathcal{N}^{out}(P_0)} x_i^{s,j} = 1 \\
\forall i \neq 0, & \displaystyle\sum_{P_j \in \mathcal{N}^{in}(P_i)} x_i^{j,i} = 1 \\
\forall j,\ P_j \neq P_s \text{ et } P_i, & \displaystyle\sum_{P_k \in \mathcal{N}^{in}(P_j)} x_i^{k,j} = \sum_{P_k \in \mathcal{N}^{out}(P_j)} x_i^{j,k} \\
\forall P_i \in V_P,\ (P_j, P_k) \in E_P, & n_{j,k} \geqslant x_i^{j,k} \\
\forall (P_j, P_k) \in E_P, & t_{j,k} = n_{j,k} c_{j,k} \\
\forall j, & \displaystyle t_j^{(in)} = \sum_{P_k \in \mathcal{N}^{in}(P_j)} t_{k,j} \\
\forall j, & \displaystyle t_j^{(out)} = \sum_{P_k \in \mathcal{N}^{out}(P_j)} t_{j,k} \\
\forall j,k, & \tilde{t}_\sigma \geqslant t_{j,k} \\
\forall j, & \tilde{t}_\sigma \geqslant t_j^{(in)} \\
\forall j, & \tilde{t}_\sigma \geqslant t_j^{(out)}
\end{cases}$$

Comme au Paragraphe 5.2.1, on peut vérifier que les variables associées à un schéma quelconque de diffusion en temps $T(N)$ d'un message de taille N vérifient les conditions du programme linéaire. On a donc

$$\text{Temps-Min}_{\text{Opt}} \leqslant \frac{T(N)}{N}$$

et $\dfrac{1}{\text{Temps-Min}_{\text{Opt}}}$ représente donc une borne supérieure sur le débit de diffusion en utilisant E_P.

Par rapport au programme linéaire obtenu au Paragraphe 5.2.1, **Diffusion-Temps-Min-Message-Unitaire** possède un nombre polynomial en la taille de la plate-forme de variables et de contraintes, ce qui permet de le résoudre en temps polynomial en la taille de la plate-forme. Par contre, il est plus délicat, comme nous le verrons dans le paragraphe suivant, de trouver un ensemble pondéré d'arbres correspondant à la solution du programme linéaire et de construire un schéma de diffusion de débit $\dfrac{1}{\text{Temps-Min}_{\text{Opt}}}$.

5.2.4 Décomposition de la solution en somme d'arbres pondérés

Pour reconstruire un ensemble pondéré d'arbres à partir de la solution du programme linéaire proposé au paragraphe précédent, nous allons utiliser le théorème suivant, qui représente une version pondérée du théorème d'Edmonds [89, vol.B chapitre 53]. Comme ce théorème s'applique à des graphes à poids entiers,

il est d'abord nécessaire de définir N, le ppcm des dénominateurs des $x_i^{j,k}$ et des $n_{j,k}$ dans la solution du programme linéaire (calculée avec Mupad [45] ou Maple [35] par exemple). En utilisant les techniques présentées au Paragraphe 3.3, on peut facilement montrer que la valeur de N peut être codée de façon polynomiale en la taille de la plate-forme.

Théorème 5.1. *[89, vol.B chapitre 53] Soit $G = (V_P, E_P, Nn_{j,k})$ un graphe pondéré orienté. Il existe Q arbres $\mathcal{T}_1, \ldots, \mathcal{T}_Q$ arbres, de poids entiers $\alpha_1, \ldots, \alpha_Q$, tels que $\forall j,k$, $\sum_l \alpha_l \chi^T_{j,k}(\mathcal{T}_l) \leqslant Nn_{j,k}$, où $\chi^T_{j,k}(\mathcal{T}_l) = 1$ si $(P_j, P_k) \in T_l$ et 0 sinon, et tels que $\sum_l \alpha_l$ est maximale. De plus, l'ensemble des arbres peut être déterminé en temps polynomial,*

$$Q \leqslant |V_P|^3 + |E_P|$$

et

$$\sum_l \alpha_l = \kappa(G, P_0).$$

Dans le théorème, $\kappa(G, P_0) = \min \kappa(G, P_0, P_i)$, où $\kappa(G, P_0, P_i)$ représente le poids minimal total à enlever aux arêtes de G pour déconnecter P_i de P_0. On peut montrer facilement (voir [14]) que $\kappa(G, P_0, P_i) = N$, en notant que les $Nx_i^{j,k}$ définissent un flot entre P_0 et P_i de poids N et en appliquant la dualité entre coupe minimale et flot maximal [37].

Le Théorème 5.1 permet donc de reconstruire, depuis la solution du programme linéaire, un ensemble pondéré d'arbres (avec $Q \leqslant |V_P|^3 + |E_P|$ arbres) $(\alpha_1, \mathcal{T}_1), \ldots, (\alpha_Q, \mathcal{T}_Q)$. En utilisant le graphe de tâches représenté à la Figure 5.2 et défini dans le Paragraphe 5.2.1, on montre qu'il est possible de construire un ordonnancement périodique de débit $\frac{N}{T_p}$, où

$$\begin{aligned}
T_p &= \max_j(\max(\sum_k c_{k,j} \sum_l \alpha_l \chi^T_{k,j}(\mathcal{T}_l), \sum_k c_{j,k} \sum_l \alpha_l \chi^T_{j,k}(\mathcal{T}_l))) \\
&\leqslant \max_j(\max(\sum_k c_{k,j} Nn_{k,j}, \sum_k c_{j,k} Nn_{j,k})) \\
&\leqslant N \max_j(\max(\sum_k c_{k,j} n_{k,j}, \sum_k c_{j,k} n_{j,k})) \\
&\leqslant N \max_j(t_j^{(in)}, t_j^{(out)}) \\
&\leqslant N\text{Temps-Min}_{\text{Opt}}
\end{aligned}$$

On a donc prouvé le théorème suivant :

Théorème 5.2. *Étant donné un graphe de plate-forme $G_P = (V_P, E_P, c_{j,k})$, il est possible de construire en temps polynomial un ordonnancement périodique des communications de débit optimal et le code en chaque nœud (également polynomial en la taille de la plate-forme) réalisant cette diffusion*

Il est intéressant de noter que le Théorème 5.1 fournit un ensemble de $Q \leqslant |V_P|^3 + |E_P|$ arbres pour réaliser la diffusion. Cette solution n'est pas optimale du point de vue de la cardinalité de l'ensemble des arbres utilisés, puisque nous avons vu au Paragraphe 5.2.1 (en utilisant le Théorème 3.1) qu'il existe un ensemble d'au plus $2|V_P|$ arbres qui permet de réaliser la diffusion avec un débit optimal (ceci dit, pour notre propos, un ensemble de cardinal polynomial en la taille de la plate-forme est bien suffisant !).

5.3 Diffusion partielle sous le modèle 1-port bidirectionnel

5.3.1 Formalisation

Dans ce paragraphe, nous considérons le problème de la diffusion d'un message de P_0 à un sous-ensemble des nœuds de la plate-forme, noté $\mathcal{D}$ dans la suite. Le problème de la diffusion à un sous-ensemble des nœuds de la plate-forme (ou multicast) ressemble par de nombreux points au problème de la diffusion totale, même s'il conduit à des résultats fondamentalement différents, puisque nous verrons au Paragraphe 5.3.3

que la détermination du meilleur débit est un problème NP-Complet. Pour les parties semblables, nous ne fournirons donc pas explicitement les preuves (que le lecteur pourra adapter des preuves du Paragraphe 5.2 ou consulter dans [42]).

Comme dans le cas de la diffusion totale, à partir de tout ordonnancement dans lequel une portion du message est reçue plusieurs fois en un même nœud, on peut construire un ordonnancement (de débit supérieur ou égal) dans lequel chaque portion du message n'est reçue qu'une seule fois en chaque nœud, de sorte qu'on peut se contenter de chercher la solution de débit optimale sous la forme d'un ensemble pondéré d'arbres (couvrants l'ensemble des nœuds destinataires $\mathcal{D}$ de la plate-forme) de diffusion $(\alpha_1, \mathcal{T}_1), \ldots, (\alpha_Q, \mathcal{T}_Q)$.

À partir de cette remarque, on peut facilement montrer, comme au Paragraphe 5.2.1 que le débit d'une opération de diffusion partielle est nécessairement inférieur à Débit-Multicast$_{\text{Opt}}$, où

Diffusion-Partielle-Max-Débit
Débit-Multicast$_{\text{Opt}}$ = **Maximiser** $\sum_q \alpha_q$,
sous les contraintes

$$\begin{cases} \forall P_i, & \displaystyle\sum_{j,\ (P_i,P_j)\in E_P} \ \sum_{q, \mathcal{T}_q \ni (P_i,P_j)} \alpha_q c_i j \leqslant 1 & \text{communications sortantes de } P_i \\ \forall P_i, & \displaystyle\sum_{j,\ (P_j,P_i)\in E_P} \ \sum_{q, \mathcal{T}_q \ni (P_j,P_i)} \alpha_q c_j i \leqslant 1 & \text{communications entrantes en } P_i \\ \alpha_q \geqslant 0 & & \text{poids des allocations} \end{cases}$$

Dans les contraintes du programme linéaire, on considère l'ensemble des arbres couvrants l'ensemble $\mathcal{D}$ des destinataires du message à diffuser.

Comme aux Paragraphes 4.1.2 et 5.2, le programme linéaire ci-dessus n'est pas facile à résoudre directement, mais néanmoins, il nous permet d'affirmer qu'il est raisonnable de se poser la question de la maximisation du débit. En effet, le programme linéaire contient $2|V_P|$ contraintes différentes de $\alpha_q \geqslant 0$, et à partir d'une allocation pondérée donnée sous la forme $(\alpha_1, \mathcal{T}_1), \ldots, (\alpha_Q, \mathcal{T}_Q)$, il est facile de vérifier en temps polynomial en Q (et la taille des coefficients des α_q) que les contraintes sur les communications sortantes et entrantes de chaque processeur sont bien vérifiées. On est donc dans les conditions d'application du Théorème 3.1, qui nous permet d'affirmer qu'il existe une solution optimale composée d'au plus $2|V_P|$ allocations, dont les poids $\alpha_q = \frac{a_q}{b_q}$ vérifient $\log(|a_q|) + \log(|b_q|) \leqslant 4|V_P|(\log(2|V_P|) + \log c_{\max})$, où $c_{\max} = \max_{i,j} c_{i,j}$. Comme au Paragraphe 5.2.1, on peut alors utiliser le graphe de tâche décrit à la Figure 5.2, construit à partir de l'ensemble pondéré d'arbres couvrants $\mathcal{D}$, solution du programme linéaire $(\alpha_1, \mathcal{T}_1), \ldots, (\alpha_Q, \mathcal{T}_Q)$, et utiliser le Théorème 3.4 pour reconstruire en temps polynomial un ordonnancement dont le code est de taille polynomiale en la taille de la plate-forme et de débit optimal.

Dans ce paragraphe, nous avons donc montré (en utilisant le Théorème 3.1) qu'il existe une formulation du problème de la maximisation du débit pour une diffusion partielle aux nœuds de $\mathcal{D}$ qui appartient à NP et nous avons montré (en utilisant le Théorème 3.4) qu'à partir d'une solution sous la forme d'un ensemble (de taille polynomiale) pondéré d'arbres de diffusion sur $\mathcal{D}$ $(\alpha_1, \mathcal{T}_1), \ldots, (\alpha_Q, \mathcal{T}_Q)$ respectant les contraintes 1-port, il est possible de construire un ordonnancement périodique de débit $\sum_q \alpha_q$.

Dans le Paragraphe 5.2.1, nous avions noté qu'il était en fait possible de résoudre directement en temps polynomial le problème **Diffusion-Max-Débit**, en utilisant les techniques qui seront présentées au Paragraphe 5.4 et la méthode des ellipsoïdes [63]. Il est intéressant de noter que la situation est toute différente dans le cas de la diffusion partielle **Diffusion-Partielle-Max-Débit**. En effet, la résolution de **Diffusion-Max-Débit** revient à résoudre, dans le dual, un problème de séparation équivalent à la recherche d'un arbre de poids maximal (qui peut donc être résolu en temps polynomial). La même technique appliquée à la résolution de **Diffusion-Partielle-Max-Débit** conduit à la recherche d'un arbre de Steiner de poids maximal (qui est un problème NP-Complet). Nous verrons d'ailleurs dans le Paragraphe 5.3.3 que le problème de la détermination du débit optimal d'une opération de diffusion partielle est en fait un problème NP-Complet.

5.3.2 Bornes supérieure et inférieure sur le débit

Comme dans le Paragraphe 5.2, nous ne présenterons pas ici le détail des preuves, que le lecteur pourra trouver dans [42] et nous ne présenterons ici que les résultats principaux.

5.3.2.1 Borne supérieure sur le débit

Pour déterminer une borne supérieure sur le débit, nous utilisons une technique semblable à celle présentée au Paragraphe 5.2.3. En utilisant les mêmes notations $x_i^{j,k}$,$t_{j,k}$ et $n_{j,k}$, mais en ne définissant $x_i^{j,k}$ que pour les nœuds $P_i \in \mathcal{D}$.

Borne-Supérieure-Diffusion-Partielle
$\text{Temps-Min-Partielle}_{\text{Opt}} = \text{MINIMISER } \tilde{t}_\sigma$,
sous les contraintes

$$\begin{cases} \forall i \in \mathcal{D}, & \sum_{P_j \in \mathcal{N}^{out}(P_0)} x_i^{s,j} = 1 \\ \forall i \in \mathcal{D} \neq 0, & \sum_{P_j \in \mathcal{N}^{in}(P_i)} x_i^{j,i} = 1 \\ \forall j,\ P_j \neq P_s \text{ et } P_i \in \mathcal{D}, & \sum_{P_k \in \mathcal{N}^{in}(P_j)} x_i^{k,j} = \sum_{P_k \in \mathcal{N}^{out}(P_j)} x_i^{j,k} \\ \forall P_i \in \mathcal{D},\ (P_j, P_k) \in E, & n_{j,k} \geqslant x_i^{j,k} \\ \forall (P_j, P_k) \in E, & t_{j,k} = n_{j,k} c_{j,k} \\ \forall j, & t_j^{(in)} = \sum_{P_k \in \mathcal{N}^{in}(P_j)} t_{k,j} \\ \forall j, & t_j^{(out)} = \sum_{P_k \in \mathcal{N}^{out}(P_j)} t_{j,k} \\ \forall j, k, & \tilde{t}_\sigma \geqslant t_{j,k} \\ \forall j, & \tilde{t}_\sigma \geqslant t_j^{(in)} \\ \forall j, & \tilde{t}_\sigma \geqslant t_j^{(out)} \end{cases}$$

Comme au Paragraphe 5.2.1, on peut vérifier que les variables associées à un schéma quelconque de diffusion en temps $T(N)$ d'un message de taille N vérifient les conditions du programme linéaire. On a donc

$$\text{Temps-Min-Partielle}_{\text{Opt}} \leqslant \frac{T(N)}{N}$$

et $\dfrac{1}{\text{Temps-Min-Partielle}_{\text{Opt}}}$ représente donc une borne supérieure sur le débit de diffusion en utilisant E_P.

Le programme linéaire obtenu, comme au Paragraphe 5.2.3 possède un nombre polynomial en la taille de la plate-forme de variables et de contraintes, ce qui permet de le résoudre en temps polynomial en la taille de la plate-forme. Par contre, contrairement au Paragraphe 5.2.3, la borne supérieure obtenue est ici stricte, et il n'est pas possible de reconstruire un ensemble d'arbres de diffusion permettant d'atteindre le débit

$$\frac{1}{\text{Temps-Min-Partielle}_{\text{Opt}}},$$

comme nous le verrons au Paragraphe 5.3.2.3

5.3.2.2 Borne inférieure sur le débit

Pour obtenir une borne inférieure sur le débit, on peut considérer le problème de la diffusion personnalisée de P_0 à l'ensemble des nœuds de $\mathcal{D}$. En utilisant les techniques proposées au Paragraphe 5.1, et en ne faisant apparaître dans le graphe de tâches de la Figure 5.1 que les nœuds de $\mathcal{D}$, on peut vérifier que ce problème peut être résolu en temps polynomial. En utilisant la technique présentée au Paragraphe 5.2, on peut également montrer que la solution du programme linéaire suivant permet de calculer le débit optimal d'une diffusion personnalisée sur $\mathcal{D}$, et fournit donc une borne inférieure

$$\frac{1}{\text{Temps-Min-Partielle-Personnalisée}_{\text{Opt}}}$$

sur le débit d'une opération de diffusion partielle sur $\mathcal{D}$.

Borne-Supérieure-Diffusion-Partielle
Temps-Min-Partielle-Personnalisée$_{\text{Opt}}$ = MINIMISER $\tilde{t}_\sigma$,
sous les contraintes

$$\begin{cases} \forall i \in \mathcal{D}, & \sum_{P_j \in \mathcal{N}^{out}(P_0)} x_i^{s,j} = 1 \\ \forall i \in \mathcal{D} \neq 0, & \sum_{P_j \in \mathcal{N}^{in}(P_i)} x_i^{j,i} = 1 \\ \forall j,\ P_j \neq P_s \text{ et } P_i \in \mathcal{D}, & \sum_{P_k \in \mathcal{N}^{in}(P_j)} x_i^{k,j} = \sum_{P_k \in \mathcal{N}^{out}(P_j)} x_i^{j,k} \\ \forall P_i \in \mathcal{D},\ (P_j, P_k) \in E, & n_{j,k} = \sum_i x_i^{j,k} \\ \forall (P_j, P_k) \in E, & t_{j,k} = n_{j,k} c_{j,k} \\ \forall j, & t_j^{(in)} = \sum_{P_k \in \mathcal{N}^{in}(P_j)} t_{k,j} \\ \forall j, & t_j^{(out)} = \sum_{P_k \in \mathcal{N}^{out}(P_j)} t_{j,k} \\ \forall j,k, & \tilde{t}_\sigma \geqslant t_{j,k} \\ \forall j, & \tilde{t}_\sigma \geqslant t_j^{(in)} \\ \forall j, & \tilde{t}_\sigma \geqslant t_j^{(out)} \end{cases}$$

5.3.2.3 Comparaison entre les deux bornes

Il est intéressant de noter que les deux bornes établies au Paragraphe 5.3.2.1 et 5.3.2.2 sont strictes, comme démontré à la Figure 5.3.

Néanmoins, les programmes linéaires des paragraphes 5.3.2.1 et 5.3.2.2 ne diffèrent que par la contrainte liant les $x_i^{j,k}$ et les $n_{j,k}$. Quand on s'intéresse à la borne inférieure, on pose $n_{j,k} = \sum_{i \in \mathcal{D}} x_i^{j,k}$, ce qui correspond à supposer les messages $x_i^{j,k}$ tous différents (hypothèse trop pessimiste comme montré à la Figure 5.3). Au contraire, dans le cas de la borne supérieure, on pose $n_{j,k} = \max_{i \in \mathcal{D}} x_i^{j,k}$, ce qui revient à supposer que tous les messages $x_i^{j,k}$ peuvent être choisis comme sous-messages du plus gros d'entre eux (hypothèse trop optimiste comme montré à la Figure 5.3). En comparant ces deux définitions de $n_{j,k}$, il est facile de montrer que l'écart entre les deux bornes inférieure et supérieure est borné par un facteur $|\mathcal{D}|$ [42].

5.3.3 Complexité de la diffusion partielle

Nous définissons le problème de la maximisation du débit sous la forme suivante.

Définition 5.1 (Diffusion-Partielle-Compacte($G_P, P_0, \mathcal{D}, \rho$)). *Étant donnés une plate-forme $G_P = (V_P, E_P, c_{j,k})$, un processeur source P_0, un ensemble $\mathcal{D}$ de destinataires du message et une borne sur le débit ρ, existe-t-il un ensemble d'arbres de diffusion (couvrant $\mathcal{D}$) $(\alpha_1, \mathcal{T}_1), \ldots, (\alpha_Q, \mathcal{T}_Q)$ tels que*

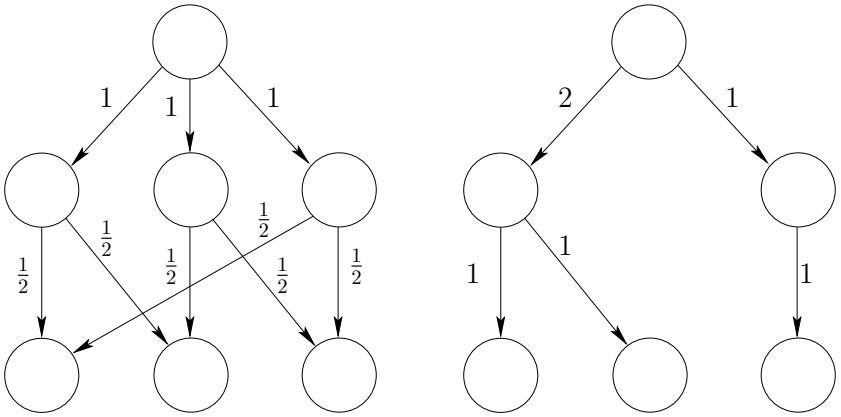
- $Q \leqslant 2|V_P|$,
- *les $\alpha_q = \frac{a_q}{b_q}$ vérifient $\log(|a_q|) + \log(|b_q|) \leqslant 4|V_P|(\log(2|V_P|) + \log c_{\max})$, où $c_{\max} = \max_{i,j} c_{i,j}$ et*
- $$\frac{\sum_l \alpha_l}{\max_j(\max(\sum_k c_{k,j} \sum_l \alpha_l \chi^T_{k,j}(\mathcal{T}_l), \sum_k c_{j,k} \sum_l \alpha_l \chi^T_{j,k}(\mathcal{T}_l)))} \geqslant \rho.$$

Nous avons montré au Paragraphe 5.3.1 (en utilisant le Théorème 3.1) que le fait de restreindre la recherche à des arbres de diffusion vérifiant $Q \leqslant 2|V_P|$ et $\log(|a_q|) + \log(|b_q|) \leqslant 4|V_P|(\log(2|V_P|) + \log c_{\max})$, où $c_{\max} = \max_{i,j} c_{i,j}$ ne change pas la valeur optimale du débit. Nous avons également montré (en utilisant le Théorème 3.4) qu'à partir d'un ensemble d'arbres tel que

$$\frac{\sum_l \alpha_l}{\max_j(\max(\sum_k c_{k,j} \sum_l \alpha_l \chi^T_{k,j}(\mathcal{T}_l), \sum_k c_{j,k} \sum_l \alpha_l \chi^T_{j,k}(\mathcal{T}_l)))} \geqslant \rho,$$

il est possible de construire un ordonnancement périodique des communications de débit $\geqslant \rho$.

On vérifie donc le théorème suivant :



(a) graphe de plate-forme E_P

(b) Borne inférieure sur le débit (paragraphe 5.3.2.2) débit=$\frac{1}{3}$

(c) Borne supérieure sur le débit (Paragraphe 5.3.2.1) débit=$\frac{2}{3}$

(d) Meilleure solution (recherche exhaustive), débit=$\frac{1}{2}$

FIG. 5.3 – Aucune des bornes n'est fine en général

Théorème 5.3. *Étant donnés une plate-forme $G_P = (V_P, E_P, c_{j,k})$, un processeur source P_0 et un ensemble de destinataires du message $\mathcal{D}$ et une borne sur le débit ρ, s'il existe un ordonnancement (quelconque) des communications réalisant la diffusion partielle d'un message de taille N en temps $T(N)$ avec*

$$\limsup \frac{N}{T(N)} \geqslant \rho,$$

alors il existe une solution de Diffusion-Partielle-Compacte($G_P, P_0, \mathcal{D}, \rho$).

Finalement, le Théorème 5.4 établit la complexité du problème de diffusion partielle.

Théorème 5.4. *Diffusion-Partielle-Compacte($G_P, P_0, \mathcal{D}, \rho$) est NP-Complet et n'appartient pas à la classe APX.*

La preuve du Théorème 5.4 peut être trouvée dans [42] et s'appuie sur une réduction à partir de **Minimum-Set-Cover** [48].

5.4 Diffusion sous le modèle 1-port unidirectionnel

Dans ce paragraphe, nous allons analyser le problème de la diffusion sous le modèle 1-port unidirectionnel. Rappelons que dans le modèle 1-port unidirectionnel, un processeur est susceptible de recevoir un message de l'un de ses voisins ou d'envoyer un message à l'un de ses voisins à un instant donné. Un des avantages de ce modèle est qu'on peut facilement transformer un nœud fonctionnant sous un modèle exotique (k-ports, Δ-port) en un ensemble de nœuds fonctionnant sous le modèle 1-port, comme nous le verrons au Paragraphe 5.4.4.

La démarche que nous allons présenter dans ce paragraphe s'applique assez directement aux problèmes que nous avons considérés jusqu'ici (distribution de tâches indépendantes, diffusion personnalisée...) aussi bien sous le modèle 1-port unidirectionnel que sous le modèle 1-port bidirectionnel. Elle a par contre l'inconvénient de reposer sur la méthode des ellipsoïdes [63], ce qui rend difficile l'affirmation de son utilité pratique.

5.4.1 Difficultés liés au modèle 1-port unidirectionnel

Dans tous les exemples que nous avons considérés jusqu'alors, nous nous sommes appuyés sur le Théorème 3.4. Ce théorème affirme qu'il est possible de découpler la recherche du motif appauvri de débit optimal (c'est à dire d'une description globale de l'activité des processeurs en régime permanent) et la construction d'un ordonnancement permettant d'atteindre en pratique ce débit.

Ce découplage est fortement lié au modèle 1-port bidirectionnel. En effet, on peut définir, sous le modèle 1-port unidirectionnel, un motif appauvri en assurant que les capacités de calcul et de communication de chaque nœud ne sont pas excédées. En reprenant le formalisme du Paragraphe 3.3, un K motif appauvri de longueur T_p sous le modèle 1-port unidirectionnel peut être défini, par analogie avec le modèle bidirectionnel (voir définition 3.11) par

Définition 5.2 (K-motif appauvri de longueur T_p unidirectionnel). *Un K-motif appauvri de longueur T_p unidirectionnel est une allocation (π, σ) de $G_A \otimes [\![1, K]\!]$ sur G_P vérifiant globalement les contraintes de ressource et les contraintes 1-port unidirectionnel, i.e.*

- *P_i a le temps de traiter toutes les tâches qui lui sont allouées*

$$\forall i, \sum_{(k,r), r\in[1,K],\ \pi(T_k,r)=P_i} w_{i,k} \leqslant T_p.$$

- *P_i a le temps de recevoir et d'envoyer toutes les communications dans lesquelles il est engagé*

$$\forall i, \sum_{(k,l,r,j), r\in[1,K],\ \sigma(e_{k,l},r)\ni P_j\to P_i} data_{k,l} \times c_{j,i} + \sum_{(k,l,r,j), r\in[1,K],\ \sigma(e_{k,l},r)\ni P_i\to P_j} data_{k,l} \times c_{i,j} \leqslant T_p.$$

Malheureusement, il n'existe pas d'équivalent du Théorème 3.4 dans le modèle unidirectionnel. Le Théorème 3.4 repose en effet sur la possibilité de décomposer le graphe de communication (qui est biparti dans le cas du modèle bidirectionnel) en une somme pondérée de couplages, dont le poids total est exactement le temps d'occupation maximal (en entrée ou en sortie) d'un des processeurs. Dans le cas du modèle unidirectionnel, le graphe de communication n'est plus biparti (le graphe représentant les communications est en fait le graphe de plate-forme, puisque chaque processeur possède exactement une interface pour les communications) et il est bien connu que le nombre chromatique (même fractionnaire) d'un graphe quelconque est en général supérieur au degré pondéré maximal d'un sommet [89].

Une des conséquences de ce résultat est qu'il est nécessaire, dans le cadre du modèle unidirectionnel, de considérer simultanément la description globale de l'activité des processeurs en régime permanent et la construction d'un ordonnancement permettant d'atteindre en pratique ce débit.

Pour illustrer cette difficulté supplémentaire, considérons le problème de la diffusion d'un message depuis P_1 sur la plateforme décrite à la Figure 5.4. Si on se concentre uniquement sur l'occupation du port de

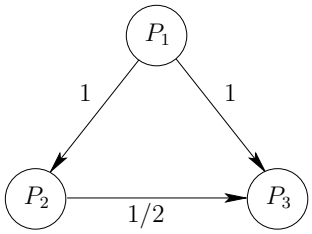


FIG. 5.4 – Illustration de la difficulté du modèle 1-port unidirectionnel (G_P)

communication de chacun des processeurs de la plate-forme, comme nous l'avons fait au Paragraphe 5.2, on est amené à résoudre le programme linéaire suivant,

DÉBIT MAXIMAL DE LA DIFFUSION SOUS LE MODÈLE 1-PORT SUR G_P (MAUVAISE SOLUTION)
MAXIMISER a,
SOUS LES CONTRAINTES

$$\begin{cases} x_2^{1,2} = a, \quad x_3^{1,2} + x_3^{1,3} = a, \quad x_2^{1,2} = a, \quad x_3^{1,3} + x_3^{2,3} = a, \quad x_3^{1,2} = x_3^{2,3} \\ t_{1,2} \leqslant x_2^{1,2}, \ t_{1,2} \leqslant x_3^{1,2}, \ t_{1,3} \leqslant x_3^{1,3}, \ t_{2,3} \leqslant \frac{1}{2} x_3^{2,3} \\ t_1 = t_{1,2} + t_{1,3}, \quad t_2 = t_{1,2} + t_{2,3}, \quad t_3 = t_{2,3} + t_{1,3} \\ t_1 \leqslant 1, \quad t_2 \leqslant 1, \quad t_3 \leqslant 1 \end{cases}$$

qui fournit la solution décrite à la Figure 5.5, où est indiqué sur chaque arête le temps d'utilisation de cette arête pendant une unité de temps. On peut vérifier que les contraintes 1-port unidirectionnel sont bien satisfaites en chacun des nœuds (le temps d'occupation de P_1 et P_2 est égal à 1, et celui de P_3 à $\frac{1}{2}$. La solution affirme qu'il est possible de diffuser un message de taille $\frac{3}{4}$ à chaque unité de temps. Mais il est

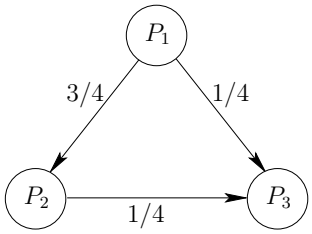
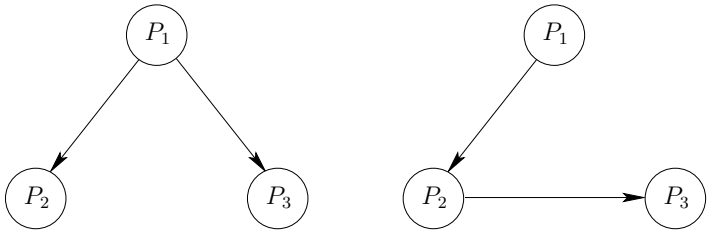


FIG. 5.5 – temps d'occupation de chaque lien sur l'exemple

impossible d'atteindre effectivement ce débit puisque par exemple, pendant une communication entre P_1 et P_3 (qui dure $\frac{1}{4}$ unité de temps), P_2 ne peut ni envoyer un message à P_3 ni en recevoir un de P_1 de sorte qu'il est nécessairement inactif (ce qui est incompatible avec le fait qu'il est occupé en permanence dans la solution du programme linéaire).

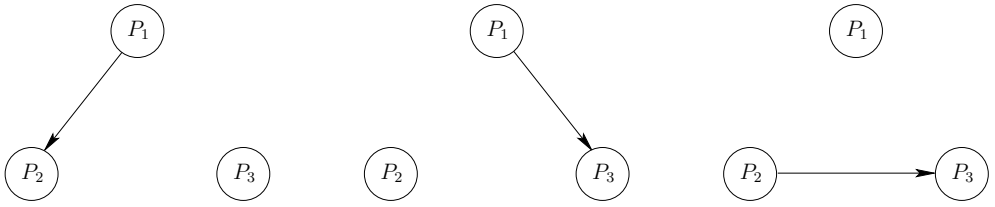
Puisqu'il n'est pas possible de découpler la description de l'activité des processeurs en régime permanent et la construction d'un ordonnancement permettant d'atteindre en pratique ce débit, cherchons donc simultanément l'ensemble des arbres couvrant G_P pouvant être utilisés pour diffuser le message (représentés à la Figure 5.6) et l'ensemble des couplages dans G_P pouvant être utilisés pour organiser les communications indépendantes (représentés à la Figure 5.7).

Avec ces définitions, il est facile de construire un programme linéaire dont la solution fournit directement un K-motif (pas appauvri) de longueur 1 sur lequel on peut appliquer les techniques développées au Paragraphe 3.3 et le Théorème 3.2 pour trouver un ordonnancement optimal réalisant le débit K. Dans ce programme linéaire y_i représente le poids affecté à l'arbre de diffusion $\mathcal{T}_i$ (i.e. la fraction des messages transmis selon $\mathcal{T}_i$) et x_i le poids affecté au couplage $\mathcal{M}_i$ (i.e. la fraction du temps pendant laquelle les messages



(a) Arbre $\mathcal{T}_1$ (b) Arbre $\mathcal{T}_2$

FIG. 5.6 – Arbres de diffusion possibles sur G_P (Figure 5.4)



(a) Couplage $\mathcal{M}_1$ (b) Couplage $\mathcal{M}_2$ (c) Couplage $\mathcal{M}_3$

FIG. 5.7 – Couplages de communications indépendantes possibles sur G_P (Figure 5.4)

seront échangés selon $\mathcal{M}_i$).

MOTIF POUR LA DIFFUSION SUR G_P
MAXIMISER $y_1 + y_2$,
SOUS LES CONTRAINTES

$$\begin{cases} x_1, \quad x_2, \quad x_3, \quad y_1, \quad y_2 \geqslant 0 \\ x_1 + x_2 + x_3 \leqslant 1 \\ x_1 = (y_1 + y_2)c_{1,2} = y_1 + y_2 \\ x_2 = y_1 \\ x_3 = y_2 c_{1,3} = \frac{1}{2} y_2 \end{cases}$$

La première contrainte ($x_1 + x_2 + x_3 \leqslant 1$) exprime que la durée totale doit être inférieure à 1 et le reste des contraintes stipule que les communications associées aux arbres ont effectivement le temps de se dérouler.

La résolution du programme linéaire nous permet de vérifier qu'il est possible de transmettre un message de taille $\frac{2}{3}$ toutes les unités de temps avec $y_1 = 0, \quad y_2 = \frac{2}{3}, \quad x_1 = \frac{2}{3}, \quad x_2 = 0, \quad$ et $x_3 = \frac{1}{3}$.

Le programme ci dessus fournit donc la solution au problème de la maximisation du débit pour une opération de diffusion sur G_P sous le modèle 1-port unidirectionnel. Dans le paragraphe suivant, nous allons donner la forme générale de ce programme linéaire pour un graphe de plate-forme G_P quelconque, et montrer qu'il est possible de le résoudre en temps polynomial (ce qui n'est pas *a priori* trivial, puisque le nombre d'arbres couvrants et le nombre de couplages peuvent être de taille exponentielle en la taille de G_P) en utilisant la méthode des ellipsoïdes [54].

5.4.2 Diffusion de débit optimal sous le modèle 1-port unidirectionnel

5.4.2.1 Programme linéaire fournissant le débit optimal

Comme nous l'avons vu au paragraphe précédent, si on note $\mathcal{M}$ l'ensemble des couplages de G_P et $\mathcal{T}$ l'ensemble des arbres couvrant la plate-forme, l'ensemble pondéré des arbres à utiliser pour diffuser le

message et l'ensemble pondéré des couplages pour construire un motif de diffusion est donné par la solution du programme linéaire suivant :

$$(\mathcal{P})\begin{cases} \textbf{Débit Maximal Diffusion} \\ \textbf{Débit-Optimal} = \text{MAXIMISER} \sum_{\mathcal{T}_t \in \mathcal{T}} y_t, \\ \text{SOUS LES CONTRAINTES} \\ & \forall \mathcal{M}_m \in \mathcal{M}, \forall \mathcal{T}_t \in \mathcal{T}, x_m \geqslant 0, \quad y_t \geqslant 0 \\ \text{(Temps total de communication)} & \sum_{\mathcal{M}_m \in \mathcal{M}} x_m \leqslant 1 \\ \text{(Réalisation des échanges sur chaque arête)} & \forall e_k = (i,j) \in E_P \sum_{\mathcal{M}_m \ni (i,j)} x_m = \sum_{\mathcal{T}_t \ni (i,j)} c_{i,j} \cdot y_t \end{cases}$$

De la solution de ce programme linéaire, on peut appliquer les techniques vues au Chapitre 3, Paragraphe 3.3, Théorème 3.2 pour construire un ordonnancement périodique qui atteint le débit optimal (en fait le Théorème 3.2 nous est utile pour vérifier les contrainte de dépendance, la vérification des contraintes de ressource ne pose aucun problème, comme dans la démonstration du Théorème 3.2). On peut également noter qu'en utilisant le Théorème 3.1, on peut affirmer qu'il existe une solution du programme linéaire $\mathcal{P}$ ci-dessus telle que

- Le nombre d'arbres et de couplages utilisés est borné par $E_P + 1$
- Le poids de chaque couplage $x_m = \frac{a_m}{b_m}$ et le poids de chaque arbre $y_t = \frac{c_t}{d_t}$ vérifient

$$\log(|a_m|) + \log(|b_m|) \leqslant 2(|E_P|+1)(\log(|E_P|+1) + \log c_{\max}) \text{ et}$$

$$\log(|c_t|) + \log(|d_t|) \leqslant 2(|E_P|+1)(\log(|E_P|+1) + \log c_{\max}), \text{ où } c_{\max} = \max_{i,j} c_{i,j}.$$

Réciproquement, considérons un schéma de diffusion de N messages en temps $T(N)$. À chaque étape de temps, on peut déterminer quel couplage est utilisé pour réaliser les communications, ce qui nous conduit à un ensemble pondéré de couplages $(\beta_m, \mathcal{M}_m)$ avec $\sum \beta_m = T(N)$. On peut également déterminer, pour chaque message élémentaire transmis, l'arbre selon lequel il est diffusé, ce qui nous conduit à un ensemble pondéré d'arbres de diffusion $(\alpha_t, \mathcal{T}_t)$ avec $\sum \alpha_t = N$. On peut alors facilement vérifier que $x_m = \frac{\beta_m}{T(N)}$ et $y_t = \frac{\alpha_t}{T(N)}$ sont solution du programme linéaire $\mathcal{P}$, et donc que tout algorithme d'ordonnancement réalisant la diffusion a un débit inférieur à **Débit-Optimal**.

5.4.2.2 Méthode des ellipsoïdes [63]

Dans ce paragraphe, nous allons citer les résultats principaux qui nous sont utiles sur la méthode des ellipsoïdes pour la programmation linéaire, qui sont présentés en détail dans [54]. Le lecteur intéressé pourra trouver plus de détail sur la méthode de résolution dans [20].

Pour appliquer la méthode des ellipsoïdes, on va considérer le programme linéaire $\mathcal{D}$ dual de $\mathcal{P}$, et qui est défini (voir [20]) par

$$(\mathcal{D})\begin{cases} \textbf{Formulation duale de } \mathcal{P} \\ \text{MINIMISER } x_1, \\ \text{SOUS LES CONTRAINTES} \\ & x_1 \geqslant 0 \\ \forall m = 1, \ldots, |\mathcal{M}| & \sum_{e_k \in M_m} x_{k+1} \leqslant x_1 \\ \forall t = 1, \ldots, |\mathcal{T}| & \sum_{e_k = (i,j) \in T_t} c_{i,j} \cdot x_{k+1} \geqslant 1 \end{cases}$$

Pour un problème d'optimisation sur un convexe K, on peut définir les deux problèmes suivants

Définition 5.3 (Problème d'optimisation forte (SOPT(K, C))). *Soient un convexe K et un vecteur $C \in \mathbb{Q}^n$, trouver un vecteur $x \in K$ qui maximise $C^T \cdot x$ ou montrer que K est vide.*

Définition 5.4 (Problème de séparation forte (SSEP(K,x))). *Étant donnés un convexe K et un vecteur $x \in \mathbb{Q}^n$, décider si $x \in K$ et sinon, trouver un hyperplan qui sépare x de K ; plus précisément, trouver un vecteur $C \in \mathbb{Q}^n$ tel que $C^T \cdot x > \max\left\{C^T \cdot y \mid y \in K\right\}$.*

L'équivalence entre les problèmes d'optimisation et de séparation peut être trouvée dans [54, chapitre 6] et est exprimée dans le théorème suivant.

Théorème 5.5 (Théorème 6.4.9 dans [54]). *Chacun des deux problèmes*
- *séparation forte,*
- *optimisation forte,*

peut être résolu en temps oracle-polynomial pour tout polyèdre "bien décrit" si on connaît un oracle pour l'autre problème.

La preuve que $\mathcal{D}$ est un polyèdre "bien décrit" peut être trouvée dans [20].

enfin, le théorème suivant affirme qu'il est possible de reconstruire la solution de $\mathcal{P}$ à partir de l'exécution de la méthode des ellipsoïdes sur $\mathcal{D}$.

Théorème 5.6 (Théorème 6.5.15 dans [54]). *Il existe un algorithme de temps oracle-polynomial qui, pour chaque $c \in \mathbb{Q}^n$ et pour tout polyèdre bien décrit $(P; n, \phi)$, où la taille de ϕ est polynomiale en la taille des entrées, et donné par un oracle de séparation forte dont chaque sortie est de taille au plus ϕ*
(i) trouve une solution du problème primal, ou
(ii) prouve que le problème admet des solutions non bornées ou pas de solution du tout.

5.4.2.3 Résolution "pratique" de $\mathcal{P}$

Considérons le problème de séparation forte Diffusion-SSEP dans le dual $\mathcal{D}$:

Définition 5.5 (Diffusion-SSEP(G_P,P_0,x)). *Étant donnés un graphe de plate-forme G_P, un nœud source P_0 et un vecteur $x \in \mathbb{Q}^{|E_P|+1}$, existe-t-il un vecteur $C \in \mathbb{Q}^n$ tel que $C^T \cdot x > \max\left\{C^T \cdot y,\ y \in \mathcal{D}\right\}$.*

Lemme 5.1. *Diffusion-SSEP(G_P,P_0,x) peut être résolu en temps polynomial.*

Démonstration. Considérons une instance du problème d'optimisation forte sur $\mathcal{D}$, donnée par un vecteur x. Nous cherchons à montrer que $x \in \mathcal{D}$ ou exhiber une contrainte qui n'est pas satisfaite en x. $\mathcal{D}$ est donné par 3 types de contraintes

$$\begin{cases} \text{Contrainte } I & x_1 \geqslant 0 \\ \text{Contraintes } II & \forall m = 1, \ldots, |\mathcal{M}| \quad \sum_{e_k \in M_m} x_{k+1} \leqslant x_1 \\ \text{Contraintes } III & \forall t = 1, \ldots, |\mathcal{T}| \quad \sum_{e_k=(i,j) \in T_t} c_{i,j} \cdot x_{k+1} \geqslant 1 \end{cases}.$$

- Étant donné x la contrainte I peut évidemment être vérifiée en temps polynomial.
- Contraintes de type II :
$$\forall m = 1, \ldots, |\mathcal{M}| \quad \sum_{e_k \in M_m} x_{k+1} \leqslant x_1$$
Considérons le graphe pondéré $G_{\mathcal{M}} = (V_P, E_P, c_{\mathcal{M}}$, où $c_{\mathcal{M}}(e_k) = x_{k+1}$. On peut calculer dans $G_{\mathcal{M}}$ un couplage $M^{\mathcal{M}}_{\max}$ de poids maximal $w^{\mathcal{M}}_{\max}$ en temps polynomial [37].
Si $w^{\mathcal{M}}_{\max} \leqslant x_1$, alors toutes les inégalités de type II sont vérifiées. Sinon, l'inégalité correspondant à $M^{\mathcal{M}}_{\max}$ n'est pas satisfaite, i.e. $\sum_{e_k \in M^{\mathcal{M}}_{\max}} x_{k+1} > x_1$.
- Contraintes de type III :
$$\forall t = 1, \ldots, |\mathcal{T}| \quad \sum_{e_k=(i,j) \in \mathcal{T}_t} c_{i,j} \cdot x_{k+1} \geqslant 1$$

où $\mathcal{T}$ est l'ensemble des arbres couvrants de G_P. Considérons le graphe pondéré $G_{\mathcal{T}} = (V_P, E_P, c_{\mathcal{T}}$, où $c_{\mathcal{T}}(e_k = (P_i, P_j)) = c_{i,j} \cdot x_{k+1}$. On peut calculer dans $G_{\mathcal{T}}$ un arbre $T^{\mathcal{T}}_{\min}$ de poids minimal $w^{\mathcal{T}}_{\min}$ en temps polynomial [37].
Si $w^{\mathcal{T}}_{\min} \geqslant 1$, alors toutes les inégalités de type III sont vérifiées. Sinon, l'inégalité correspondant à $T^{\mathcal{T}}_{\min}$ n'est pas satisfaite, i.e. $\sum_{e_k=(P_i,P_j)\in T^{\mathcal{T}}_{\min}} c_{i,j}x_{k+1} < 1$.

On peut donc, en temps polynomial vérifier que $x \in \mathcal{D}$ ou exhiber une contrainte de $\mathcal{D}$ qui n'est pas satisfaite en x. ■

En utilisant le Lemme 5.1, le Théorème 5.6 et le Théorème 3.2, on peut alors montrer le théorème suivant :

Théorème 5.7. *Étant donnés une plate-forme G_P et un nœud source P_0, il est possible de calculer en temps polynomial en la taille de la plate-forme G_P :*
- *le débit optimal d'une opération de diffusion dans G_P depuis P_0 (Lemme 5.1)*
- *de trouver l'ensemble des arbres sur lesquels sont réalisés les diffusions et l'ensemble des couplages sur lesquels sont réalisées les échanges de messages (Théorème 5.6) et*
- *un ordonnancement périodique de débit optimal (Théorème 3.2)*

Toutefois, nous ne pouvons pas considérer ce résultat comme pratique, à cause du temps d'exécution de l'algorithme de résolution du programme linéaire avec la méthode des ellipsoïdes.

5.4.3 Extensions

Il est vraiment regrettable qu'il n'existe pas d'algorithme de résolution efficace de programme linéaire avec la méthode des ellipsoïdes. En effet, tous les problèmes considérés dans les chapitres 4 et 5 peuvent être résolus en utilisant la méthodologie développée dans le Paragraphe 5.4. Pour reprendre les exemples de ce chapitre, si on considère la diffusion sous le modèle 1-port bidirectionnel, le problème de la détermination d'un couplage de poids maximal dans G_P devient un problème de couplage de poids maximal dans le graphe biparti $(V_P \times V_P, E_P)$, qui peut également être résolu en temps polynomial. Les problèmes de diffusion personnalisée, sous le modèle 1-port unidirectionnel ou bidirectionnel peuvent également être résolus de cette manière, de même que la diffusion généralisée (dans laquelle chaque processeur P_i envoie le même message m_i à tous les autres processeurs [20]).

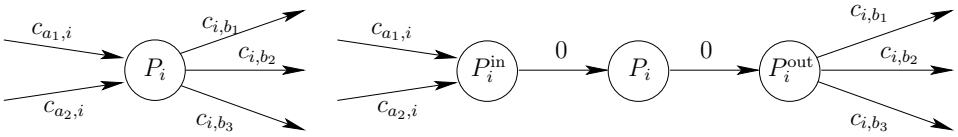
Pour le problème de la diffusion partielle (Paragraphe 5.3), le problème de séparation dans le dual se ramène à un calcul d'arbre de Steiner de poids maximal (au lieu d'un arbre couvrant de poids maximal). Ce résultat illustre la difficulté de la diffusion partielle, démontrée par le Théorème 5.4.

On peut également résoudre les problèmes abordés dans le Chapitre 4 en utilisant cette méthode. Comme dans le cas de la diffusion personnalisée, le problème de la maximisation du débit, pour un graphe de tâche quelconque nécessite la résolution (pour le problème de séparation dans le dual), de MINIMUM-MULTIWAY-CUT, qui est NP-complet (et même APX-complet) [4]. Ce résultat illustre également la difficulté de ce problème, démontrée par le Théorème 4.2.

5.4.4 Transformation des nœuds

Une des qualités du modèle 1-port unidirectionnel est qu'il est facile de transformer un nœud fonctionnant sous un modèle 1-port bidirectionnel ou Δ-port (i.e. capable de communiquer en envoi et en réception avec tous ses voisins) en un groupe de nœuds opérants sous le modèle 1-port unidirectionnel, comme illustré aux Figures 5.8 et 5.9.

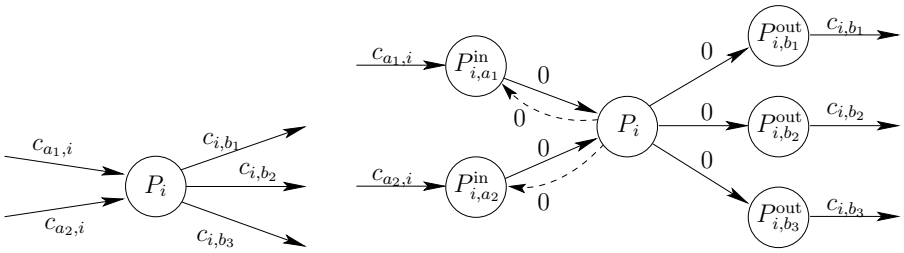
La preuve de la validité de ces transformations peut être trouvée dans [20]. Il est donc possible de résoudre en temps polynomial avec la méthode des ellipsoïdes et en utilisant ces transformations, le problème de la diffusion sur une plate-forme quelconque contenant des nœuds opérants sous différents modes (les transformations de G_P présentées aux Figures 5.8 et 5.9 produisent un graphe de taille polynomiale en la taille de G_P).



(a) Nœud initial opérant en 1-port bidirectionnel

(b) Nœud simulé opérant en 1-port unidirectionnel

FIG. 5.8 – Simulation d'un nœud opérant sous le modèle 1-port bidirectionnel.



(a) Nœud initial opérant en Δ-port

(b) Nœud simulé opérant en 1-port unidirectionnel

FIG. 5.9 – Simulation d'un nœud opérant sous le modèle Δ-port.

5.5 Conclusion, perspectives et problèmes ouverts

5.5.1 Complexité

5.5.1.1 Ce qui a été fait

Comme dans le cas des tâches indépendantes (Chapitre 4), les résultats obtenus en terme de complexité sont très satisfaisants, et le passage de la minimisation du temps de complétion à la maximisation du débit simplifie effectivement les problèmes d'ordonnancement de communications collectives (le lecteur pourra se reporter à [42] et [14] pour trouver des résultats de complexité, du point de vue de la minimisation du temps de complétion, pour différentes modélisations des coûts de communication).

Nous avons montré que pour le modèle 1-port bidirectionnel, il existe des algorithmes polynomiaux efficaces pour résoudre la diffusion personnalisée (scatter, Paragraphe 5.1), le regroupement (gather, Paragraphe 5.1), la diffusion (broadcast, Paragraphe 5.2). Ces résultats s'étendent sans difficulté à une diffusion personnalisée depuis un nombre quelconque de sources ou à une diffusion depuis un nombre quelconque de sources. On peut également l'étendre à des opérations de réduction [67] (opération dans laquelle on doit calculer $\bigoplus_1^n X_i$, où chaque P_i possède exactement un X_i et $\bigoplus$ est une loi associative quelconque, mais l'optimalité prouvée dans [67] n'est valable que pour une loi non commutative).

Dans le cas du modèle 1-port unidirectionnel, il est également possible de construire en temps polynomial des algorithmes d'ordonnancement de débit optimaux pour tous les problèmes précédents, avec des techniques similaires à celles que nous avons présentées au Paragraphe 5.4. Toutefois, tous ces algorithmes s'appuient sur la méthode des ellipsoïdes [63] et on ne peut donc parler d'algorithmes polynomiaux efficaces dans ce contexte.

Par contre, nous avons montré au Paragraphe 5.2 que le problème de la diffusion partielle est NP-Complet et mal approximable (nous avons montré qu'il n'appartient pas à la classe APX). Le problème du calcul préfixe

(dans lequel chaque P_i possède exactement un X_i et doit calculer $\bigoplus_1^i X_j$, où $\bigoplus$ est une loi associative et non commutative) est également NP-Complet (le lecteur pourra trouver la preuve dans [42]).

5.5.1.2 Ce qui reste à faire

Nous avons vu que le problème de trouver un algorithme efficace pour les problèmes de communications collectives sous le modèle 1-port est encore ouvert (cette remarque s'applique également à la distribution de tâches indépendantes simples sur une plate-forme quelconque dans le modèle 1-port unidirectionnel). Le modèle 1-port bidirectionnel semble (heureusement) plus réaliste pour les infrastructures réseau actuelles. Toutefois, comme nous l'avons vu au Paragraphe 5.4.4, il est facile de réduire toute plate-forme, avec des nœuds opérants sous différents modes, en une plate-forme dans laquelle tous les nœuds opèrent sous le modèle 1-port unidirectionnel. La recherche d'algorithmes efficaces est donc un problème pratique important.

La complexité exacte des deux problèmes NP-Complets que nous avons rencontrés dans ce document (la maximisation du débit pour la distribution de tâches complexes et la diffusion partielle) est également ouvert (même si nous avons montré qu'aucun de ces problèmes n'appartient à APX). Dans le cas de la diffusion partielle, nous avons proposé une heuristique (s'appuyant sur la diffusion personnalisée et la diffusion sur l'ensemble de la plate-forme) au Paragraphe 5.3.2, qui est garanti à un facteur $|\mathcal{D}|$ où $\mathcal{D}$ est l'ensemble des destinataires du message. Les résultats de complexité de ces deux problèmes sont peut être moins désespérants qu'il n'y paraît, et mériteraient d'être précisés sur des modèles de plate-forme plus réalistes. En effet, les deux preuves s'appuient sur des plates-formes très asymétriques, dans lesquelles tous les liens ne peuvent être utilisés que dans une seule direction. Il serait intéressant d'analyser la complexité dans le cas d'une plate-forme symétrique (dans laquelle $\forall (i,j) \in E_P, c_{i,j} = c_{j,i}$) et dans le cas d'une plate-forme "à asymétrie bornée" (dans laquelle $\max_{i,j} \frac{c_{i,j}}{c_{j,i}} \leqslant \rho$). La complexité de ces deux problèmes est ouverte.

5.5.2 Mise en œuvre pratique

5.5.2.1 Difficultés liées à la mise en œuvre pratique

Nous retrouvons ici les difficultés déjà évoquées au Paragraphe 4.3.2 et qui sont liées à la décomposition du graphe des communications en somme pondérée de couplages dans l'obtention d'un ordonnancement périodique de débit optimal (Théorème 3.4). Les solutions possibles évoquées au Paragraphe 4.3.2, pour s'adapter à des variations des performances des liens s'appliquent également dans le contexte des communications collectives (s'appuyer sur la solution optimale et imposer des synchronisations fortes à la fin de chaque couplage, ou construire dynamiquement et localement une suite de couplages maximaux par négociation entre les processeurs).

Nous retrouvons également les problèmes liés aux contraintes mémoire et à la taille du motif construit par la méthode présentée au Paragraphe 3.3. Les problèmes de maximisation du débit pour les communications collectives en présence de limitation mémoire n'ont pas été analysés, contrairement à ce qui avait été fait pour la distribution de tâches indépendantes, mais on peut conjecturer sans trop de risque que les résultats de NP-Complétude obtenus dans le cas de la distribution de tâches indépendantes s'étendent aux communications collectives.

Du point de vue de la mise en œuvre pratique sur une plate-forme dont les performances varient au cours du temps, les problèmes de diffusion (personnalisée ou collective) posent une difficulté supplémentaire, et nécessitent sans doute une reformulation du problème. Considérons par exemple le problème de la diffusion d'un message sur une plate-forme en arbre. D'un point de vue théorique, le problème est trivial puisque chaque nœud P_i peut calculer la taille maximale du message qu'il peut transmettre en temps unitaire à ses descendants ($x_i = \frac{1}{\sum_j c_{i,j}}$) et le débit optimal est alors donné par $\min_i x_i$. Si la plate-forme est stable, chaque nœud peut alors alors distribuer cycliquement le message à ses descendants pour atteindre le régime permanent optimal. Toutefois, dans ce contexte, si les performances d'un lien sont très mauvaises ($\exists (i,j)$, tel que $c_{i,j}$ est très grand) le débit de la diffusion devient très petit. Rappelons que nous avons justifié l'intérêt des opération de diffusion au Chapitre 2 par la diffusion (avant calcul et transmission des données propres à chaque tâche) des données (le modèle) nécessaires à toutes les tâches de calcul. Dans ce contexte, il n'est

pas acceptable de ralentir toute l'opération de diffusion à cause d'un seul nœud. Il est donc indispensable de se poser la question de la diffusion à un sous ensemble des nœuds de la plate-forme, pour maximiser le débit de l'opération globale (distribution des données et distributions des tâches), ce qui nécessite de traiter simultanément les problèmes évoqués aux Chapitres 4 et 5.

5.5.2.2 Ce qui a été fait

Ce qui a été fait peut paraître bien léger par rapport aux problèmes évoqués dans les paragraphes précédents. Néanmoins, les résultats présentés justifient l'intérêt d'une recherche statique de la solution optimale, du point de vue de la maximisation du débit, par rapport à des solutions gloutonnes, naturellement plus dynamiques.

Pour le problème de la diffusion, nous avons comparé les résultats obtenus (dans le cadre de la diffusion d'un message selon un seul arbre de diffusion) par des heuristiques construisant l'arbre à partir de la décomposition optimale en somme d'arbres pondérés, et par des heuristiques gloutonnes de la littérature. Les résultats obtenus (sur une grande variété de plates-formes "réalistes" générées par Tiers [39]) montrent la supériorité des heuristiques s'appuyant sur la solution optimale dont le calcul a été présenté au Paragraphe 5.2. Ils sont présentés dans [21].

Pour le problème de la diffusion partielle, nous avons présenté rapidement au Paragraphe 5.3 deux heuristiques. L'une consiste à calculer la solution obtenue pour la diffusion sur l'ensemble de la plate-forme. Elle peut être raffinée itérativement en ne conservant dans la plate-forme que les nœuds par lesquels circulent une quantité importante du message destiné aux nœuds cibles de la diffusion partielle. La seconde heuristique présentée consiste au contraire à partir de la diffusion personnalisée du message. Elle peut être raffinée itérativement en ajoutant dans l'ensemble cible des nœuds de la plate-forme par lesquels circulent une partie importante des messages destinés aux nœuds cibles. Ces deux heuristiques sont présentées en détail dans [42]. Des comparaisons avec des heuristiques de diffusion partielle (généralement gloutonnes) de la littérature sont également présentées. Elles montrent que dans le cas des plates-formes "réalistes" générées par Tiers [39], les heuristiques de diffusion partielle que nous proposons ont un comportement très satisfaisant par rapport à la borne inférieure constituée par le temps de diffusion. La qualité de ces résultats peut s'expliquer par la différence entre les plates-formes utilisées pour montrer la NP-Complétude des problèmes de diffusion partielle (plates-formes dans lesquels les liens sont fortement asymétriques et le degré entrant des nœuds est grand) et les plates-formes "réalistes" générées par Tiers.

5.5.2.3 Un exemple possible d'application : la diffusion vidéo

L'exemple de la diffusion vidéo semble très adapté pour pouvoir appliquer nos travaux à un problème pratique important. Dans le cadre de la diffusion de vidéos, le problème de la maximisation du débit est en effet essentiel. Plus précisément, il est nécessaire d'assurer un certain débit pour que la diffusion soit possible (grossièrement, il est nécessaire de transmettre environ 24 images par seconde pour assurer une bonne qualité de diffusion). Au contraire, on peut accepter un certain décalage (en temps) dans la réception de la vidéo par les différents utilisateurs. Toutefois, les problèmes de mémoire évoqués dans les paragraphes précédents sont ici importants puisque dans les schémas de diffusion que nous proposons, les différentes parties du message peuvent suivre des routes différentes et leur réception peut être décalée par rapport à leur diffusion, ce qui nécessite une mémorisation des messages arrivés "trop tôt".

Le problème de la diffusion vidéo nécessite également de faire intervenir des calculs, puisqu'on a la possibilité de compresser le message ou de dégrader sa qualité pour obtenir le débit souhaité. Une compression du message avant émission nécessite une décompression à la réception, ce qui nécessite de prendre en compte cette tâche (le problème de la compression en émission et de la décompression en réception a été analysé pour gzip dans un contexte différent dans [61]).

Conclusion

Dans le Chapitre 1, nous avons tenté d'extraire les caractéristiques de plates-formes de type grille de calcul qui ont un impact sur la conception d'algorithmes d'ordonnancement :

- les ressources de communication et de calcul sont hétérogènes
- l'utilisation des liens longue distance et d'intergiciels complexes nécessite la prise en compte de latences de communication et de calcul.
- un processeur ne peut pas être engagé simultanément dans un nombre quelconque de communications.
- les plates-formes visées sont instables dans le temps et la simulation est la seule voie pour comparer les algorithmes d'ordonnancement (mais pour cela, il nous faudrait également disposer d'un grand nombre de plates-formes avec des traces d'occupation des ressources, ce qui n'est pas encore le cas).

Dans le Chapitre 2, nous avons tenté de justifier que les caractéristiques évoquées ci-dessus rendent désespérée la recherche d'algorithmes d'ordonnancement garantis pour la minimisation du temps de complétion. En outre, nous avons vu que la nature des applications déployées sur les grilles de calcul fait de la maximisation du débit une métrique raisonnable pour la conception d'algorithmes d'ordonnancement.

Du point de vue de la complexité, le changement de métrique a montré son efficacité dans les Chapitres 4 et 5. De tous les problèmes considérés, seuls deux ne peuvent être résolus en temps polynomial et ces résultats mériteraient encore d'être affinés, en se restreignant à des plates-formes "réalistes". Tous les résultats présentés aux Chapitres 4 et 5 prennent en compte les caractéristiques statiques des plates-formes évoquées ci-dessus (hétérogénéité, latences, limitation sur le nombre de communications simultanées). Par contre, la prise en compte de la dynamicité des plates-formes a (pour l'instant) largement résisté à nos efforts.

Pour comprendre la difficulté introduite par la dynamicité, il est nécessaire de revenir à des problèmes simples. Le seul exemple présenté dans ce document (même s'il manque de fondements théoriques) d'algorithme décentralisé et robuste aux variations de performance est celui de la distribution de tâches indépendantes sur un arbre. Il peut constituer un point de départ pour apprendre à analyser le comportement d'algorithmes décentralisés et leur robustesse. Nous pourrons ensuite compliquer le modèle des tâches, le modèle de la plate-forme et prendre en compte la présence de plusieurs applications. Nous pourrons ainsi juger si d'autres simplifications du modèle sont nécessaires (communications interruptibles).

À l'autre extrémité, il est également nécessaire de réfléchir à l'adaptation des algorithmes à ces plates-formes. Pour appliquer les techniques évoquées dans ce document, il est nécessaire que les algorithmes s'expriment sous la forme d'un grand nombre de tâches indépendantes, ce qui pousse à repenser la formulation des problèmes classiques (vers des méthodes de Monte Carlo ou des algorithmes asynchrones par exemple). Même si les formulations ainsi obtenues sont moins efficaces, elles peuvent se prêter à de meilleures distributions sur des plates-formes de grille de calcul et, si nous avons vu que l'utilisation de ces plates-formes est périlleuse et délicate, tirons parti du fait que ce ne sont pas les ressources de calcul qui y manquent !

Annexe A

Bibliographie

[1] M. Adler, Y. Gong, and A. L. Rosenberg. Optimal sharing of bags of tasks in heterogeneous clusters. In *15th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA'03)*, pages 1–10. ACM Press, 2003.

[2] D. Altilar and Y. Paker. An optimal scheduling algorithm for parallel video processing. In *IEEE Int. Conference on Multimedia Computing and Systems*. IEEE Computer Society Press, 1998.

[3] D. Altilar and Y. Paker. Optimal scheduling algorithms for communication constrained parallel processing. In *Euro-Par 2002*, LNCS 2400, pages 197–206. Springer Verlag, 2002.

[4] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation*. Springer, Berlin, Germany, 1999.

[5] C. Banino, O. Beaumont, A. Legrand, and Y. Robert. Scheduling strategies for master-slave tasking on heterogeneous processor grids. In *PARA'02: International Conference on Applied Parallel Computing*, LNCS 2367, pages 423–432. Springer Verlag, 2002.

[6] Cyril Banino, Olivier Beaumont, Larry Carter, Jeanne Ferrante, Arnaud Legrand, and Yves Robert. Scheduling strategies for master-slave tasking on heterogeneous processor platforms. *IEEE Trans. Parallel Distributed Systems*, 15(4):319–330, 2004.

[7] Amotz Bar-Noy, Sudipto Guha, Joseph Naor, and Baruch Schieber. Message multicasting in heterogeneous networks. *SIAM J. Comput*, 30(2):347–358, 2000.

[8] Olivier Beaumont, Vincent Boudet, Arnaud Legrand, Fabrice Rastello, and Yves Robert. Heterogeneity considered harmful to algorithm designers. In *Cluster'2000*, pages 403–404. IEEE Computer Society Press, 2000. extended version available as RR2000-24 at http://www.ens-lyon.fr/LIP/Pub/rr2000.html.

[9] Olivier Beaumont, Vincent Boudet, Fabrice Rastello, and Yves Robert. Partitioning a square into rectangles: Np-completeness and approximation algorithms. *Algorithmica*, 34(3):217–239, 2002.

[10] Olivier Beaumont, Larry Carter, Jeanne Ferrante, Arnaud Legrand, and Yves Robert. Bandwidth-centric allocation of independent tasks on heterogeneous platforms. In *International Parallel and Distributed Processing Symposium IPDPS'2002*. IEEE Computer Society Press, 2002.

[11] Olivier Beaumont, Henri Casanova, Arnaud Legrand, Yves Robert, and Yang Yang. Scheduling divisible loads on star and tree networks: results and open problems. *IEEE Trans. Parallel Distributed Systems*, january 2005.

[12] Olivier Beaumont, Arnaud Legrand, Loris Marchal, and Yves Robert. Assessing the impact and limits of steady-state scheduling for mixed task and data parallelism on heterogeneous platforms. In *HeteroPar'2004: International Conference on Heterogeneous Computing, jointly published with ISPDC'2004: International Symposium on Parallel and Distributed Computing*. IEEE Computer Society Press, 2004. extended version available as RR2004-20 at http://www.ens-lyon.fr/LIP/Pub/rr2004.html.

[13] Olivier Beaumont, Arnaud Legrand, Loris Marchal, and Yves Robert. Complexity results and heuristics for pipelined multicast operations on heterogeneous platforms. In *2004 International Conference on Parallel Processing (ICPP'2004)*, pages 267–274. IEEE Computer Society Press, 2004. extended version available as RR2004-07 at http://www.ens-lyon.fr/LIP/Pub/rr2004.html.

[14] Olivier Beaumont, Arnaud Legrand, Loris Marchal, and Yves Robert. Pipelining broadcasts on heterogeneous platforms. *IEEE Trans. Parallel Distributed Systems*, 2005, to appear. extended version available as RR2003-34 at http://www.ens-lyon.fr/LIP/Pub/rr2003.html.

[15] Olivier Beaumont, Arnaud Legrand, Loris Marchal, and Yves Robert. Independent and divisible task scheduling on heterogeneous star-shaped platforms with limited memory. In *13th Euromicro Conference on Parallel, Distributed and Network-based Processing*. IEEE Computer Society Press, February 2005, accepted for publication. extended version available as RR2004-22 at http://www.ens-lyon.fr/LIP/Pub/rr2004.html.

[16] Olivier Beaumont, Arnaud Legrand, Fabrice Rastello, and Yves Robert. Dense linear algebra kernels on heterogeneous platforms: redistribution issues. *Parallel Computing*, 28(2):155–185, 2002.

[17] Olivier Beaumont, Arnaud Legrand, and Yves Robert. Static scheduling strategies for heterogeneous systems. In *ISCIS XVII, Seventeenth International Symposium On Computer and Information Sciences*, pages 18–22. CRC Press, 2002. Available as RR2002-07 at www.ens-lyon.fr/LIP/Pub/rr2002.html.

[18] Olivier Beaumont, Arnaud Legrand, and Yves Robert. The master-slave paradigm with heterogeneous processors. *IEEE Trans. Parallel Distributed Systems*, 14(9):897–908, 2003.

[19] Olivier Beaumont, Arnaud Legrand, and Yves Robert. Scheduling divisible workloads on heterogeneous platforms. *Parallel Computing*, 29:1121–1152, 2003.

[20] Olivier Beaumont and Loris Marchal. Pipelining broadcasts on heterogeneous platforms under the one-port model. Technical Report 2004-32, LIP, ENS Lyon, France, jul 2004. available at http://www.ens-lyon.fr/LIP/Pub/rr2004.html.

[21] Olivier Beaumont, Loris Marchal, and Yves Robert. Broadcast trees for heterogeneous platforms. Technical Report 2004-xx, LIP, ENS Lyon, France, oct 2004. available very soon at http://www.ens-lyon.fr/LIP/Pub/rr2004.html.

[22] D. Bertsimas and D. Gamarnik. Asymptotically optimal algorithm for job shop scheduling and packet routing. *Journal of Algorithms*, 33(2):296–318, 1999.

[23] V. Bharadwaj, D. Ghose, V. Mani, and T.G. Robertazzi. *Scheduling Divisible Loads in Parallel and Distributed Systems*. IEEE Computer Society Press, 1996.

[24] P. B. Bhat, V. K. Prasanna, and C. S. Raghavendra. Efficient collective communication in distributed heterogeneous systems. In *19th IEEE International Conference on Distributed Computing Systems (ICDCS'99)*. IEEE Computer Society Press, 1999.

[25] Prashanth B. Bhat, C. S. Raghavendra, and Viktor K. Prasanna. Efficient collective communication in distributed heterogeneous systems. *J. Parallel Distrib. Comput.*, 63(3):251–263, 2003.

[26] J. Blazewicz, M. Drozdowski, and M. Markiewicz. Divisible task scheduling - concept and verification. *Parallel Computing*, 25:87–98, 1999.

[27] Jacek Blazewicz, M. Machowiak, Gregory Mounie, and Denis Trystram. Approximation algorithms for scheduling independent malleable tasks. In *Proceedings of EuroPar'01*, 2001.

[28] Pierre Boulet, Jack Dongarra, Fabrice Rastello, Yves Robert, and Frédéric Vivien. Algorithmic issues on heterogeneous computing platforms. *Parallel Processing Letters*, 9(2):197–213, 1999.

[29] Peter Brucker and Sigrid Knust. Complexity results for scheduling problems. Website http://www.mathematik.uni-osnabrueck.de/research/OR/class/.

[30] Kenneth L. Calvert, Matthew B. Doar, and Ellen W. Zegura. Modeling internet topology. *IEEE Communications Magazine*, 35(6):160–163, June 1997. Available at http://citeseer.nj.nec.com/calvert97modeling.html.

[31] Henri Casanova. Simgrid: A toolkit for the simulation of application scheduling. In *Proceedings of the IEEE Symposium on Cluster Computing and the Grid (CCGrid'01)*. IEEE Computer Society, May 2001. Available at http://grail.sdsc.edu/papers/simgrid_ccgrid01.ps.gz. Further information on Simgrid is available at http://gcl.ucsd.edu/simgrid.

[32] Henri Casanova. Modeling large-scale platforms for the analysis and the simulation of scheduling strategies. In *Proceedings of the 6th Workshop on Advances in Parallel and Distributed Computational Models*. IEEE, 2004.

[33] Henri Casanova, Arnaud Legrand, and Loris Marchal. Scheduling distributed applications: the SimGrid simulation framework. In *Proceedings of the third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03)*, May 2003.

[34] Henri Casanova and Loris Marchal. A network model for simulation of grid application. Technical Report 2002-40, LIP, oct 2002.

[35] B. W. Char, K. O. Geddes, G. H. Gonnet, M. B. Monagan, and S. M. Watt. *Maple Reference Manual*, 1988.

[36] Y. Chen, M. Winslett, S. Kuo, Y. Cho, M. Subramaniam, and K. E. Seamons. Performance modeling for the Panda array I/O library. In *Proceedings of Supercomputing '96*. ACM Press and IEEE Computer Society Press, 1996. Available at http://cdr.cs.uiuc.edu/pubs/super96.ps.

[37] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[38] Phillip M. Dickens, Philip Heidelberger, and David M. Nicol. A distributed memory LAPSE: Parallel simulation of message-passing programs. In *Proceedings of the 8th Workshop on Parallel and Distributed Simulation (PADS '94)*, 1994. Available at http://www.csam.iit.edu/~pmd/pubs/Distributed.ps.

[39] M. Doar. A better model for generating test networks. In *Proceedings of Globecom '96*, November 1996. Available at http://citeseer.nj.nec.com/doar96better.html.

[40] M. Drozdowski. *Selected problems of scheduling tasks in multiprocessor computing systems*. PhD thesis, Instytut Informatyki Politechnika Poznanska, Poznan, 1997.

[41] P-F. Dutot, G. Mounie, , and D. Trystram. Scheduling parallel tasks: Approximation algorithms. In Joseph Leung, editor, *Handbook on Scheduling algorithms: Algorithms, Models and Performance Analysis*. CRC press, 2004.

[42] Pierre-François Dutot. Complexity of master-slave tasking on heterogeneous trees. *European Journal of Operational Research*, 2003. Special issue on the Dagstuhl meeting on Scheduling for Computing and Manufacturing systems (to appear).

[43] Pierre-François Dutot. Master-slave tasking on heterogeneous processors. In *International Parallel and Distributed Processing Symposium IPDPS'2003*. IEEE Computer Society Press, 2003.

[44] R. Esser and R. Knetcht. Intel Paragon XP/S - Architecture and Software Environment. Technical Report KFA-ZAM-IB-9305, Zentralinstitut fur Angewandte Mathematik - Forschungszentrum Julich, April 1993.

[45] The MuPAD Group (Benno Fuchssteiner et al.). *MuPAD User's Manual - MuPAD Version 1.2.2*. John Wiley and sons, Chichester, New York, march 1996.

[46] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, pages 251–262, 1999. Available at http://citeseer.nj.nec.com/faloutsos99powerlaw.html.

[47] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl J. Supercomputer Applications*, 11(2):115–128, 1997. Available at ftp.globus.org/pub/globus/papers/globus.ps.gz.

[48] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1991.

[49] Genome@HOME project. Website http://genomeathome.stanford.edu/.

[50] R. Giroudeau and J.C. Konig. General non-approximability results in presence of hierarchical communications. In IEEE Computer Society Press, editor, *HeteroPar'04, Cork, Ireland*, 2004.

[51] Rodolphe Giroudeau. *L'impact des délais de communications hiérarchiques sur la complexité et l'approximation des problèmes d'ordonnancement*. PhD thesis, Université d'Évry Val d'Essonne, 2000.

[52] G. H. Golub and C. F. Van Loan. *Matrix computations*. Johns Hopkins, 1989.

[53] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. High-performance, portable implementation of the MPI Message Passing Interface Standard. *Parallel Computing*, 22(6):789–828, 1996. Available at http://www-unix.mcs.anl.gov/mpi/mpich/.

[54] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithm and Combinatorial Optimization*. Algorithms and Combinatorics 2. Springer-Verlag, 1994. Second corrected edition.

[55] A. Gupta, G. Parmentier, , and D. Trystram. Scheduling precedence task graphs with disturbances. *RAIRO Operations Research*, 37, 2003.

[56] C. Hanen and A. Munier. Cyclic scheduling on parallel processors: an overview. In P. Chrétienne, E. G. Coffman, J. K. Lenstra, and Z. Liu, editors, *Scheduling Theory and its Applications*, pages 193–226. John Wiley & Sons, 1995.

[57] Dorit S. Hochbaum. *Approximation algorithms for NP-hard problems*. PWS Publishing Co., 1997.

[58] Bo Hong and Viktor K. Prasanna. Distributed adaptive task allocation in heterogeneous computing environments to maximize throughput. In *Proceedings of IPDPS'04*. IEEE Computer Society, 2004.

[59] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matching in bipartite graphs. *SIAM J. Computing*, 2(4):225–231, 1973.

[60] Van Jacobson, apr 1997. Presented at the Mathematical Science Research Institute. Available at ftp://ftp.ee.lbl.gov/pathchar/.

[61] Emmanuel Jeannot, Björn Knutsson, and Mats Björkman. Adaptive online data compression. In *HPDC'02*, pages 379–388. IEEE Computer Society Press, 2002.

[62] Emmanuel Jeannot and Frédéric Wagner. Two fast and efficient message scheduling algorithms for data redistribution through a backbone. In *Proceedings of IPDPS'04*. IEEE Computer Society, 2004.

[63] L. G. Khachiyan. A polynomial algorithm in linear programming (in russian). *Soviet Mathematikcs Doklady*, 20:191–194, 1979.

[64] Barbara Kreaseck. *Dynamic autonomous scheduling on Heterogeneous Systems*. PhD thesis, University of California, San Diego, 2003.

[65] A. Lastovetsky and R. Reddy. Data partitioning with a realistic performance model of networks of heterogeneous computers. In *Proceedings of HCW'04*. IEEE Computer Society, 2004.

[66] C. Lee and M. Hamdi. Parallel image processing applications on a network of workstations. *Parallel Computing*, 21:137–160, 1995.

[67] A. Legrand, L. Marchal, and Y. Robert. Optimizing the steady-state throughput of scatter and reduce operations on heterogeneous platforms. In *APDCM'2004, 6th Workshop on Advances in Parallel and Distributed Computational Models*. IEEE Computer Society Press, 2004.

[68] A. Legrand, F. Mazoit, and M. Quinson. An application-level network mapper. Research Report RR-2003-09, LIP, ENS Lyon, France, feb 2003.

[69] "A. Legrand and M. Quinson". "automatic deployment of the Network Weather Service. In *High Performance Grid Computing workshop*. "IEEE Computer Society Press", "2004".

[70] Arnaud Legrand. *Algorithmique parallèle hétérogène et techniques d'ordonnancement : approches statiques et dynamiques*. PhD thesis, École Normale Supérieure de Lyon, dec 2003.

[71] Renaud Lepère, Denis Trystram, and Gerhard J. Woeginger. Approximation algorithms for scheduling malleable tasks under precedence constraints. *Int. J. Found. Comput. Sci.*, 13(4), 2002.

[72] J. Lerouge and A. Legrand. Towards realistic scheduling simulation of distributed applications. Technical Report 2002-28, LIP, ENS Lyon, France, July 2002.

[73] Alberto Medina, Ibrahim Matta, and John Byers. On the origin of power laws in internet topologies. *ACM Computer Communication Review*, 30(2), April 2000. Available at http://citeseer.nj.nec.com/medina00origin.html.

[74] Alix Munier. *Contribution à l'étude des ordonnancements cycliques*. PhD thesis, Université Paris 6, February 1991.

[75] The network simulator - ns-2. http://www.isi.edu/nsnam/ns.

[76] Peter Pacheco. *Parallel programming with MPI*. Morgan Kaufmann, 1997.

[77] François Pellegrini. Scotch and libscotch 3.4 user's guide. Technical report, 2001.

[78] Ljubomir Perkovic. *Edge Coloring, Polyhedra and Probability*. PhD thesis, Carnegie Mellon University, November 1998.

[79] G. N. Srinivasa Prasanna, A. Agrawal, and B. R. Musicus. Hierarchical compilation of macro dataflow graphs for multiprocessors with local memory. *IEEE Trans. Parallel Distrib. Syst.*, 5(7):720–736, 1994.

[80] G. N. Srinivasa Prasanna and Bruce R. Musicus. Generalized multiprocessor scheduling for directed acyclic graphs. In *Proceedings of the 1994 conference on Supercomputing*, pages 237–246. IEEE Computer Society Press, 1994.

[81] Prime. URL: http://www.mersenne.org.

[82] Jean-Pierre Raffarin. Discours lors de la synthèse des assises des libertés locales à rouen. Website http://www.snesup.fr/docs/exterieurs/ext03-raffarin_decentralisation_rouen-extraits.htm.

[83] Frederic Raoult. Ordonnancement et partitionnement de graphes, 2002.

[84] F. Rastello. *Partitionnement: optimisations de compilation et algorithmique hétérogène*. PhD thesis, Ecole Normale Supérieure de Lyon, September 2000.

[85] A. L. Rosenberg. Sharing partitionable workloads in heterogeneous NOWs: greedier is not better. In D. S. Katz, T. Sterling, M. Baker, L. Bergman, M. Paprzycki, and R. Buyya, editors, *Cluster Computing 2001*, pages 124–131. IEEE Computer Society Press, 2001.

[86] T. Saif and M. Parashar. Understanding the behavior and performance of non-blocking communications in mpi. In *9th International Euro-Par Conference (Euro-Par 2004)*, pages 173–182. Lecture Notes in Computer Science 3149, Springer-Verlag, 2004.

[87] Edward R. Scheinerman and Daniel H. Ullman. *Fractional Graph Theory: A Rational Approach to the Theory of Graphs*. Wiley-Interscience Series in Discrete Mathematics and Optimization, 1997.

[88] Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, New York, 1986.

[89] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer-Verlag, 2003.

[90] SETI@home project. Website http://setiathome.ssl.berkeley.edu/.

[91] G. Shao. *Adaptive scheduling of master/worker applications on distributed computational resources*. PhD thesis, Dept. of Computer Science, University Of California at San Diego, 2001.

[92] Gary Shao, Francine Berman, and Rich Wolski. Using effective network views to promote distributed application performance. In *International Conference on Parallel and Distributed Processing Techniques and Applications*. CSREA Press, June 1999. Available at http://apples.ucsd.edu/pubs/pdpta99.ps.

[93] J. Sohn, T.G. Robertazzi, and S. Luryi. Optimizing computing costs using divisible load analysis. *IEEE Transactions on parallel and distributed systems*, 9(3):225–234, March 1998.

[94] H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, Kenjiro Taura, and Andrew A. Chien. The microgrid: a scientific tool for modeling computational grids. In *Supercomputing*, 2000. Available at http://www.sc2000.org/techpapr/papers/pap.pap286.pdf.

[95] K. Taura and A. A. Chien. A heuristic algorithm for mapping communicating tasks on heterogeneous resources. In *Heterogeneous Computing Workshop*, pages 102–115. IEEE Computer Society Press, 2000.

[96] Teragrid project. Website http://www.teragrid.org/.

[97] H. Topcuoglu, S. Hariri, and M.-Y. Wu. Task scheduling algorithms for heterogeneous processors. In *Eighth Heterogeneous Computing Workshop*. IEEE Computer Society Press, 1999.

[98] R.Y. Wang, A. Krishnamurthy, R.P. Martin, T.E. Anderson, and D.E. Culler. Modeling communication pipeline latency. In *Measurement and Modeling of Computer Systems (SIGMETRICS'98)*, pages 22–32. ACM Press, 1998.

[99] Bernard M. Waxman. Routing of Multipoint Connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, December 1988.

[100] R. Clint Whaley and Jack Dongarra. Automatically tuned linear algebra software. In *Proceedings of Supercomputing '98*. Sponsored by ACM SIGARCH and IEEE Computer Society, 1998. Winner, best paper in the systems category, SC98: High Performance Networking and Computing.

[101] R. Wolski, N.T. Spring, and J. Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(10):757–768, 1999.

[102] Richard Wolski. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, 1(1):119–132, 1998. Available at http://www.cs.ucsd.edu/users/rich/papers/nws-tr.ps.gz.

[103] Tao Yang and Apostolos Gerasoulis. Pyrros: static task scheduling and code generation for message passing multiprocessors. In *Proceedings of the 6th international conference on Supercomputing*, pages 428–437. ACM Press, 1992.

[104] Wenci Yu. *The two-machine flow shop problem with delays and the one-machine total tardiness problem.* PhD thesis, Technishe Universiteit Eidhoven, June 1996.

Appendix B

Curriculum Vitae

État civil

	Olivier Beaumont
né le	16 septembre 1970 à Saint Etienne (42)
Nationalité	Française
Position actuelle	Maître de Conférences à l'ENSEIRB - Université de Bordeaux 1
Adresse	LaBRI - UMR CNRS 5800 - Équipe INRIA Futurs Scalapplix Campus Universitaire 351, cours de la libération 33405 Talence cedex Tel. : (+33) 5 4000 3798
Email	Olivier.Beaumont@labri.fr

Cursus et expérience professionnelle

2001-2004 **Maître de conférences** à l'ENSEIRB.
Recherches effectuées dans l'équipe **Scalapplix** (UMR-CNRS 5800 INRIA Futurs)

1999-2001 **Maître de conférences** à l'ENS Lyon.
Recherches effectuées dans l'équipe **Remap** (UMR-CNRS 5668 INRIA Rhône-Alpes)

1998-1999 **ATER à l'Université de Rennes 1.**
Recherches effectuées dans l'équipe **Aladin** (UMR-CNRS 6074 IRISA)

12 janvier 1999 **Thèse** (Informatique) de l'Université de Rennes 1 (mention: très honorable).
Titre: Algorithmique pour les intervalles: comment obtenir un résultat sûr quand les données sont incertaines ?
Jury:

M.	Jean-Pierre	BANÂTRE	Pr. Informatique, Rennes 1	Président
MM.:	Jean-Marie	CHESNEAUX	Pr. Informatique, Paris 6	
	Jean-Michel	MULLER	CR CNRS, ENS Lyon	
	Jiří	ROHN	Pr. Mathématiques, Prague	Rapporteurs
M.	Siegfried	RUMP	Pr. Informatique, Hambourg	
M.	Bernard	PHILIPPE	DR Inria, Rennes	Directeur de thèse

1995-1999 **Allocataire Moniteur Normalien** à l'Université de Rennes 1

1994-1995 **Service Militaire** et **Agrégation de Mathématiques**

1991-1996 **Étudiant à l'École Normale Supérieure de Lyon**
Magistère MIM (Magistère Informatique et Modélisation)
1993-1994: **DEA en Informatique** (Mention Bien) ENSL / Lyon 1.
1993: Stage de 3 mois à l'ANU (Canberra) sous la direction de R.P. Brent

B.1 Fiche Synthétique

B.1.1 Laboratoires de recherche et Publications

- 2001-2004: **Maître de conférences** à l'ENSEIRB (équipe Scalapplix).
- 1999-2001: **Maître de conférences** à l'ENS Lyon (équipe Remap).
- 1995-1999 **Allocataire Moniteur Normalien** puis **ATER** à l'Université de Rennes 1 (équipe Aladin).
- **Titulaire de la PEDR** depuis 2001
- **14 publications dans des journaux internationaux**
- **32 publications dans des conférences internationales** (ou chapitres de livres)

B.1.2 Encadrement d'étudiants

- **2 étudiants de DESS ou troisième année d'École d'ingénieur**
  Arnaud Münch, Rennes, 1999 (co-encadré avec Bernard Philippe)
  Frédéric Raoult, Bordeaux, 2002 (co-encadré avec Jean Roman)
- **2 étudiants de DEA**
  Arnaud Legrand, Lyon, 2000 (co-encadré avec Yves Robert)
  Cyril Banino, Bordeaux, 2002
- **2 étudiants de thèse**
  Arnaud Legrand, Lyon, 2000-2003 (co-encadré avec Yves Robert)
  Loris Marchal, Lyon, 2003-... (co-encadré avec Yves Robert)
- **Participation à l'encadrement de 2 étudiants de thèse**
  Vincent Boudet, Lyon, 1999-2002
  Fabrice Rastello, Lyon, 1999-2001
- **1 étudiant en stage post-doctoral**
  Wahid Nasri, Bordeaux, 2004
- **Tutorat et encadrement de stages industriels** d'une quinzaine d'étudiants par an à l'ENSEIRB depuis 2002.

B.1.3 Responsabilités administratives

- Co-responsable (Avec Alexey Kalinov) d'un **projet de l'Institut Franco-Russe Liapunov** (INRIA - Université de Moscou), 2000-2002
- **Responsable de la $2^{\text{ième}}$ année de la filière Télécommunication** à l'ENSEIRB depuis 2001
- Co-responsable (avec Jean Roman) de l'organisation des **journées GRID2** à Bordeaux (2002)
- Responsable des **achats de logiciel et de matériel** pour le laboratoire (LIP) et le département informatique (DMI) (ENS Lyon) 1999-2001
- Responsable du **groupe de travail Scalapplix-Runtime** (2003-...)
- Élu à la **commission de spécialistes** de Bordeaux (2003, renouvellement partiel et 2004)
- Co-responsable (avec Daniel Hirschkoff) des **stages de première année** à l'ENS Lyon (1999-2001)

B.1.4 Collaborations académiques (régulières) et industrielles

- Moscou (ISPRAS) avec **Alexey Kalinov**. Séjour de 15 jours à Moscou en 2001
- San Diego (UCSD) avec **Larry Carter, Henri Casanova et Jeanne Ferrante**. Séjour de 10 jours en 2003
- **ACI INRIA Fiable**
- **ACI INRIA GRID2**
- Contrat (Jocelyne Erhel) avec **Dassault Aviation** (98-99)

B.1.5 Comités de programme et arbitrages

- **Comités de programme**: PMAA'02, HeteroPar'03, HeteroPar'04, RenPar'05, IPDPS'05
- **Arbitrages pour les revues internationales**: IEEE TPDS (4 à 5 articles par an), Parallel Computing, Reliable Computing, Journal of Computing and Information Technology, EJOR, Computer Journal, IJHPCA.
- **Arbitrages (réguliers) pour les conférences internationales** : Cluster, STACS, EuroPar, IPDPS, SuperComputing, HPCSE...
- Expert auprès de la SFI (**Science Fundation Ireland**) depuis 2003

B.1.6 Exposés invités

- **Cours doctoral** à l'ISSTT à Tunis (2002)
- Cours (avec Denis Trystram) sur l'ordonnancement lors de **RenPar 2002**
- Exposés à **WASC'04** (Bertinoro) et au **Workshop on Scheduling and Grid Computing 04** (Aussois)

Tous les points évoqués précédemment seront développés lors de la description des activités de recherche et d'enseignement.

B.2 Description des activités de recherche

B.2.1 Algorithmique d'intervalles (1995-1999) IRISA, Rennes 1

B.2.1.1 Description

Pendant ma thèse, je me suis intéressé à la validation numérique de quelques algorithmes (algèbre linéaire et évaluation de polynômes), compte tenu à la fois de l'imprécision sur la connaissance des données et de la représentation des nombres en machine. L'arithmétique d'intervalles, dans laquelle les flottants sont remplacés par des intervalles, est une technique séduisante pour représenter les nombres en machine dans ce contexte. En effet, on peut utiliser la possibilité, offerte par la norme IEEE-754, d'arrondir de manière garantie les résultats des opérations élémentaires (+,-,*,/) sur des données de type intervalle.

La difficulté provient alors de l'organisation des calculs. En effet, plus encore que dans le cas des flottants, le résultat de l'évaluation d'une expression arithmétique dépend fortement de l'ordre dans lequel les opérations sont réalisées. Par exemple
$[-1,1] * ([2,3] - [1,2]) \subseteq [-1,1] * ([0,2]) \subseteq [-2,2]$ et
$[-1,1] * ([2,3] - [1,2]) \subseteq [-3,3] - [-2,2] \subseteq [-5,5]$.
Toute l'astuce consiste donc à déterminer un algorithme qui induit une croissance minimale de l'intervalle résultat. De façon surprenante, les meilleurs algorithmes connus en algorithmique d'intervalles ne sont pas ceux qui réalisent un minimum d'opérations (problème déjà étudié dans le cas scalaire), ce qui laisse la place à des algorithmes ad-hoc et originaux. Les principales originalités de l'approche que j'ai développée pendant ma thèse sont

- l'utilisation de la programmation linéaire (déjà...) pour résoudre des problèmes d'algèbre linéaire (pour différents type de résolution de systèmes linéaires) et
- la représentation des données intermédiaires sous la forme de boîtes obliques (c'est à dire de vecteurs d'intervalles dont les côtés ne sont pas nécessairement parallèles aux axes) pour la résolution des équations différentielles ordinaires.

B.2.1.2 Diffusion des résultats

Ces travaux ont fait l'objet de 2 publications dans des journaux internationaux [1, 2], de 3 publications dans des conférences internationales [15, 16, 17] et de 2 chapitres de livre [18, 19].

B.2.1.3 Encadrement d'étudiants

Pendant ma thèse, j'ai eu l'occasion d'encadrer un stage de DESS (avec Bernard Philippe) et un stage de magistère (avec Philippe Chartier). Le stage d'Arnaud Münch (DESS) portait sur l'utilisation de l'arithmétique d'intervalles pour certaines primitives de géométrie algorithmique. Le stage de magistère portait sur la résolution d'équations différentielles ordinaires quand les données initiales ne sont pas connues exactement (mais données sous la forme d'intervalles).

B.2.1.4 Collaborations académiques et industrielles

Entre juin 97 et juin 99, j'ai participé à l'ACI INRIA Fiable, qui incluait différents projets INRIA Aladin, Eureca, Prisme, Arenaire et Safir et le CHPV(Paris VI). Le but de ce projet était la conception de méthodes sûres pour le calcul de déterminants et l'évaluation de polynômes (avec des applications en géométrie algorithmique et en robotique).

En 98 et 99, j'ai par ailleurs participé (avec Jocelyne Erhel) au contrat Genie2, qui liait l'INRIA et Dassault Aviation. Notre but était de valider numériquement (compte tenu des erreurs sur le calcul flottant) un logiciel de calcul de trajectoires développé par Dassault et dont la validité avait été prouvée (dans le cadre du calcul exact) à l'aide de Coq.

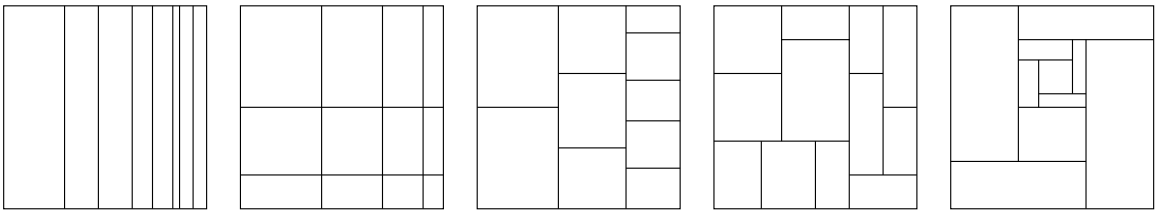


Figure B.1: Taxonomie des partitions envisagées du carré unité

	1D	2D	En colonne	Récursive	Quelconque
$\sum$	Polynomial	NP-complet	Polynomial	Pas de résultat connu.	NP-complet. Heuristique garantie à un facteur 5/4
max			NP-complet Heuristique garantie à un facteur $2/\sqrt{3}$.	Pas de résultat connu.	NP-complet. Heuristique garantie à un facteur $2/\sqrt{3}$

Figure B.2: Résultats de complexité

B.2.2 Algèbre linéaire sur plates-formes hétérogènes (1999-2001) LIP, ENS Lyon

B.2.2.1 Description

J'ai été nommé maître de conférences à l'ENS Lyon en septembre 99. J'ai alors intégré l'équipe Remap et j'ai opéré une reconversion thématique, de l'arithmétique des ordinateurs vers le parallélisme. Pendant deux ans, nous avons travaillé sur la distribution des données pour des problèmes d'algèbre linéaire (essentiellement le produit de matrices et la factorisation LU) dans le cas de processeurs hétérogènes.

Dans ce cadre, si on s'intéresse à l'équilibrage de charge, chaque processeur doit recevoir une quantité de données proportionnelle à sa vitesse relative. On peut donc se ramener à la partition d'un carré unité en blocs de surfaces prédéfinies. Nous avons étudié différents types de partitionnements possibles, indiqués à la Figure B.1. En plus de l'équilibrage de charge, il est intéressant de considérer la minimisation du volume de communications échangées pendant le déroulement de l'algorithme. Nous avons considéré deux modèles extrêmes et grossiers pour modéliser les communications en supposant

- que toutes les communications peuvent se dérouler en parallèle, ce qui revient à minimiser la surface du plus grand bloc (problème noté max)
- que toutes les communications doivent se dérouler de manière séquentielle, ce qui revient à minimiser la somme des périmètres de tous les blocs (problème noté $\sum$).

Nous avons obtenu différents résultats de complexité et nous avons construit des heuristiques garanties, pour différents types de partitionnement et différents modèles de coût de communication. Ces résultats sont résumés à la Figure B.2. Ces travaux ont été menés dans le cadre des thèses de Fabrice Rastello et de Vincent Boudet.

Dans le cadre du stage de DEA d'Arnaud Legrand, nous nous sommes intéressés à la conception de versions dynamiques de ces algorithmes de distribution, quand les performances des différents processeurs varient au

cours du temps (en environnement non dédié). Nous avons alors proposé des schémas de redistribution légère pour équilibrer la charge, sans remettre complètement en cause la distribution initiale. En outre, tous ces résultats ont été validés expérimentalement sur des réseaux de stations hétérogènes à l'ENS.

B.2.2.2 Diffusion des résultats

Ces travaux ont fait l'objet de 5 publications dans des journaux internationaux [3, 4, 5, 6, 7], de 7 publications dans des conférences internationales [20, 21, 22, 23, 24, 25, 36] et d'un chapitre de livre [30]. Par ailleurs ces travaux ont fait l'objet d'un exposé invité à Renpar 2002 (avec Denis Trystram) et d'un cours doctoral à l'Université de Tunis (ISSTT).

B.2.2.3 Encadrement d'étudiants

Entre 1999 et 2001, j'ai participé (mais pas officiellement, puisque les deux thèses avaient commencé avant mon arrivée à Lyon) à l'encadrement des thèses de Vincent Boudet (avec Yves Robert) et Fabrice Rastello (avec Frédéric Desprez et Yves Robert). Vincent Boudet est actuellement maître de conférences à l'Université de Montpellier (LIRMM) et Fabrice Rastello est actuellement CR2 INRIA au sein du projet CompSys à l'ENS Lyon.

Par ailleurs, j'ai co-encadré (avec Yves Robert) le stage de DEA d'Arnaud Legrand, puis (officiellement cette fois) sa thèse (avec Yves Robert) entre 2000 et 2003. Arnaud Legrand est actuellement CR2 CNRS dans le laboratoire ID (Grenoble).

B.2.2.4 Collaborations académiques et industrielles

Les problèmes de distribution de données pour des problèmes d'algèbre linéaire en environnement hétérogène avaient déjà été abordés par Alexey Kalinov, de l'Université de Moscou (ISPRAS). Entre 2000 et 2002, j'ai été co-responsable (avec Alexey Kalinov) d'un projet de l'Institut Franco-Russe Liapunov (INRIA - Université de Moscou) intitulé "Distribution des données sur des réseaux hétérogènes de stations pour des problèmes d'algèbre linéaire". À cette occasion, je suis allé pendant 15 jours en 2001 à Moscou pour travailler sur l'utilisation du langage MpC (développé à l'ISPRAS) pour décrire les schémas de distribution de données sur processeurs hétérogènes. Cette collaboration se poursuit depuis, elle a abouti entre autres à la création des conférences HeteroPar (j'ai été membre des comités de programme d'HeteroPar'03 et HeteroPar'04).

B.2.3 Ordonnancement sur plates-formes hétérogènes: (2001-2004) LaBRI, Bordeaux

B.2.3.1 Description

En 2001, j'ai demandé et obtenu pour raisons personnelles (rapprochement de conjoint) ma mutation à Bordeaux (ENSEIRB). Je ne détaillerai pas la description de cette activité de recherche, puisqu'elle a fait l'objet du document principal de cette habilitation. Nous nous sommes intéressés à la conception d'algorithmes d'ordonnancement pour des problèmes de type maître-esclave sur des plates-formes hétérogènes dans lesquelles l'hétérogénéité des liens de communication et des ressources de calcul est explicitement prise en compte. Après avoir démontré la difficulté de problèmes très simples dans le cadre de la minimisation du temps de complétion (makespan), nous nous sommes intéressés à la complexité de ces problèmes dans le cadre de la maximisation du débit (ce qui suppose la présence d'un grand nombre de tâches indépendantes à traiter). Ces travaux ont fait l'objet de la fin de la thèse d'Arnaud Legrand.

Plus récemment, nous avons considéré la conception d'algorithmes d'ordonnancement de débit optimal pour des problèmes de communication collective (diffusion, diffusion partielle...). Ces travaux ont été menés dans le cadre de la thèse de Loris Marchal, que je co-encadre (avec Yves Robert) depuis septembre 2003. Par ailleurs, nous avons également travaillé sur le modèle des tâches divisibles (tâches parallèles idéales

qui peuvent être arbitrairement découpées), qui ressemble par certains aspects à la distribution d'un grand nombre de tâches indépendantes.

Par ailleurs, je travaille également sur l'application de ces techniques à l'ordonnancement de graphes de tâches issus de factorisations de grandes matrices creuses avec Pascal Henon, Pierre Ramet et Jean Roman.

B.2.3.2 Diffusion des résultats

Les travaux sur l'ordonnancement de graphes de tâches et de communications collectives ont fait l'objet de 5 publications dans des journaux internationaux [8, 9, 11, 12, 14] et de 11 publications dans des conférences internationales [25, 26, 29, 31, 32, 35, 38, 40, 41, 42, 43]. Les travaux sur les tâches divisibles ont fait l'objet de deux publications dans des revues internationales [10, 13] et dans des conférences internationales [39, 46].

J'ai été invité à présenter ces résultats cette année lors des workshops WASC'04 (Bertinoro) et "Workshop on scheduling and Grid Computing" (Aussois).

B.2.3.3 Encadrement d'étudiants

Ces travaux correspondent à la fin de la thèse d'Arnaud Legrand et au début de celle de Loris Marchal. Ces travaux ont également fait l'objet du stage de DEA de Cyril Banino à Bordeaux. Cyril Banino est actuellement en thèse à Trondheim (NTNU) et nous continuons à collaborer ensemble sur ces questions (cette collaboration devrait conduire à une co-tutelle l'année prochaine). L'application des techniques d'ordonnancement aux graphes de tâches issus de factorisation de matrices creuses a fait l'objet du stage (troisième année de l'ENSEIRB) de Frédéric Raoult (co-encadré avec Jean Roman) et du stage post-doctoral de Wahid Nasri (ISSTT, Tunis) à Bordeaux cette année (3 mois).

B.2.3.4 Collaborations académiques et industrielles

Les problématiques de maximisation du débit ont été abordées pour la première fois lors du séjour de Larry Carter et Jeanne Ferrante (UC San Diego) à Lyon en 2001, lors d'un séjour sabbatique. Cette collaboration se poursuit depuis dans le cadre d'une équipe associée entre l'ENS Lyon et l'UC San Diego. En outre, les travaux sur les tâches divisibles ont été partiellement menés, lors d'un séjour à San Diego (10 jours en 2003), avec Henri Casanova et Yang Yang.

B.3 Activités d'enseignement

B.3.1 Disciplines enseignées et public

Mon but n'est pas ici de détailler l'ensemble des matières que j'ai eu l'occasion d'enseigner depuis mon monitorat à l'université de Rennes 1. J'insisterai donc essentiellement sur les modules dont j'ai été responsable (cours et projets) et qui représentent une durée significative (plus de 60h par an). Il est à noter que les deux enseignements sur lesquels je vais insister ne font pas partie de mon domaine de compétences naturel.

À l'ENS Lyon, j'ai été chargé de monter le cours de compilation (32h de cours et 32h de projet). La principale originalité de ce cours était de se concentrer sur la partie optimisation de code, puisque les analyses syntaxique et sémantique étaient déjà largement abordées par ailleurs. En outre, j'ai introduit le projet de compilation du langage à objet fortement typé COOL, développé à Berkeley par Alex Aiken et dont une description peut être trouvée à l'adresse: `http://www.cs.berkeley.edu/ aiken/cool/`.
À l'ENSEIRB, j'ai été chargé de monter le cours de systèmes d'exploitation (45h de cours et 24h de projet). Il s'agit d'un cours de systèmes d'exploitation assez classique, mais qui est directement relié au projet Nachos, qui a été développé (encore) à l'Université de Berkeley par Tom Anderson, et dont une description peut être trouvée à la page `http://www.cs.washington.edu/homes/tom/nachos/`
Dans les deux cas, la principale originalité (et richesse de mon point de vue) des cours que j'ai enseignés est de s'appuyer sur de gros projets de développement logiciel, qui permettent de comprendre en profondeur la discipline enseignée.

Outre ces enseignements, j'ai également assuré

- des TDs (24h) et des TPs (24h) de "programmation fonctionnelle" (SCHEME) (Deug, Rennes 1)
- le cours d'"initiation à l'algorithmique" (12h mais 230 étudiants) (première année ENSEIRB, toutes filières)
- les TDs d'"algorithmique parallèle" (32h) (deuxième année, ENS Lyon)
- le cours de DEA (6h) de remise à niveau pour les étudiants de mathématiques suivant des cours du DEA d'informatique
- le cours d'"algorithmique parallèle" (24h) (DEA DIF, ENS Lyon)
- le cours "graphe et routages dans les réseaux" (12h) (deuxième année, ENSEIRB)
- le cours "ordonnancement" (10h) (troisième année, ENSEIRB)
- les TPs du cours "réseaux rapides" (8h) (deuxième année, ENSEIRB)
- les TDs de graphe (24h) (Deug, Univ. Lyon 1)
- le cours "précision des calculs sur ordinateur" (10h) (DESS, Rennes 1)

B.3.2 Responsabilités

À l'ENS Lyon, j'ai été co-responsable (avec Daniel Hirschkoff) de l'organisation des stages de première année du magistère MIM (recherche de sujets, suivi des stages, organisation des soutenances).

À l'ENSEIRB, je suis responsable de la deuxième année de la filière Télécommunication (environ 50 étudiants par an), filière dirigée par Charles Consel. À ce titre, je suis chargé (en collaboration avec C. Consel) de la constitution du programme et du choix des enseignants. Je suis également chargé de la réalisation des emplois du temps et du recrutement des étudiants de deuxième année qui n'ont pas suivi la première année à l'ENSEIRB. En outre, je suis le tuteur chaque année d'une quinzaine d'étudiants, dont je dois suivre la scolarité et le déroulement des stages industriels.

B.4 Liste des publications personnelles

Revues internationales

[1] O. Beaumont. «Solving interval linear systems with linear programming techniques». *Linear Algebra and its Applications* (1998), 293–309.

[2] O. Beaumont et B. Philippe. «Linear Interval Tolerance Problem and Linear Programming Techniques». *Reliable Computing* **7**, numéro 6 (2001), 433–447.

[3] O. Beaumont, A. Legrand, F. Rastello et Y. Robert. «Static LU decomposition on heterogeneous platforms». *Int. Journal of High Performance Computing Applications* **15**, numéro 3 (2001), 310–323.

[4] O. Beaumont, V. Boudet, F. Rastello et Y. Robert. «Matrix multiplication on heterogeneous platforms». *IEEE Trans. Parallel Distributed Systems* **12**, numéro 10 (2001), 1033–1051.

[5] O. Beaumont, V. Boudet, A. Petitet, F. Rastello et Y. Robert. «A proposal for a heterogeneous cluster ScaLAPACK (dense linear solvers)». *IEEE Trans. Computers* **50**, numéro 10 (2001), 1052–1070.

[6] O. Beaumont, A. Legrand, F. Rastello et Y. Robert. «Dense linear algebra kernels on heterogeneous platforms: Redistribution issues». *Parallel Computing* **28** (2002), 155–185.

[7] O. Beaumont, V. Boudet, F. Rastello et Y. Robert. «Partitioning a square into rectangles: NP-completeness and approximation algorithms». *Algorithmica* **34** (2002), 217–239.

[8] O. Beaumont, A. Legrand et Y. Robert. «Static scheduling strategies for heterogeneous systems». *Computing and Informatics* **21** (2002), 413–430.

[9] O. Beaumont, A. Legrand et Y. Robert. «The master-slave paradigm with heterogeneous processors». *IEEE Trans. Parallel Distributed Systems* **14**, numéro 9 (2003), 897–908.

[10] O. Beaumont, A. Legrand et Y. Robert. «Scheduling divisible workloads on heterogeneous platforms». *Parallel Computing* **29** (2003), 1121–1152.

[11] O. Beaumont, A. Legrand, L. Marchal et Y. Robert. «Scheduling strategies for mixed data and task parallelism on heterogeneous clusters». *Parallel Processing Letters* **13**, numéro 2 (2003).

[12] C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand et Y. Robert. «Scheduling strategies for master-slave tasking on heterogeneous processor platforms». *IEEE Trans. Parallel Distributed Systems* **15**, numéro 4 (2004), 319–330.

[13] O. Beaumont, H. Casanova, A. Legrand, Y. Robert et Y. Yang. «Scheduling divisible loads on star and tree networks: results and open problems». *IEEE Trans. Parallel Distributed Systems* (2004, to appear).

[14] O. Beaumont, A. Legrand, L. Marchal et Y. Robert. «Pipelining broadcasts on heterogeneous platforms». *IEEE Trans. Parallel Distributed Systems* (2004, to appear).

Conférences internationales avec comité de lecture et chapitres de livres

[15] O. Beaumont. «A Simplex-like Method for Interval Linear Systems». Dans *Interval'96* (1996).

[16] O. Beaumont et B. Philippe. «An iterative algorithm for interval eigenvalue problems». Dans *SCAN'97* (1997).

[17] O. Beaumont. «Solving interval linear systems with oblique boxes.». Dans *SCAN'98* (1998).

[18] O. Beaumont et B. Philippe. «Arithmétique d'intervalles». Dans *Qualité des Calculs sur Ordinateurs. Vers des arithmétiques plus fiables?*, M. Daumas et J.-M. Muller, éditeurs. Masson, 1997, chapitre 4.

[19] O. Beaumont, J. Erhel et B. Philippe. «Aquarels : a problem-solving environment for validating scientific software». Dans *Enabling technologies for computational science*, Houstis, Rice, Gallopoulos et Bramley, éditeurs. Kluwer Academic Publishers, Boston, 2000.

[20] O. Beaumont, V. Boudet, F. Rastello et Y. Robert. «Load balancing strategies for dense linear algebra kernels on heterogeneous two-dimensional grids». Dans *14th International Parallel and Distributed Processing Symposium (IPDPS'2000)* (2000), IEEE Computer Society Press, pp. 783–792.

[21] O. Beaumont, V. Boudet, F. Rastello et Y. Robert. «Matrix-matrix multiplication on heterogeneous platforms». Dans *2000 International Conference on Parallel Processing (ICPP'2000)* (2000), IEEE Computer Society Press, pp. 289–298.

[22] O. Beaumont, V. Boudet, A. Legrand, F. Rastello et Y. Robert. «Heterogeneity Considered Harmful to Algorithm Designers». Dans *Cluster'2000* (2000), IEEE Computer Society Press, pp. 403–404.

[23] O. Beaumont, V. Boudet, A. Legrand, F. Rastello et Y. Robert. «Dense linear algebra kernels on heterogeneous platforms». Dans *Parallel Matrix Algorithms and Applications* (2000), Université de Neuchâtel. Voir http://www.unine.ch/iiun/matrix/seminars/pmaa2000/sessions.html.

[24] O. Beaumont, V. Boudet, A. Legrand, F. Rastello et Y. Robert. «Heterogeneous Matrix-Matrix Multiplication, or Partitioning a Square into Rectangles: NP-Completeness and Approximation Algorithms». Dans *EuroMicro Workshop on Parallel and Distributed Computing (EuroMicro'2001)* (2001), IEEE Computer Society Press, pp. 298–305.

[25] O. Beaumont, A. Legrand et Y. Robert. «Master-slave tasking with heterogeneous processors». Dans *2001 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2001)* (2001), CSREA Press, pp. 857–863.

[26] O. Beaumont, A. Legrand et Y. Robert. «The master-slave paradigm with heterogeneous processors». Dans *Cluster'2001* (2001), D. Katz, T. Sterling, M. Baker, L. Bergman, M. Paprzycki et R. Buyya, éditeurs, IEEE Computer Society Press, pp. 419–426.

[27] O. Beaumont, V. Boudet et Y. Robert. «The iso-level scheduling heuristic for heterogeneous processors». Dans *PDP'2002, 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing* (2002), IEEE Computer Society Press.

[28] O. Beaumont, V. Boudet et Y. Robert. «A realistic model and an efficient heuristic for scheduling with heterogeneous processors». Dans *HCW'2002, the 11th Heterogeneous Computing Workshop* (2002), IEEE Computer Society Press.

[29] O. Beaumont, L. Carter, J. Ferrante, A. Legrand et Y. Robert. «Bandwidth-centric allocation of independent tasks on heterogeneous platforms». Dans *International Parallel and Distributed Processing Symposium IPDPS'2002* (2002), IEEE Computer Society Press.

[30] O. Beaumont, V. Boudet, A. Legrand, F. Rastello et Y. Robert. «Static Data Allocation and Load Balancing Techniques for Heterogeneous Systems». Dans *Annual Review of Scalable Computing*, C. Yuen, éditeur, volume 4. World Scientific, 2002, chapitre 1, pp. 1–37.

[31] C. Banino, O. Beaumont, A. Legrand et Y. Robert. «Scheduling strategies for master-slave tasking on heterogeneous processor grids». Dans *PARA'02: International Conference on Applied Parallel Computing* (2002), LNCS 2367, Springer Verlag, pp. 423–432.

[32] O. Beaumont, A. Legrand et Y. Robert. «Static scheduling strategies for heterogeneous systems». Dans *ISCIS XVII, Seventeenth International Symposium On Computer and Information Sciences* (2002), CRC Press.

[33] O. Beaumont, V. Boudet, F. Desprez, P. Ramet, J. Roman et C. Travers. «Modélisation de pipelines hétérogènes». Dans *GRID'2002, Aussois, France* (2002).

[34] O. Beaumont, P. Ramet et J. Roman. «Asymptotically optimal algorithm for Laplace task graphs on heterogeneous platforms». Dans *Fifth International Conference on Parallel Processing and Applied Mathematics, Workshop HeteroPar, Czestochowa, Pologne* (2003).

[35] O. Beaumont, A. Legrand et Y. Robert. «A polynomial time algorithm for allocating independent tasks on heterogeneous fork-graphs». Dans *ISCIS'02, 17th International Symposium on Computer and Information Sciences* (2002), CRC Press.

[36] O. Beaumont, A. Legrand et Y. Robert. «Static scheduling strategies for dense linear algebra kernels on heterogeneous clusters». Dans *Parallel Matrix Algorithms and Applications* (2002), Université de Neuchâtel.

[37] O. Beaumont, A. Legrand et Y. Robert. «Ordonnancement en régime permanent pour plateformes hétérogènes». Dans *GRID'2002, Actes de l'école thématique sur la globalisation des ressources informatiques et des données* (2002), INRIA Lorraine, pp. 325–334.

[38] O. Beaumont, A. Legrand et Y. Robert. «Scheduling strategies for mixed data and task parallelism on heterogeneous clusters and grids». Dans *PDP'2003, 11th Euromicro Workshop on Parallel, Distributed and Network-based Processing* (2003), IEEE Computer Society Press, pp. 209–216.

[39] O. Beaumont, A. Legrand et Y. Robert. «Optimal algorithms for scheduling divisible workloads on heterogeneous systems». Dans *HCW'2003, the 12th Heterogeneous Computing Workshop* (2003), IEEE Computer Society Press.

[40] O. Beaumont, A. Legrand, L. Marchal et Y. Robert. «Pipelining Broadcasts on Heterogeneous Platforms». Dans *IPDPS'2004* (2004), IEEE Computer Society Press.

[41] O. Beaumont, A. Legrand, L. Marchal et Y. Robert. «Steady-state scheduling on heterogeneous clusters: why and how?». Dans *6th Workshop on Advances in Parallel and Distributed Computational Models APDCM 2004* (2004), IEEE Computer Society Press.

[42] O. Beaumont, A. Legrand, L. Marchal et Y. Robert. «Complexity results and heuristics for pipelined multicast operations on heterogeneous platforms». Dans *2004 International Conference on Parallel Processing (ICPP'2004)* (2004), IEEE Computer Society Press, pp. 267–274.

[43] O. Beaumont, A. Legrand, L. Marchal et Y. Robert. «Assessing the impact and limits of steady-state scheduling for mixed task and data parallelism on heterogeneous platforms». Dans *HeteroPar'2004: International Conference on Heterogeneous Computing, jointly published with ISPDC'2004: International Symposium on Parallel and Distributed Computing* (2004), IEEE Computer Society Press.

[44] O. Beaumont, E. Daoudi, N. Maillard, P. Manneback et J.-L. Roch. «Tradeoff to minimize extra-computations and stopping criterion tests for parallel iterative schemes». Dans *PMAA'04 Parallel Matrix Algorithms and Applications* (2004), CIRM, Marseille.

[45] O. Beaumont, V. Boudet, P.-F. Dutot, Y. Robert et D. Trystram. *Informatique répartie : architecture, parallélisme et système*. Hermès Publications, 2004, chapitre "Fondements théoriques pour la conception d'algorithmes efficaces de gestion de ressources", pp. ??–??

[46] O. Beaumont, A. Legrand, L. Marchal et Y. Robert. «Independent and Divisible Task Scheduling on Heterogeneous Star-shaped Platforms with Limited Memory». Dans *13th Euromicro Conference on Parallel, Distributed and Network-based Processing* (February 2005, accepted for publication), IEEE Computer Society Press.

Appendix C

Arithmétique d'intervalles

L'article suivant (publié dans "Linear Algebra and its Applications") présente une méthode originale pour calculer la solution d'un programme linéaire $[A]x = [b]$, dans lequel $[A] = [A_c - \Delta A, A_c + \Delta A]$ et $[b] = [b_c - \Delta b, b_c + \Delta b]$ sont respectivement une matrice et un vecteur d'intervalles (voir Paragraphe B.2.1).

On cherche la solution de ce système sous la forme d'un vecteur d'intervalles $[x]$, contenant l'ensemble des solutions possibles de systèmes linéaires extraits, i.e.

$$\mathcal{S} = \{x, \quad \exists A \in [A], \exists b \in [b], \quad Ax = b\}.$$

Pour cela, on s'appuie sur un théorème de Oettli et Prager qui décrit $\mathcal{S}$ sous la forme

$$x \in \mathcal{S} \Longleftrightarrow |A_c x - b_c| \leqslant \Delta A|x| + \Delta|b|$$

et on utilise la programmation linéaire, après avoir linéarisé le terme $|x|$, pour obtenir un encadrement du vecteur solution.

ELSEVIER Linear Algebra and its Applications 281 (1998) 293–309

LINEAR ALGEBRA AND ITS APPLICATIONS

Solving interval linear systems with linear programming techniques

O. Beaumont [1]

Campus Beaulieu, Institut de Recherche en Informatique et Systemes Aleatoires, 35042 Rennes, France

Received 13 June 1997; accepted 16 January 1998

Submitted by H. Schneider

Abstract

In this paper, we show how it is possible to use convex polyhedra for solving linear interval systems without using preconditioning. We first show how to derive, from an enclosure of $\square\Sigma([A],[b])$, a polyhedron which contains the convex hull of the solution set. Then, a simplex-like method enables us to find a new outer inclusion. Moreover, the constraints obtained may be used to compute an inner inclusion of $\square\Sigma([A],[b])$.

Keywords: Interval computations; Linear systems; Linear programming

1. Introduction

A lot of work [5] has been done in order to solve interval linear systems $[A]x = [b]$. Rohn and Kreinovich [9,3] have proved that the calculation of $\square\Sigma([A],[b])$, the smallest box that contains all the solutions of $[A]x = [b]$, is an NP-hard problem. On the other hand, several algorithms obtain good results, especially when the diameter of $[A]$ is small [7,5]. In this paper, we propose a new algorithm which is based on linear programming. We suppose that we know an enclosure of $\square\Sigma([A],[b])$, which implies that $[A]$ is regular. It consists of an iterative scheme which considers as input an enclosure of the solution of $\square\Sigma([A],[b])$ and returns an (usually better) enclosure.

[1] E-mail: obeaumon@irisa.fr.

0024-3795/98/$19.00
PII: S0024-3795(98)10044-7

This algorithm converges toward a superset of the convex hull of the united solution set $\Sigma([A],[b])$. In this paper, we use the following notations:

$$\Sigma([A],[b]) = \{x, \exists A \in [A],\ \exists b \in [b],\ Ax = b\},$$

$\Gamma([A],[b])$ denotes the convex hull of $\Sigma([A],[b])$,

$\square\Sigma([A],[b])$ denotes the interval hull of $\Sigma([A],[b])$.

The interval linear systems we consider in this paper are defined by the following notations:

$$[A] = [A_c - \Delta A, A_c + \Delta A], \qquad [b] = [\underline{b}, \bar{b}] = [b_c - \Delta b, b_c + \Delta b],$$

where ΔA and Δb are respectively nonnegative matrix and vector.

2. How to find a polyhedron that contains the convex hull of the united solution set

It is known [5] that the solution of the problem $[A]x = [b]$ is in general not convex. As far as we are only interested in $\square\Sigma([A],[b])$, we may use $\Gamma([A],[b])$ as intermediate set. Oettli and Prager [6] proved the following theorem.

Theorem 1.

$$(\exists A \in [A],\ \exists b \in [b],\ Ax = b) \iff |A_c x - b| \leqslant \Delta A|x| + \Delta b.$$

This expression does not lead directly to a polyhedron, because of the absolute value, which underlines the fact that the solution set is usually not a convex set.

We give a first result which provides a way to get rid of the absolute values.

Lemma 1. *If* $[\underline{x}, \bar{x}] \subset \mathbb{R}^n$, $\underline{x} < \bar{x}$ *and, if*

$$\alpha_j = \frac{|\bar{x}_j| - |\underline{x}_j|}{\bar{x}_j - \underline{x}_j} \quad \text{and} \quad \beta_j = \frac{\bar{x}_j|\underline{x}_j| - \underline{x}_j|\bar{x}_j|}{\bar{x}_j - \underline{x}_j},$$

where x_j *denotes the jth component of* x, *we have:*

$$\forall x \in [\underline{x}, \bar{x}],\ \forall j,\ 1 \leqslant j \leqslant n, \quad |x_j| \leqslant \alpha_j x_j + \beta_j.$$

Proof. In fact, several situations may occur (see Figs. 1–3):

- *First case:* $\bar{x}_j \leqslant 0$, then $\alpha_j = -1, \beta_j = 0$, and $\alpha_j x + \beta_j = -x = |x|$.
- *Second case:* $\underline{x}_j \geqslant 0$, then $\alpha_j = 1, \beta_j = 0$, and $\alpha_j x + \beta_j = x = |x|$.
- *Third case:* $\bar{x}_j \underline{x}_j \leqslant 0$, then $\alpha_j = (\bar{x}_j + \underline{x}_j)/(\bar{x}_j - \underline{x}_j), \beta_j = (-2\bar{x}_j \underline{x}_j)/(\bar{x}_j - \underline{x}_j)$.

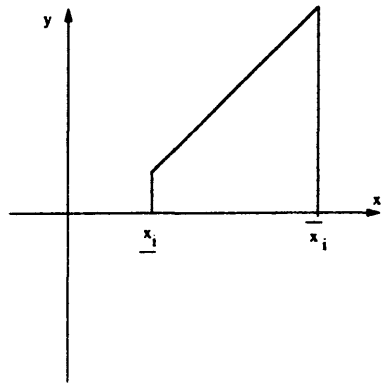


Fig. 1. Situation when both $\underline{x}_i$ and $\bar{x}_i$ are positive.

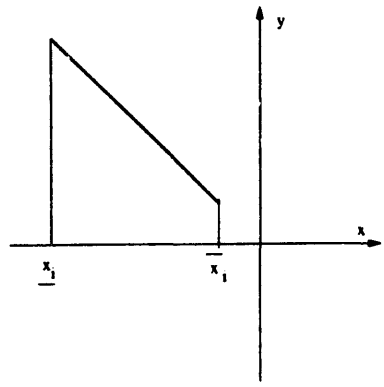


Fig. 2. Situation when both $\underline{x}_i$ and $\bar{x}_i$ are negative.

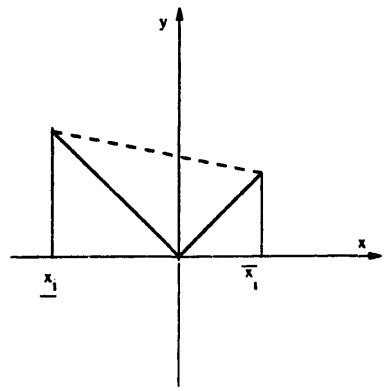


Fig. 3. Situation when $\underline{x}_i \leqslant 0$ and $\bar{x}_i \geqslant 0$.

- If $0 \leqslant z \leqslant \bar{x}_j$, then $\alpha_j z + \beta_j - |z| = -2\underline{x}_j/(\bar{x}_j - \underline{x}_j) * (\bar{x}_j - z) \geqslant 0$. Thus, $|z| \leqslant \alpha_j z + \beta_j$.
- If $\underline{x}_j \leqslant z \leqslant 0$, then $\alpha_j z + \beta_j - |z| = 2\bar{x}_j/(\bar{x}_j - \underline{x}_j) * (z - \underline{x}_j) \geqslant 0$. Thus, $|z| \leqslant \alpha_j z + \beta_j$. □

Lemma 2. *The convex hull of the set*

$$\mathscr{S} = \{(x,y) \in \mathbb{R}^2, \quad x \in [\underline{x}_j, \bar{x}_j], \ y \leqslant |x|\}$$

is the polyhedron defined by

$$\mathscr{P} = \{(x,y), x \in [\underline{x}_j, \bar{x}_j], 0 \leqslant y \leqslant \alpha_j x + \beta_j\}.$$

Proof. If $\underline{x}_j \geqslant 0$ or $\bar{x}_j \leqslant 0$, then $\mathscr{S} = \mathscr{P}$ and the property is trivial. We assume now that $\underline{x}_j < 0 < \bar{x}_j$. The set $\mathscr{P}$, which is defined by linear inequalities, is convex, and Lemma 2 implies $\mathscr{S} \subset \mathscr{P}$. Therefore, $\mathrm{Co}(\mathscr{S}) \subset \mathscr{P}$ (where $\mathrm{Co}(\mathscr{S})$ denotes the convex hull of $\mathscr{S}$).

Reciprocally, let us consider

$$(a,b) \in \{(x,y), x \in [\underline{x}_j, \bar{x}_j], y \leqslant \alpha_j x + \beta_j\} \setminus \mathscr{S}.$$

Let us consider now the line $y = \alpha_j x + (b - \alpha_j a)$, which intersects $y = x$ and $y = -x$ at the points $(a_1, b_1) = (b - \alpha_j a)/(1 - \alpha_j)(1, 1)$ and $(a_2, b_2) = (b - \alpha_j a)/(1 + \alpha_j)(-1, 1)$ respectively. Since $b - \alpha_j a \leqslant \beta$, we obtain $\underline{x}_j \leqslant a_1 \leqslant a_2 \leqslant \bar{x}_j$ and, therefore, $(a_1, b_1) \in \mathscr{S}$, $(a_2, b_2) \in \mathscr{S}$ and $(a,b) \in \mathrm{Co}(\mathscr{S})$. We have proved that $\mathrm{Co}(\mathscr{S}) = \mathscr{P}$. □

Let us assume that we know an enclosure of $\square\Sigma([A],[b])$, which may be obtained, for instance, by the algorithm proposed by Rump [7], $\square\Sigma([A],[b]) \subset [\underline{x}, \bar{x}]$. Therefore, we can expect that the following simplex $\Omega([A],[b],[\underline{x},\bar{x}])$ represents a good approximation of $\Gamma([A],[b])$. If we denote by D_x the diagonal matrix whose diagonal entries are the α_i's and β the vector of the β_i's obtained from $[\underline{x},\bar{x}]$, then we define $\Omega([A],[b],[\underline{x},\bar{x}])$ as follows:

$$\Omega([A],[b],[\underline{x},\bar{x}]) \begin{cases} A_c x - \Delta A \, D_x x \leqslant \bar{b} + \Delta A \beta, \\ A_c x + \Delta A \, D_x x \geqslant \underline{b} - \Delta A \beta. \end{cases}$$

$\Omega([A],[b],[\underline{x},\bar{x}])$ depends on the initial enclosure $[\underline{x},\bar{x}]$ because both D_x and β are defined under the condition $\square\Sigma([A],[b]) \subset [\underline{x},\bar{x}]$. We can now enunciate the main result of this paper.

Theorem 2. *Let* $\square\Sigma([A],[b]) \subset [\underline{x},\bar{x}]$. *Then, we have*

$$\Sigma([A],[b]) \subset \left\{ x ; \begin{pmatrix} A_c - \Delta A \, D_x \\ -A_c - \Delta A \, D_x \end{pmatrix} x \leqslant \begin{pmatrix} \bar{b} + \Delta A \beta \\ -\underline{b} + \Delta A \beta \end{pmatrix} \right\},$$

where

$$\alpha_j = \frac{|\bar{x}_j| - |\underline{x}_j|}{\bar{x}_j - \underline{x}_j} \quad \textit{and} \quad \beta_j = \frac{\bar{x}_j|\underline{x}_j| - \underline{x}_j|\bar{x}_j|}{\bar{x}_j - \underline{x}_j}$$

Proof. This theorem is a direct consequence of Theorem 1 and Lemma 1. □

We have seen above how an enclosure of $\square\Sigma([A],[b])$ leads to a simplex that describes a superset of $\Gamma([A],[b])$. The set of problems $\max_{x\in\Omega([A],[b],[\underline{x},\bar{x}])} x_i$ and $\min_{x\in\Omega([A],[b],[\underline{x},\bar{x}])} x_i$ can be solved, for instance, by applying $2n$ times the Simplex method [1]. Therefore, a new enclosure of $\square\Sigma([A],[b])$ is obtained, and an iterative scheme can be developed.

The limit of this iterative algorithm is usually a good enclosure of $\square\Sigma([A],[b])$ (not too large). Unfortunately, it is difficult to find a characterization of this limit in order to study its accuracy. The limit is usually not equal to $\square\Sigma([A],[b])$. We will display the results concerning the accuracy of the enclosure obtained in Section 4.

We may nevertheless conclude in several situations. For instance, when we know an enclosure of $\square\Sigma([A],[b])$ in which each x_i keeps a constant sign, it is easy to prove that the algorithm described above provides the exact solution $\square\Sigma([A],[b])$ after only one step. Indeed, in this case, the sets $\{(x,y),\ x\in[\underline{x}_j,\bar{x}_j],\ y\leqslant\alpha_j x+\beta_j\}$ and $\{(x,y),\ x\in[\underline{x}_j,\bar{x}_j],\ y\leqslant|x|\}$ are equal, and $\Omega([A],[b],[\underline{x},\bar{x}])$ is an exact representation of the solution set of $[A]x=[b]$, which is in this case convex.

We show on small examples given in [5] (matrices of size 2) the different situations that may occur:

If

$$[A] = \begin{pmatrix} [2,4] & [-1,1] \\ [-1,1] & [2,4] \end{pmatrix}, \qquad [b] = \begin{pmatrix} [-3,3] \\ 0 \end{pmatrix},$$

then

$$\square\Sigma([A],[b]) = \begin{pmatrix} [-2,2] \\ [-1,1] \end{pmatrix},$$

the enclosure obtained by the algorithm proposed by Rump [7] is

$$\square\Sigma([A],[b]) \subset \begin{pmatrix} [-2.11,2.11] \\ [-1.11,1.11] \end{pmatrix},$$

and, when starting our iterative algorithm from the enclosure obtained by the algorithm proposed by Rump, the limit we obtain is

$$\square\Sigma([A],[b]) \subset \begin{pmatrix} [-2,2] \\ [-1,1] \end{pmatrix}.$$

In this case, the algorithm we propose converges to $\square\Sigma([A],[b])$.

If

$$[A] = \begin{pmatrix} [2,4] & [-1,1] \\ [-1,1] & [2,4] \end{pmatrix}, \qquad [b] = \begin{pmatrix} [-0.5,6] \\ [1,1.5] \end{pmatrix},$$

then

$$\square\Sigma([A],[b]) = \begin{pmatrix} [-0.83,4.16] \\ [-1.16,2.83] \end{pmatrix},$$

the enclosure obtained by the algorithm proposed by Rump [7] is

$$\square\Sigma([A],[b]) \subset \begin{pmatrix} [-2.56,4.23] \\ [-2.06,2.89] \end{pmatrix},$$

and, when starting our iterative algorithm from the enclosure obtained by the algorithm proposed by Rump, the limit we obtain is

$$\square\Sigma([A],[b]) \subset \begin{pmatrix} [-1.50,4.16] \\ [-1.45,2.83] \end{pmatrix}.$$

In this case, the right bounds of the solution set we obtain are exact, but not the left ones. The algorithm converges to a strict superset of $\square\Sigma([A],[b])$.

If

$$[A] = \begin{pmatrix} 2 & [-1,0] \\ [-1,0] & 2 \end{pmatrix}, \qquad [b] = \begin{pmatrix} 1.2 \\ -1.2 \end{pmatrix},$$

then

$$\square\Sigma([A],[b]) = \begin{pmatrix} [0.3,0.6] \\ [-0.6,0.3] \end{pmatrix},$$

the enclosure obtained by the algorithm proposed by Rump [7] is

$$\square\Sigma([A],[b]) \subset \begin{pmatrix} [0.23,0.72] \\ [-0.72,-0.24] \end{pmatrix},$$

and, when starting our iterative algorithm from the enclosure obtained by the algorithm proposed by Rump, the result we obtain after one step is

$$\square\Sigma([A],[b]) \subset \begin{pmatrix} [0.3,0.6] \\ [-0.6,0.3] \end{pmatrix}.$$

In this case, the solution set we obtain is equal to $\square\Sigma([A],[b])$ after only one step of the algorithm.

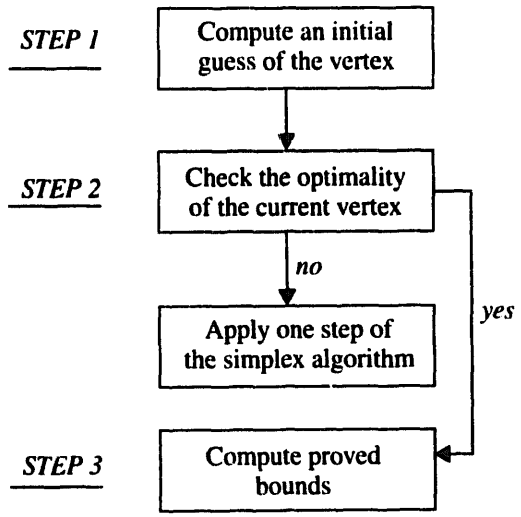
In Section 3, we present a modification of the algorithm presented above, in order to perform it in a reasonable amount of time.

3. Algorithm for the outer inclusion

In Section 2, we have presented an iterative method which solves interval linear systems. This method requires the execution of $2n$ Simplex algorithms during each step of the algorithm. Since an execution of the Simplex algorithm requires roughly $O(n^3)$ flops [1,11], the total amount of work per iteration is therefore of order $O(n^4)$. The algorithms proposed by Rump [7] and Neumaier [5] require an amount of work of order $5n^3$. We show in this section how to perform a step of the algorithm in time $O(n^3)$.

We propose a three-step algorithm in order to obtain an outer inclusion. The first step consists in using perturbation theory in order to find good starting points for the Simplex algorithm. The second step consists in checking the optimality of the starting points defined during step 1, and, if necessary in starting the Simplex algorithm. Indeed, as we will show in Section 4, the starting points obtained during step 1 are in many cases optimal especially when the perturbation matrix ΔA is small. Therefore, in many cases, the use of the Simplex algorithm is not necessary and the computation of the solution can be obtained in time of order $O(n^3)$. The last step is a correction step, which is necessary to obtain a proved outer inclusion, since all previous computations are not exact.

3.1. General sketch of the algorithm



3.2. First step

In order to obtain a cheap resolution of the problem, it is of importance to know good starting points for the Simplex algorithm. The first step of the algorithm consists therefore in using perturbation theory in order to find these

starting points. Let us suppose that x_0 is the solution of the system $A_c x_0 = b_c$, then $(x_0 + \delta x) \in \Sigma([A], [b])$ if and only if

$$\exists\, \delta A \in [-\Delta A, \Delta A],\ \exists\, \delta b \in [-\Delta b, \Delta b], \quad (A_c + \delta A)(x_0 + \delta x) = b_c + \delta b.$$

Therefore, at first order,

$$\delta x = A_c^{-1}(-\delta A x_0 + \delta b),$$

$$\forall i, 1 \leqslant i \leqslant n, \quad e_i^t\, \delta x = -e_i^t\, A_c^{-1}(\delta A x_0 + \delta b).$$

At this stage, we need to use the following basic lemma of interval arithmetic.

Lemma 3. *Let* $(x, y) \in \mathbb{R}^{n^2}$, $\Delta A \in M_n(\mathbb{R})$, $\Delta A \geqslant 0$, *then*

$$\{x^t \delta A y, \quad \delta A \in [-\Delta A, \Delta A]\} = [-|x^t|\Delta A|y|,\ |x^t|\Delta A|y|],$$

$$x^t \delta A y = -|x^t|\Delta A|y| \quad \text{if } \delta A = -D_1 \Delta A D_2,$$
$$x^t \delta A y = |x^t|\Delta A|y| \quad \text{if } \delta A = D_1 \Delta A D_2,$$

where D_1 *and* D_2 *denote the diagonal matrices whose entries are the sign vectors of* x *and* y *respectively.*

Therefore, when δA and δb describe respectively $[-\Delta A, \Delta A]$ and $[-\Delta b, \Delta b]$, $e_i^t \delta x$ describes, at first order,

$$[-|e_i^t A_c^{-1}|(\Delta A|x_0| + \Delta b),\ |e_i^t A_c^{-1}|(\Delta A|x_0| + \Delta b)].$$

Therefore, in order to approximately minimize $e_i^t \delta x$, a good choice consists in considering the system $A_c x - b_c = D_1(\Delta b + \Delta A D_2 x)$, where D_1 and D_2 are defined above.

Moreover, since $\Delta A D_2 x$ is supposed to be close to $\Delta A D_x x$, we can associate the solution of the previous equation with the vertex of $\Omega([A], [b], [\underline{x}, \bar{x}])$ defined by

$$A_c x - b_c = D_1(\Delta b + \Delta A D_x x).$$

Therefore, we can start the Simplex algorithm for minimizing $e_i^t \delta x$ from the vertex defined above. We will analyze the quality of this starting point in Section 4.

The total cost of this step consists in the inversion of A, i.e. n^3 flops when using an LU factorization [2].

3.3. Second step

This step consists in checking the optimality of the constraints set corresponding to a given point. In order to apply general theorems of linear pro-

gramming, we perform the following change of variables which induce use of positive variables. We set $y = x - \underline{x}$:

$$\Omega([A],[b],[\underline{x},\bar{x}]) = \left\{ y, \begin{pmatrix} A_c - \Delta A D_\alpha \\ -A_c - \Delta A D_\alpha \\ -Id_n \end{pmatrix} \right.$$
$$\left. y \leqslant \begin{pmatrix} \bar{b} + \Delta A\beta - (A_c - \Delta A D_\alpha)\underline{x} \\ -\underline{b} + \Delta A\beta + (A_c + \Delta A D_\alpha)\underline{x} \\ 0 \end{pmatrix} \right\}.$$

Therefore, a vertex y of $\Omega([A],[b],[\underline{x},\bar{x}])$ satisfies an equation of the following form

$$(DA_c - \Delta A D_\alpha)y = Db' + \Delta b', \tag{1}$$

where D is a diagonal matrix satisfying $|D| = Id_n$, $b'_c = b_c - A_c\underline{x}$ and $\Delta b' = \Delta b + \Delta A(D_\alpha \underline{x} + \beta)$.

3.3.1. Checking the optimality of the constraint set

Our aim is to check the optimality of the set of the constraints defined by Eq. (1) with respect to the minimization of y_1 over Ω. We use the following fundamental theorem of linear programming [1]:

Theorem 3. *Let u^t be the solution of*

$$u^t(DA_c - \Delta A D_\alpha) = e_1^t, \tag{2}$$

where e_1^t is the first canonical basis vector. If $u \geqslant 0$, then:

- *The set of constraints defined above is optimal for the problem* $\min_{y\in\Omega} y_1$.
- *The vertex y corresponding to the minimization satisfies $(DA_c - \Delta A D_\alpha)y = Db' + \Delta b'$.*
- $\min_{y\in\Omega} y_1 = e_1^t y = u^t(DA_c - \Delta A D_\alpha)y = u^t(Db' + \Delta A b')$.

3.3.2. Algorithm

We now present an algorithm to solve approximately Eq. (2). Let M be defined as $M = \Delta A D_\alpha B$, where B is the computed inverse of A_c. Note that M does not depend on the extremal point problem we consider. We use the following iterative scheme:

$$\delta_1^t = e_1^t B, \qquad \delta_{i+1}^t = (\delta_i^t D)M.$$

We stop the iteration scheme when $\delta_k \leqslant \epsilon_1$, a small threshold, and we set $\tilde{u}^t = (\sum_1^k \delta_i^t)D$.

We consider that the set of constraints is optimal if $\tilde{u} \leqslant \epsilon_2$, where ϵ_2 is a small positive vector. Therefore, if $\tilde{u} \leqslant \epsilon_2$, we go to the third step, otherwise, we perform a step of the Simplex algorithm and then we restart the second step. Note that if $\tilde{u}^t$

is the vector associated with the maximization of y_1, then $\tilde{u}'' = (\sum_1^k (-1)^{i+1}\delta_i^t)D$. Therefore, the computation of $\tilde{u}'$ only involves k additions of vectors.

3.4. Third step

3.4.1. Computing sure bounds

We are looking for a proved outer inclusion of the interval hull of the solution set. We therefore need to perform a correction step, since all previous computations were not sure. At this stage, we know that $\tilde{u} \leqslant \epsilon_2$.

Let us set

$$\hat{u} = \min(\tilde{u}, 0), \quad S = \begin{pmatrix} A_c - \Delta A D_\alpha \\ -A_c - \Delta A D_\alpha \end{pmatrix} \quad \text{and} \quad s = \begin{pmatrix} b_c' + \Delta b' \\ -b_c' + \Delta b' \end{pmatrix}.$$

If $\epsilon^t = \hat{u}^t S - e_1^t$, then we have

$$\begin{aligned} \hat{u}^t s &\leqslant \max\{z^t s,\ z \leqslant 0,\ z^t S \leqslant e_1^t + \epsilon^t\} \\ &\leqslant \min\{(e_1 + \epsilon)^t y, y \geqslant 0, Sy \leqslant s\} \text{ duality theorem of L.P.} \\ &\leqslant \min\{e_1^t y, y \geqslant 0, Sy \leqslant s\} + \max\{\epsilon^t y, y \geqslant 0, Sy \leqslant s\} \end{aligned}$$

and, therefore

$$\min_{y \in \Omega} y_1 \geqslant \hat{u}^t s - \max\{\epsilon^t y,\ 0 \leqslant y \leqslant \bar{x} - \underline{x}\} \geqslant \hat{u}^t s - ||\epsilon||\ ||\bar{x} - \underline{x}||.$$

The expression above gives a proved lower bound for $\min_{y \in \Omega} y_1$ and therefore for $\min_{y \in \Sigma} y_1$.

3.4.2. Practical implementation

In the sequel, we show how to bypass the computation of ϵ, by obtaining an upper bound of $||\epsilon||$. Let A' denote the exact inverse of the computed inverse B of A (whenever A' does not exist, we cannot apply what follows). Let us set

$$\Delta = A' - A_c, \qquad C = DA_c - \Delta A D_\alpha,$$

$$\mu = (\tilde{u} - \hat{u})^t \qquad \tilde{u}^t D = e_1^t B \sum_0^{k-1} (DM)^i,$$

$$u^t D = e_1^t B \sum_0^{\infty} (DM)^i \iff u^t (C - D\Delta) = e_1^t,$$

$$v^t D = e_1^t A_c \sum_0^{\infty} (DM)^i \iff u^t C = e_1^t.$$

Theorem 4. *If* $||M|| \leqslant \frac{1}{2}$, *and* $\epsilon' ||A_c||\ ||B|| \leqslant \frac{1}{2}$, *where* ϵ', *defined below, only depends on the machine precision, then*

$$||\epsilon|| \leqslant (||A_c|| + ||\Delta A||)(||\epsilon_1|| + ||\mu||) + 4\epsilon'||A_c||^2||B||(||\tilde{u}|| + ||\epsilon_1||).$$

Proof. Evaluation of $||(\tilde{u}-u)^tC||$: $(\tilde{u}-u)^tD = e_1^tB\sum_k^\infty (DM)^i = \delta_k DM\sum_0^\infty (DM)^i$. Since $||M|| \leqslant \frac{1}{2}$,

$$||(\tilde{u}-u)^tC|| \leqslant ||\epsilon_1||(||A_c|| + ||\Delta A||) \quad \text{and} \quad ||(\tilde{u}-u)^t|| \leqslant ||\epsilon_1||.$$

Evaluation of $||(v-u)^tC||$: $(v-u)^tC = u^tD\Delta$ and therefore $||(v-u)^tC|| \leqslant ||\Delta||(||\tilde{u}|| + ||\epsilon_1||)$.

Evaluation of $||(v-\tilde{u})^tC||$: $||(v-\tilde{u})^tC|| \leqslant ||(v-u)^tC|| + ||(\tilde{u}-u)^tC||$ and therefore

$$||(v-\tilde{u})^tC|| \leqslant ||\epsilon_1||(||A_c|| + ||\Delta A||) + ||\Delta||(||\tilde{u}|| + ||\epsilon_1||).$$

Evaluation of $||\Delta||$: If the computed inverse B of A_c is obtained with LU factorization, we know [4] that $A_cB = I + E$ where $||E|| \leqslant \epsilon'||A_c||||B||$, where ϵ' depends on machine accuracy. Thus, $A' = (I+E)^{-1}A_c$ and, since $||E|| \leqslant \epsilon'||A_c||||B|| \leqslant \frac{1}{2}$, $A' - A_c = A_cE\sum_0^\infty (-1)^{k+1}E^k$ and finally

$$||\Delta|| \leqslant 2\epsilon'||A_c||^2\,||B||.$$

Since $\epsilon = \tilde{u}^tC - e_1^t = (\tilde{u}-v)^tC + (v^tC - e_1^t) - \mu C$, we obtain the proof of the theorem. □

The crucial point is that the majoration of $||\epsilon||$ does not require additional computations.

4. Numerical results

In this section, we describe numerical results obtained with random matrices A_c and ΔA and for random vectors b_c and Δb. We compare the results of the proposed algorithm after one iteration with the results of the algorithm proposed by Rump [7]. The initial enclosure we consider is the result of Rump's algorithm. It is known [8] that, for Rump's algorithm, the ratio perimeter of Rump's outer inclusion/perimeter of the interval hull depends on $\text{norm}(A)/\text{norm}(\Delta A)\,\text{cond}(A)$.

We therefore display results according to $\text{norm}(A)/\text{norm}(\Delta A)\,\text{cond}(A)$. As the quality of the enclosure of Σ in Ω depends on the position of the solution set with respect to 0, we consider different situations for Σ (that is to say containing 0 or not intersecting any axis). The x-axis represents the size of all the test matrices (Fig. 4).

In Figs. 5–7 we display τ_1, the average number of the steps of the Simplex algorithm necessary for the computation of an extremal point from the initial guess (step 2) and

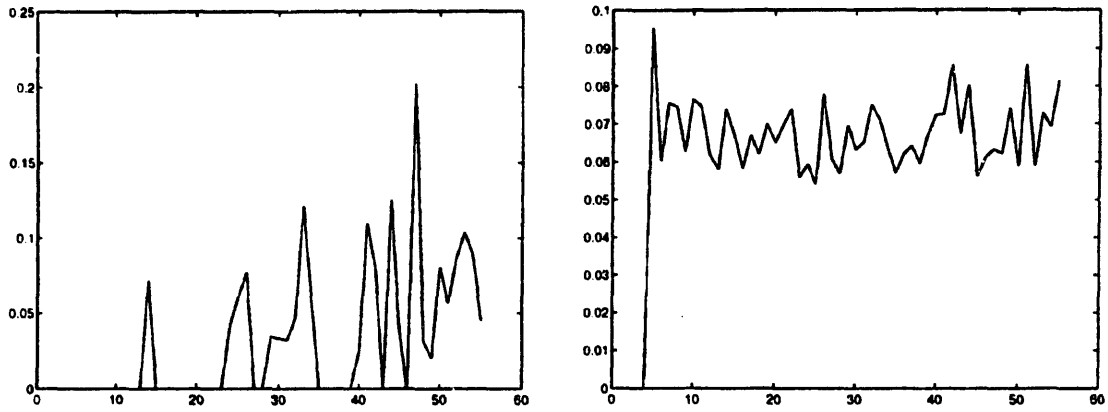


Fig. 4. τ_1 and τ_2 Σ not centered – norm(ΔA) = (norm$(A_c) * 0.1$)/cond(A_c).

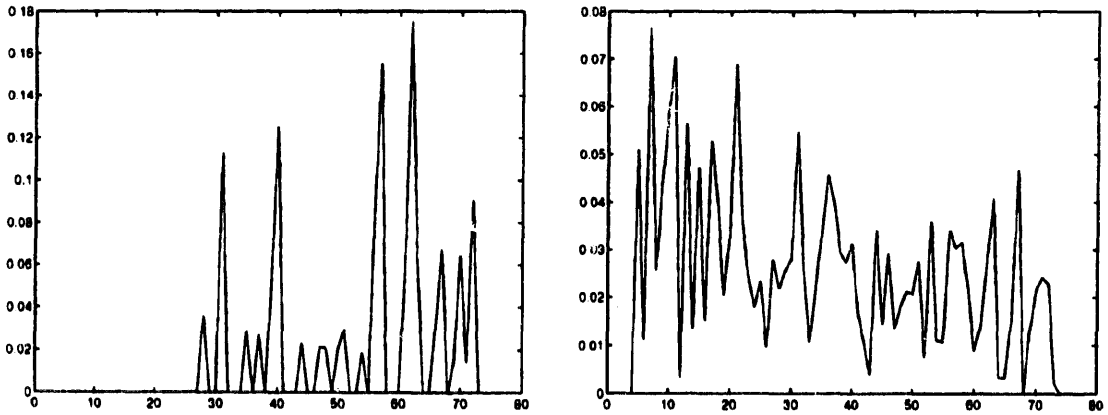


Fig. 5. τ_1 and τ_2, Σ centered – norm(ΔA) = (norm$(A_c) * 0.1$)/cond(A_c).

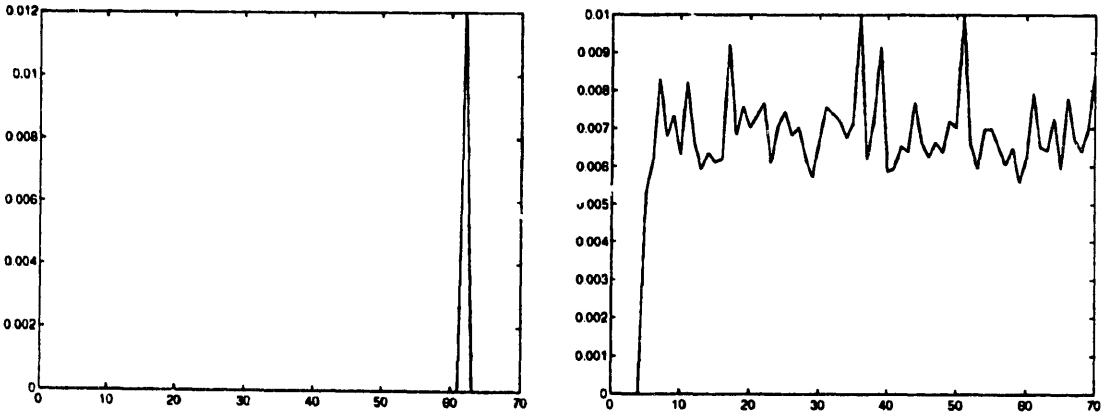


Fig. 6. τ_1 and τ_2, Σ not centered – norm(ΔA) = (norm$(A_c) * 0.1$)/cond(A_c).

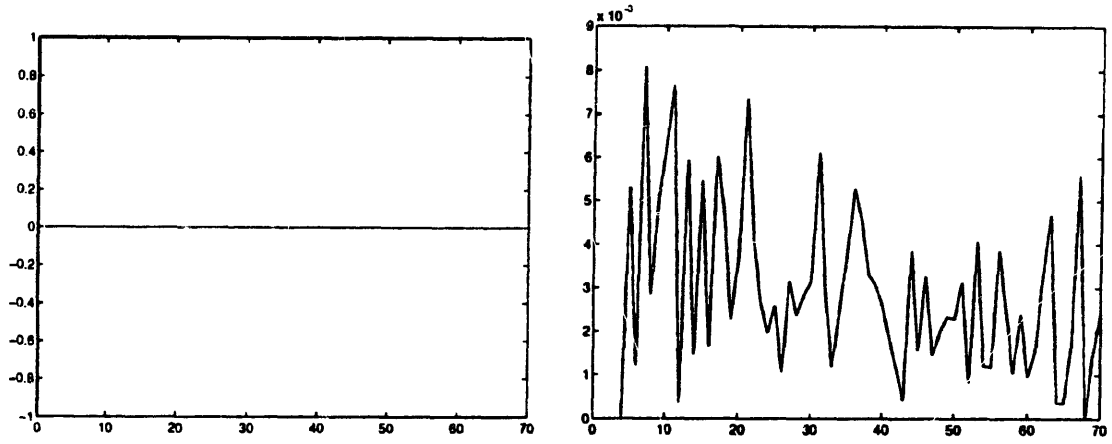


Fig. 7. τ_1 and τ_2, Σ centered – norm(ΔA) = (norm(A_c) * 0.01)/cond(A_c).

$$\tau_2 = 1 - \frac{\text{perimeter(Simplex method)}}{\text{perimeter(Rump's algorithm)}}.$$

5. Algorithm for the inner inclusion

We now present an algorithm which computes an inner inclusion of the solution set. By inner inclusion, we mean an interval vector $[\underline{y}, \bar{y}]$ such that

$$[\underline{y}, \bar{y}] \subset \square\Sigma([A], [b]) \subset \Omega([A], [b], [\underline{x}, \bar{x}]).$$

Note that it is a completely different problem to solve $[\underline{y}, \bar{y}] \subset \Sigma([A], [b])$. The main interest of the inner inclusion we compute is to estimate the accuracy of the outer inclusion and to enclose the interval hull of the solution set between two interval vectors. Moreover, numerical results indicate that the inner inclusion is very close to the interval hull (in fact, it is very often the exact interval hull).

The algorithm is based on the results for the outer inclusion. It computes $2n$ points of the solution set which are supposed to be close to extremal points.

5.1. How to find inner "extremal" points?

Let us consider again the problem of the minimization of y_1. We know that the extremal point of Ω which realizes this minimization is defined by

$$D(A_c x - b_c) = \Delta A(D_x x + \beta) + \Delta b. \tag{3}$$

Let us now consider a point on the frontier of Σ.

Lemma 4 (Rohn). *x belongs to the frontier of Σ $\iff$ $|A_c x - b_c| = \Delta A|x| + \Delta b$ and*

$$|A_c x - b_c| = \Delta A|x| + \Delta b \iff$$

$$\exists D' \text{ diagonal}, \ |D'| = Id(n), \ D'(A_c x - b_c) = \Delta A|x| + \Delta b. \tag{4}$$

This lemma is a direct application of the Oettli–Prager [8] theorem. Since $D_\alpha x + \beta$ represents an approximation of $|x|$, we can notice an analogy between the definition of x in expressions (3) and (4).

We therefore consider as extremal point associated to the minimization of x_1 for the inner inclusion the point x defined by

$$D(A_c x - b_c) = \Delta A|x| + \Delta b,$$

where D is the matrix associated to the minimization of y_1 over Ω.

5.2. Algorithm

The algorithm consists in solving the equations

$$D(A_c x - b_c) = \Delta A|x| + \Delta b \iff x = M|x| + a,$$

where $M = A_c^{-1} D \Delta A$ and $a = A_c^{-1}(D\Delta b + b)$.

Since such a point belongs to the frontier of Σ, the algorithm leads to an inner inclusion of Σ. If we suppose that $[A]$ is strongly regular, that is to say $\rho(|A_c^{-1}|\Delta A) < 1$, then $\rho(M) < 1$ and we can use the algorithm proposed by Rohn.

Theorem 5. *If $\rho(M) < 1$, then, for every a, the equation $x = M|x| + a$ has a unique solution, and the iteration*

$$x^{l+1} := M|x^l| + a (l = 0, 1, 2, \ldots)$$

converges to the solution for every choice of the starting vector x^0.

In order to obtain a good starting vector, we can solve the equation

$$D(A_c x - b_c) = \Delta A(D_\alpha x + \beta) + \Delta b.$$

We therefore start from the vector which corresponds to the minimization of x_1 over Ω. When $\rho(M)$ is not small enough, it is known that the convergence may be slow. Rohn therefore proposed the sign accord algorithm, which does not require strong regularity of $[A]$.

It consists in solving $x = MD'x + a$, for different matrices D'.

Sign accord algorithm (Rohn)

Step 1: Select D' with $|D'| = Id(n)$

Step 2: For $s = 1, \ldots, 2^n$ do
solve $x = MD'x + a$;
if $D'x \geqslant 0$ terminate (success); otherwise compute
$k := \min\{j = 1, \ldots, n; D'_{jj} x_j < 0\}$;
change the sign of D'_{kk};

Step 3: Terminate (failure).

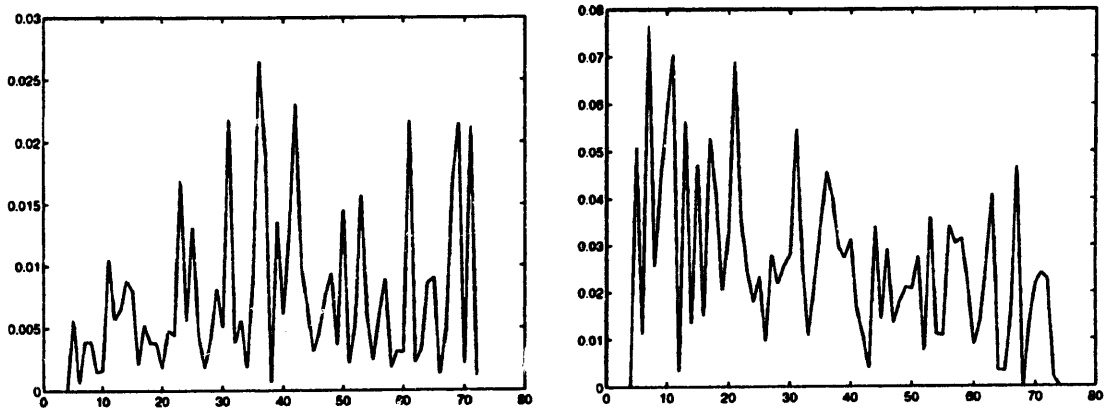


Fig. 8. τ_1 and τ_2, Σ centered – norm(ΔA) = (norm(A_c) * 0.1)/cond(A_c).

Rohn [10] has proved that the algorithm is finite when $[A]$ is regular. Although the number of steps may be exponential, it is generally reasonable. If we know the signs of the solution of $D(A_c x - b_c) = \Delta A(D_x x + \beta) + \Delta b$, then we can start with the matrix D' such that $D'x \geqslant 0$. In the cases such that the points that realize the minimization of x_1 over Σ and Ω have the same sign vector, we will not have to perform any change of sign. In fact, none of the cases we considered for the outer inclusion requires the execution of more than one step.

5.3. Numerical results

We display results for matrices of the same kind as those used for the outer inclusion. We only consider the case Σ centered since results for inner and outer inclusion are the same when Σ does not intersect any axis. Figs. 8 and 9 represent the evolution with n of both quantity τ_1 and τ_2, where

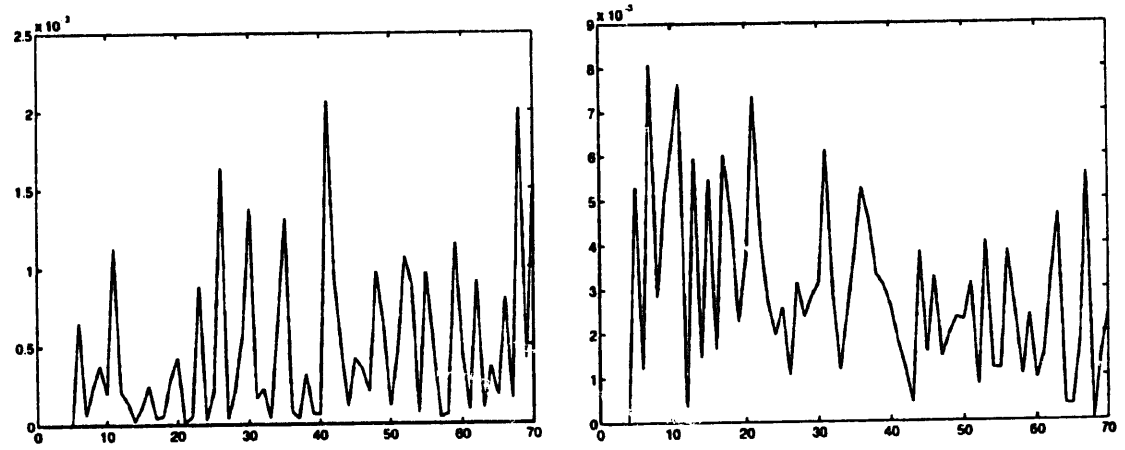


Fig. 9. τ_2 and τ_2, Σ centered – norm(ΔA) = (norm(A_c) * 0.01)/cond(A_c).

$$\tau_1 = 1 - \frac{\text{perimeter(inner inclusion)}}{\text{perimeter(outer inclusion)}}$$

$$\tau_2 = 1 - \frac{\text{perimeter(outer inclusion)}}{\text{perimeter(Rump's algorithm)}}.$$

We can see that inner and outer inclusions are usually very close. Even in the case where $\text{norm}(\Delta A) = (\text{norm}(A_c) * 0.1)/\text{cond}(A_c)$, we usually obtain

$$1 - \frac{\text{perimeter}(\square\Sigma)}{\text{perimeter(outer inclusion)}} \leqslant 1\%.$$

6. Conclusion

In this paper, we show how to derive from the Simplex algorithm an efficient method to compute inner and outer inclusion of the united solution set. The outer inclusion is obtained in $O(n^3)$ flops and does not require the preconditioning of the system. The inner inclusion coincides very often with the exact interval hull of the united solution set, but we have no way to prove this property. However, an estimation of the quality of the outer inclusion can be obtained, even without computing the inner inclusion.

Acknowledgements

The author wishes to thank Bernard Philippe and Jiri Rohn for careful reading and for for their proposals, which greatly improved the clarity of this paper.

References

[1] A. Chvatal, Linear Programming, Freeman, New York, 1983.
[2] P. Gill, W. Murray, M. Wright, Numerical Linear Algebra and Optimization, Addison-Wesley, Reading, MA, 1991.
[3] V. Kreinovich, J. Rohn, Computing exact componentwise bounds on solutions of linear systems with interval data is NP-Hard, SIAM J. Matrix Anal. Appl. 16 (2) (1995) 415–420.
[4] P. Lascaux, R Theodor, Analyse Numérique Matricielle Appliquée à l'Art de l'Ingénieur, Masson.
[5] A. Neumaier, Interval methods for systems of equations, Cambridge University Press, Cambridge, 1990.
[6] W. Oettli, W. Prager, Computability of approximate solution of linear equations with given error bounds for coefficients and right-hand sides, Numer. Math. 6 (1964) 405–409.
[7] S.M. Rump, On the solution of interval linear systems, Computing 47 (1992) 337–353.

[8] S.M. Rump, Verification methods for dense and sparse systems of equations, in: J. Herzberger (Ed.), Topics in Validated computations, Elsevier, Amsterdam, 1994.
[9] J. Rohn, NP-hardness results for linear algebraic problems with interval data, in: J. Herzberger (Ed.), Topics in Validated computations, Elsevier, Amsterdam, 1994.
[10] J. Rohn, Checking bounds on solutions of linear interval equation is NP-hard, Linear Algebra Appl. 223–224 (1995) 589–596.
[11] A. Schrijver, Theory of Linear and Integer Programming, Wiley, New York, 1986.

Appendix D

Algèbre linéaire avec des processeurs hétérogènes

Les trois articles suivants (publiés respectivement dans IEEE TPDS, IEEE TC et Algorithmica) considèrent le problème de la distribution des données pour le produit de matrices quand les ressources de calcul (mais pas les ressources de communication) sont hétérogènes. Comme nous l'avons évoqué au Paragraphe B.2.2, le problème revient à partitionner un carré unité en rectangles de surfaces prédéfinies.

L'article publié dans IEEE TPDS s'intéresse au cas où le réseau impose de séquentialiser toutes les communications. On se ramène alors à un problème de partitionnement dans lequel l'objectif est de minimiser la somme des périmètres des rectangles.

Au contraire, celui publié dans Algorithmica considère le problème avec un réseau qui permet de faire un nombre quelconque de communications en parallèle. On se ramène alors à un problème de partitionnement dans lequel l'objectif est de minimiser le périmètre maximal des rectangles.

Enfin, l'article publié dans IEEE TC considère une distribution particulière, plus proche de celle utilisée dans le cas homogène, dans laquelle on impose un découpage du carré unité sous la forme d'une grille. On ne peut alors plus obtenir d'équilibrage parfait de la charge, mais le schéma de communication induit par la distribution est alors beaucoup plus simple.

Matrix Multiplication on Heterogeneous Platforms

Olivier Beaumont, Vincent Boudet,
Fabrice Rastello, and Yves Robert, *Member*, *IEEE*

Abstract—In this paper, we address the issue of implementing matrix multiplication on heterogeneous platforms. We target two different classes of heterogeneous computing resources: heterogeneous networks of workstations and collections of heterogeneous clusters. Intuitively, the problem is to load balance the work with different speed resources while minimizing the communication volume. We formally state this problem in a geometric framework and prove its NP-completeness. Next, we introduce a (polynomial) column-based heuristic, which turns out to be very satisfactory: We derive a theoretical performance guarantee for the heuristic and we assess its practical usefulness through MPI experiments.

Index Terms—Parallel algorithms, load balancing, communication volume, matrix multiplication, numerical linear algebra libraries, heterogeneous platforms, cluster computing, metacomputing.

1 Introduction

In this paper, we deal with the implementation of a very simple but important linear algebra kernel, namely, matrix multiplication (MM for short), on heterogeneous platforms. Several parallel MM algorithms are available for parallel machines or homogeneous networks of workstations or PCs (see [1], [20], [31], among others). The popular ScaLAPACK library [7] includes a highly-tuned, very efficient routine targeted to two-dimensional processor grids. This routine uses a block-cyclic distribution of the matrices in both grid dimensions. We briefly recall parallel MM algorithms for homogeneous machines in Section 2.1.

Why extend parallel MM algorithms to heterogeneous platforms? The answer is clear: Future computing platforms are best described by the keywords *distributed* and *heterogeneous*. We target two different classes of heterogeneous computing resources:

eterogeneous Networks of Workstations (NOWs) are ubiquitous in university departments and companies. They represent the typical poor man's parallel computer: Running a large PVM or MPI experiment (possibly all night long) is a cheap alternative to buying supercomputer hours. When implementing MM algorithms on HNOWs, the idea is to make use of *all* available resources, namely, slower machines *in addition to* more recent ones. This is a challenging but very useful task given the importance of MM in scientific computing. Also, it is a first step towards understanding how to implement more complicated linear algebra kernels on HNOWs.

Collections of Clusters are made up of nodes or clusters, each of them being itself a HNOW of a parallel machine. These nodes may well be geographically scattered all around the world. Internode communications are typically an order of magnitude slower than intranode communications. The need to design a MM algorithm which would execute on a collection of clusters is less obvious. Are there actual applications which involve huge matrices whose product cannot be computed with a single parallel machine or workstation network? Larger and larger experiments are conducted throughout the world within the NPACI[1] initiative using tools such as Globus [18] and Legion [24]. Huge linear algebra kernels are often at the core of these experiments, so investigating "metacomputing" MM algorithms is quite natural. Anyway, we view MM algorithms as a perfect case study for the implementation of tightly coupled high-performance applications on the metacomputing grid [19]: Indeed, such applications are much more difficult to tackle than loosely-coupled cooperative applications. Because MM is a simple kernel which encompasses a lot of data movements, we view it as a perfect testbed to be studied before experimenting more challenging computational problems on the grid, especially those which exhibit a high spatial locality (e.g., such as finite difference schemes).

The major limitation to programming heterogeneous platforms arises from the additional difficulty of balancing the load when using processors running at different speeds. Data and computations are not evenly distributed to processors. Minimizing communication overhead becomes a challenging task: In fact, the MM problem with different-speed processors turns out to be surprisingly difficult. The main result of this paper is the NP-completeness of the MM problem on heterogeneous platforms. Rather than the

• *The authors are with the Laboratoire de l'Informatique du Parallelisme, UMR CNRS-ENS Lyon-INRIA 5668, École Normale Supérieure de Lyon, F-69364 Lyon Cedex 07, France. E-mail: {Yves.Robert, Olivier.Beaumont, Vincent.Boudet, Fabrice.Rastello}@ens-lyon.fr.*

Manuscript received 24 Jan. 2000; revised 1 Aug. 2000; accepted 1 May 2001. For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 111321.

1. National Partnership for Advanced Computational Infrastructure, see http://www.npaci.edu.

proof, the result itself is interesting because it reveals the intrinsic difficulty of designing heterogeneous algorithms.

The rest of the paper is organized as follows: In Section 2, we summarize existing MM algorithms for homogeneous platforms and we discuss how to extend these to cope with heterogeneity. In Section 3, we formally state the MM optimization problem for heterogeneous platforms, which we formulate as a geometric optimization problem, and we establish its NP-completeness (the long and technical proof of this important result is given in the Appendix). In Section 4, we briefly survey related NP-complete optimization problems. Section 5 is devoted to the design of efficient (polynomial) heuristics, whose practical usefulness is demonstrated through MPI experiments on a HNOW and on a 2-cluster configuration (Section 6). We give some final remarks and conclusions in Section 7.

2 MM Algorithms

In this section, we briefly describe how to implement a parallel (or distributed) MM algorithm on a heterogeneous platform. We adopt an abstract view by assuming that we have a collection of p heterogeneous computing resources $P_1, P_2, \ldots, P_p$. If each computing resource P_i reduces to a single processor, we are dealing with a heterogeneous network of workstations or PCs (HNOW). When each computing resource P_i is itself a heterogeneous cluster or a parallel machine, we are targeting a metacomputing environment made up from a collection of clusters. The high-level algorithmic description is the same for all target machines. However, our model will have to cope with different hypotheses on communication issues. We come back to the impact of communication modeling in Section 3.2. Before dealing with heterogeneous resources, we briefly summarize existing algorithms for homogeneous machines.

2.1 Homogeneous Grids

We start by briefly recalling the MM algorithm implemented in the ScaLAPACK library [7] on 2D homogeneous grids. For the sake of simplicity, we restrict ourselves to the multiplication $C = AB$ of two square $n \times n$ matrices A and B. In that case, ScaLAPACK uses the outer product algorithm described in [1], [20], [31]. Consider a 2D processor grid of size $p = p_1 \times p_2$ and assume for a while that $n = p_1 = p_2$. In that case, the three matrices share the same layout over the 2D grid: Processor $P_{i,j}$ stores $a_{i,j}$, $b_{i,j}$, and $c_{i,j}$. Then, at each step k,

- Each processor $P_{i,k}$ (for all $i \in \{1, .., p_1\}$) horizontally broadcasts $a_{i,k}$ to processors $P_{i,*}$ and
- Each processor $P_{k,j}$ (for all $j \in \{1, .., p_2\}$) vertically broadcasts $b_{k,j}$ to processors $P_{*,j}$,

so that each processor $P_{i,j}$ can independently update $c_{i,j} = c_{i,j} + a_{i,k}b_{k,j}$.

This current version of the ScaLAPACK library uses a blocked version of this algorithm to squeeze the most out state-of-the-art processors with pipelined arithmetic units and multilevel memory hierarchy [17], [12]. Each matrix coefficient in the description above is replaced by a $r \times r$ square block, where optimal values of r depend on the memory hierarchy and on the communication-to-computation ratio of the target computer. Finally, a level of virtualization is added: Usually, the number of blocks $\lceil \frac{n}{r} \rceil \times \lceil \frac{n}{r} \rceil$ is much greater than the number of processors $p_1 p_2$. Thus, blocks are scattered in a cyclic fashion along both grid dimensions, so that each processor is responsible for updating several blocks at each step of the algorithm.

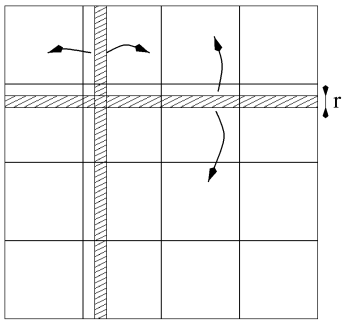


Fig. 1. One step of the MM algorithm on a 4×4 homogeneous 2D-grid.

To prepare for the description of the heterogeneous version, we introduce another "logical" description of the algorithm:

- We take a macroscopic view and concentrate on allocating (and operating on) matrix blocks to processors: Each element in A, B, and C is a square $r \times r$ block and the unit of computation is the updating of one block, i.e., a matrix multiplication of size r.
- At each step, a column of blocks (the pivot column) is communicated (broadcast) horizontally and a row of blocks (the pivot row) is communicated (broadcast) vertically.
- The C matrix is partitioned into $p_1 \times p_2$ rectangles. There is a one-to-one mapping between these rectangles and the processors. Each processor is responsible for updating its rectangle: More precisely, it updates each block in its rectangle with one block from the pivot row and one block form the pivot column, as illustrated in Fig. 1. For square $p \times p$ homogeneous 2D-grids and when the number of blocks in each dimension n is a multiple of p (the actual matrix size is thus $n.r \times n.r$), it turns out that all rectangles are identical squares of $\frac{n}{p} \times \frac{n}{p}$ blocks.

In Fig. 1, we see that the total amount of communications performed by the MM algorithm is proportional to the sum of the half-perimeters of the rectangles allocated to the processors: More precisely, at each step each processor responsible for a rectangle of $h \times v$ blocks must receive (vertically) h blocks of matrix B and (horizontally) v blocks of matrix A. This explains why allocated rectangles are identical squares for square $p \times p$ homogeneous 2D-grids when p divides n: In that case, all rectangles of fixed area $\frac{n}{p} \times \frac{n}{p}$ are squares. Because the (half)-perimeter of a rectangle of fixed area is minimized when it is a square, this choice does minimize the communication volume.

There are other homogeneous MM algorithms: For instance, Cannon's algorithm [31] (whose main drawback

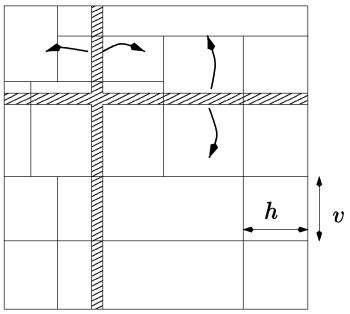


Fig. 2. The MM algorithm on a heterogeneous platform.

is to require an initial permutation of matrices A and B) replaces all the horizontal and vertical broadcasts by nearest-neighbor shifts. The total communication volume at each step is the same, but the communications are different. Still, all processors independently update their rectangle of C blocks at each step.

2.2 Heterogeneous Platforms

How to modify the previous MM algorithms for a heterogeneous platform? The idea is to keep the same framework: at each step, one pivot column and one pivot row are communicated to all processors and independent updates take place. However, with different speed processors, we cannot distribute same size rectangles from the C matrix to the processors. Intuitively, we want to balance the computing load so that each processor receives an amount of work in accordance to its computing power. Because all C blocks require the same amount of arithmetic operations, each processor executes an amount of work which is proportional to the number of blocks that are allocated to it, hence, proportional to the area of its rectangle. To parallelize the matrix product $C = AB$, we have to tile the C matrix into p nonoverlapping rectangles, each rectangle being assigned to one processor. Fig. 2 shows an example with 13 different-speed computing resources.

The question is: How to compute the *area* and *shape* of these p rectangles so as to minimize the total execution time? As usual, with parallel algorithms, there are two nonindependent and maybe conflicting goals: 1) load-balancing computations and 2) minimizing communication overhead. Goal 1) is related to the area of the rectangles that are allocated to the processors while goal 2) is related to their shapes. We discuss areas and shapes in the next section, in order to formally state (and try to solve) this difficult optimization problem.

3 The Heterogeneous MM Optimi ation Problem

Consider a matrix product $C = AB$, where A, B, and C are square matrices of $n \times n$ square blocks of size r. Assume that we have p computing resources $P_1, P_2, \ldots, P_p$ of (relative) cycle-times $t_1, t_2, \ldots, t_p$: If all processors have same speed, then $t_i = 1$ for $1 \leq i \leq p$. If, say, P_2 is twice faster than P_1, then $t_1 = 2t_2$. We start with load-balancing issues before dealing with communication overhead.

3.1 Load Balancing

To perfectly load-balance the computation, each processor should receive an amount of work in accordance to its computing power. If, say, P_2 is twice faster than P_1 ($t_1 = 2t_2$), then P_2 should be assigned twice as many elements as P_1. In other words, the *area* of its rectangle should be the double of that of P_1. Let s_i be the area of the rectangle R_i allocated to processor P_i. Obviously, the first equation is $\sum_{i=1}^{p} s_i = n^2$, in order to obtain a true partition of the C matrix. Next, since P_i processes its rectangle within $s_i t_i$ time-steps, we have

$$s_1 t_1 = s_2 t_2 = \ldots = s_p t_p.$$

The last constraint is to write s_i as $s_i = h_i v_i$, where h_i and v_i are the number of rows and columns of R_i. These equations do not always have integer solutions, which means that a perfect load balancing of the computations is not always possible.

However, we are not really interested in an exact solution. A more concrete and interesting question is the following: Given the p computing resources, how to compute the respective area of the rectangles R_i so that the workload is asymptotically optimally balanced: the larger the matrix size (expressed in blocks), the more accurate the tiling into rectangles. This question translates into the following system: Given $t_1, \ldots, t_p$, search for real unknowns s_i, h_i, and v_i, $1 \leq i \leq p$, such that:

$$\begin{cases} 1) & s_1 t_1 = s_2 t_2 = \ldots = s_p t_p \\ 2) & \sum_{i=1}^{p} s_i = 1 \\ 3) & \text{The } p \text{ rectangles of size } h_i \times v_i \text{ (where } h_i v_i = s_i) \\ & \qquad\qquad\qquad\qquad \text{tile the unit square.} \end{cases}$$

Condition 1 ensures that the area of the rectangle R_i allocated to processor P_i is inversely proportional to its cycle time. Condition 2 is for normalization: The sum of the areas of the p rectangles is that of the unit square, a necessary condition for Condition 3 to hold. Note that, as expected, Conditions 1 and 2 allow to compute the s_i: We obtain

$$s_i = \frac{\frac{1}{t_i}}{\sum_{i=1}^{p} \frac{1}{t_i}}.$$

We see that s_i is computed from the harmonic mean of the t_i and it is not an integer ($0 < s_i < 1$ as soon as $p \geq 2$).

There are always solutions to the normalized problem. For instance, we fulfill Condition 3 by choosing to tile the unit square into p horizontal slices of height $v_i = s_i$ (and width $h_i = 1$) or into p vertical slices of width $h_i = s_i$ (and height $v_i = 1$). This degree of freedom comes from the fact that load balancing imposes constraints on the area of the rectangles R_i, but not on their shapes. Shapes come into the story when discussing communication issues, as explained below.

Finally, note that it is straightforward to retrieve a solution of the original problem (tiling a matrix of $n \times n$ blocks) from the solution of the normalized problem: We simply multiply all the h_i and the v_i by n, getting $h_i(n) = nh_i$ and $v_i(n) = nv_i$. Then, we round up values to integers, $h'_i(n)$ and $v'_i(n)$, while preserving the

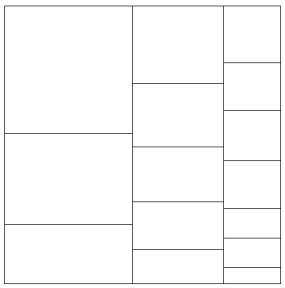


Fig. 3. Tiling the unit square into columns of rectangles.

constraint $\sum_{i=1}^{p} h_i'(n) = \sum_{i=1}^{p} v_i'(n) = n$ (there are many possible variations). Therefore, we derive a *generic* solution to the original problem, which is valid for all values of the parameter n.

3.2 Communication Overhead

At each step of the MM algorithm, communications take place between processors: The total volume of data exchanged is proportional to the sum $\hat{C} = \sum_{i=1}^{p}(h_i + v_i)$ of the half-perimeters of the p rectangles R_i. In fact, this is not exactly true: Because the pivot row and column are not sent to the processors that own them, we should subtract 2 from $\hat{C}$, 1 for the horizontal communications and 1 for the vertical ones. Since minimizing $\hat{C}$ or $\hat{C} - 2$ is equivalent, so we keep the value of $\hat{C}$ as stated.

Se uential Communications. Minimizing $\hat{C}$ seems to be a very natural goal because it represents the total volume of communications. For instance, it is natural to assume that communications will be mostly sequential on a HNOW where processors are linked by a simple Ethernet network; also, there will be little or none computation/communication overlap on such a platform. In that context, minimizing the total communication volume is the main objective: It is proportional to the communication time needed at each step of the MM algorithm with the underlying hypothesis that the network is homogeneous. In this paper, we do not investigate further the situation where different speed links are available between the processors (see Section 3.5 for a pointer).

Parallel Communications. Conversely, some communications can occur in parallel or some efficient broadcast mechanisms can be used if the computing resources are linked through a dedicated high-speed network and if parallel communication links are provided. In that context, we may want to use a columnwise allocation as depicted in Fig. 3: Vertical communications are performed in parallel in all columns and broadcasts or at least scatters can be performed horizontally.

Collections of Clusters. Finally, in a metacomputing context, intercluster communications are typically one order of magnitude slower than intracluster communications, so we may want to adopt a two-level scheme: We assign rectangles to clusters as described in Fig. 4 while inside each cluster some master-slave mechanism could be provided.

Optimi ation Criteria. It seems that minimizing the total communication volume is the most important optimization problem because of its wide potential applicability. Also, forgetting about MM algorithms for a while, consider the implementation of any application (such as a finite-difference scheme), where heterogeneous processors communicate boundary elements at each step (the communication scheme need not be nearest-neighbor, it can be anything): Minimizing the total communication volume while load-balancing the work amounts to solving exactly the same optimization problem.

The rest of the paper is devoted to solving the MM optimization problem using the total communication volume $\hat{C}$ as the objective function to be minimized. We formally state this optimization problem in Section 3.3. For the sake of completeness, we discuss some extensions of the problem in Section 3.5.

3.3 The MM Optimi ation Problem

We are ready to state the MM optimization problem for heterogeneous platforms. We have p computing resources P_i, $1 \le i \le p$. Each P_i is assigned a rectangle R_i of prescribed area s_i, where $\sum_{i=1}^{p} s_i = 1$. The shape of each

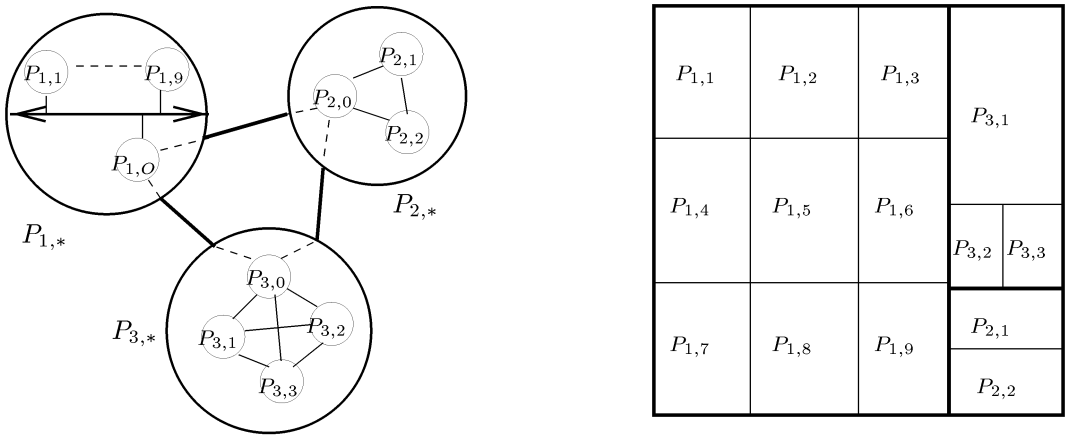


Fig. 4. Two level allocation scheme for a collection of clusters. In this example, one processor within each cluster, namely, $P_{i,0}$, is dedicated to intercluster communications.

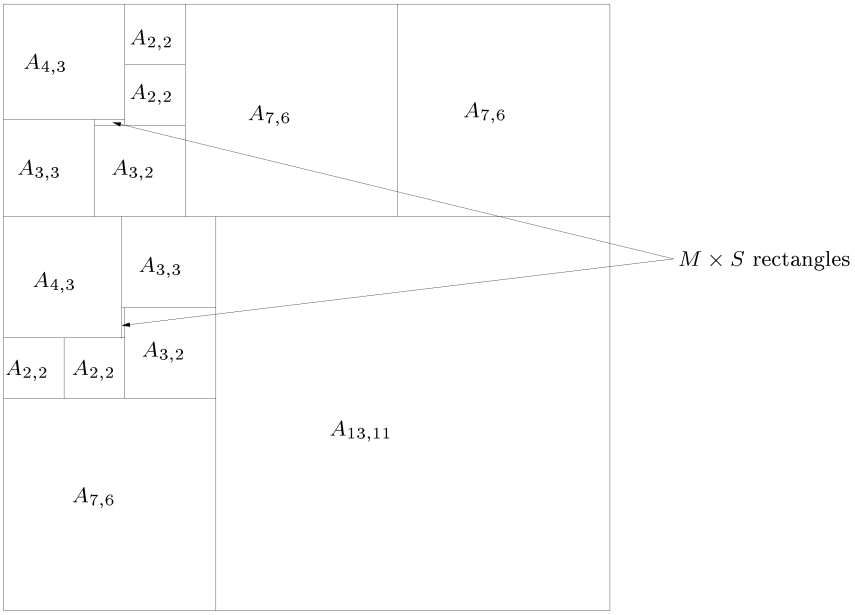


Fig. 5. General position of the squares.

R_i is the degree of freedom: We want to tile the unit square so as to minimize the total communication volume $\hat{C}$. The abstract optimization problem is the following:

Definition 1. *MM-OPT(s): Given p real positive numbers $s_1, \ldots, s_p$ s.t. $\sum_{i=1}^{p} s_i = 1$, find a partition of the unit square into p rectangles R_i of area s_i and of size $h_i \times v_i$ so that $\hat{C} = \sum_{i=1}^{p}(h_i + v_i)$ is minimized.*

Given the solution (or an approximation of the solution) of MM-OPT(s), we round up the values to the nearest integers so as to derive a concrete solution for matrices of given size n. As stated above, the integer solution will be asymptotically optimal. There is an obvious lower bound for MM-OPT(s):

Lemma 1. *For all solutions of MM-OPT(s), $\hat{C} \geq 2\sum_{i=1}^{p}\sqrt{s_i}$.*

Proof. The half-perimeter of each rectangle R_i will be always larger that $2\sqrt{s_i}$, the value when it is a square. Of course, tiling the unit square into p squares of area s_i is not always possible (think of the problem of tiling the unit square into two squares of same area 0.5), so this lower bound is not always tight. □

As already mentioned, it is easy to solve MM-OPT(s) when using a square two-dimensional grid of homogeneous processors ($s_{ij} = 1/p^2$ for $1 \leq i, j \leq p$). However, with heterogeneous processors, the MM-OPT(s) optimization problem turns out to be difficult, as shown in the next section.

3.4 NP-Completeness

The decision problem associated to the optimization problem MM-OPT is the following:

Definition 2. *MM-DEC(s,K): Given p real positive numbers $s_1, \ldots, s_p$ s.t. $\sum_{i=1}^{p} s_i = 1$ and a positive real bound K, is there a partition of the unit square into p rectangles R_i of area s_i and of size $h_i \times v_i$ so that $\sum_{i=1}^{p}(h_i + v_i) \leq K$?*

Our main result states the intrinsic difficulty of the MM optimization problem:

Theorem 1. *MM-DEC(s,K) is NP-complete.*

Because the proof is both lengthy and technical, we provide it in the Appendix. More important than the proof, the theorem itself clearly demonstrates the intrinsic difficulty of static load-balancing on heterogeneous platforms while minimizing communication cost.

The main ideas of the proof are the following:

- First, we polynomially reduce the decision problem MM-DEC(s,K) to a geometric problem (ASP) that amounts to check if there exists a partition of the unit square into squares of given areas.
- Then, we prove the NP-completeness of ASP using a polynomial reduction to the 2-Partition-Equal problem which is NP-complete [21]. The draft of this last proof is the following:

We start from an arbitrary instance of the 2-Partition-Equal problem, i.e., from a set $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ of n integers, which we aim at partitioning into two subsets of same cardinal and same sum. The idea is to build an equivalent instance of this problem using a set $\mathcal{B} = \{b_1, b_2, \ldots, b_n\}$ of n integers such that $b_i > \frac{2}{3}\max b_i$. We simply define (polynomially) $b_i = 2(a_i + 2n\max_k a_k)$. Under a few technical assumptions, we show that there exists a solution to the initial 2-Partition-Equal problem if and only if $\mathcal{B}$ can be partitioned into two subsets of same sum (not necessarily of same cardinal). Finally, we build from $\mathcal{B}$ an instance of ASP using three kinds of squares: large squares (denoted as $A_{i,j}$ in Fig. 5), n squares of size $b_i \times b_i$ (denoted as A_{b_i} in Fig. 6), and a polynomial number of other squares (denoted as $A_{\overline{b_i},j}$ in Fig. 6). We show that the only possible configuration is the one shown in Fig. 5. In this configuration, there are two nonadjacent $M \times S$ rectangular zones (where $M = \frac{4}{3}\max_i b_i$ and $S = \sum_i \frac{b_i}{2}$), which are partitioned as shown in Fig. 6.

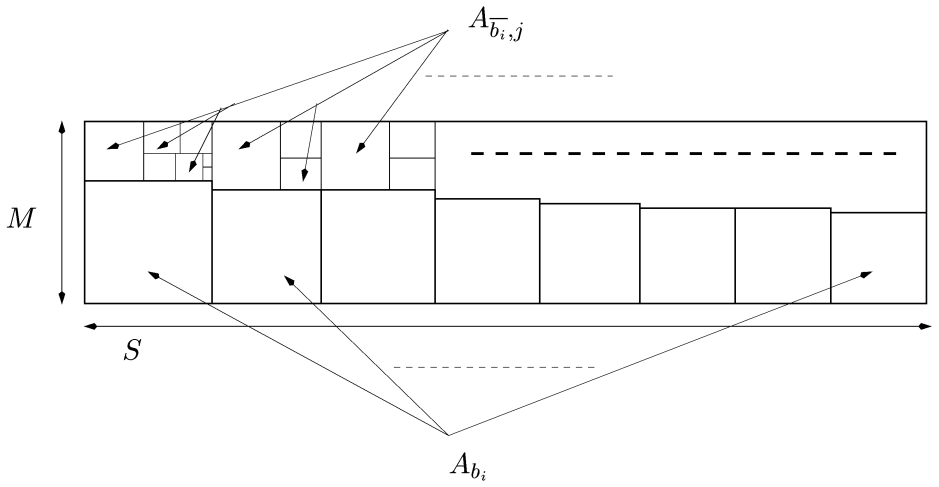


Fig. 6. oom on the $M \times S$ rectangle areas.

Because of the condition $b_i > \frac{M}{2}$, necessarily the A_{b_i} squares of size $b_i \times b_i$ are aligned. Therefore, for each rectangle the sum of the b_i is equal to S. Intuitively, the large rectangles are introduced to create the two nonadjacent rectangular zones of area $M \times S$; the n squares A_{b_i} must be aligned within these two zones, and the other squares are here to fill up the holes in the two rectangular zones.

3.5 Extensions of the Model

As pointed out in Section 3.2, minimizing the total volume of communications $\hat{C}$ seems to be a very natural goal. However, other objective functions could be selected, because the target computing platform may influence the way communications are implemented.

Ob ective Function for Parallel Communications. If *all* communications could be performed in parallel, the bottleneck would come from the processor which sends/receives the largest messages. To model such a situation, a possible objective function would be to minimize

$$\hat{M} = \max_{i=1}^{p}(h_i + v_i)$$

instead of

$$\hat{C} = \sum_{i=1}^{p}(h_i + v_i).$$

Unfortunately, the problem remains NP-complete with this objective function [3].

Ob ective Function for eterogeneous Networks. Another extension of the model could come from the modeling of the network. It is natural to consider the case of a heterogeneous network, where processors communicate through different-speed links. This problem is difficult too: It is obviously NP-complete if we use different-speed processors and if we weigh the cost of each communication with a factor proportional to the bandwidth of the communication link (because it is more complicated than MM-OPT). But interestingly, the problem remains NP-complete, even when using homogeneous processors, i.e., a heterogeneous network linking processors computing with the same speed [32]. Still, we believe that it is possible to modify the column-based heuristics presented in Section 5 to design a MM algorithm targeted to a heterogeneous network linking different speed processors.

4 Related Results

We survey related papers from the literature in this section. They range into two categories: papers dealing with linear algebra on heterogeneous platforms on one hand and papers covering geometric optimization problems similar to MM-DEC(s,K) on the other hand.

4.1 Linear Algebra on Heterogeneous Platforms

Load balancing strategies for heterogeneous platforms have been widely studied. Distributing the computations (together with the associated data) can be performed either dynamically or statically or a mixture of both. Some simple schedulers are available, but they use naive mapping strategies such as master-slave techniques or paradigms based upon the idea *use the past predict the future* , i.e., use the currently observed speed of computation of each machine to decide for the next distribution of work [14], [13], [6]. Dynamic strategies, such as *self-guided scheduling* [34], could be useful too. There is a challenge in determining a trade-off between the data distribution parameters and the process spawning and possible migration policies. Redundant computations might also be necessary to use a heterogeneous cluster at its best capabilities. However, dynamic strategies are outside the scope of this paper (and mentioned here for the sake of completeness). Because we have a library designer's perspective, we concentrate on static allocation schemes, which are less general and more difficult to design than dynamic approaches, but are better suited for the implementation of fixed algorithms such as linear algebra kernels, such as those of the ScaLAPACK library [7].

Several authors have dealt with the *static* implementation of MM algorithms on heterogeneous platforms. One simple approach is given by Kalinov and Lastovetky [26]. Their idea is to achieve a perfect load-balance as follows: First,

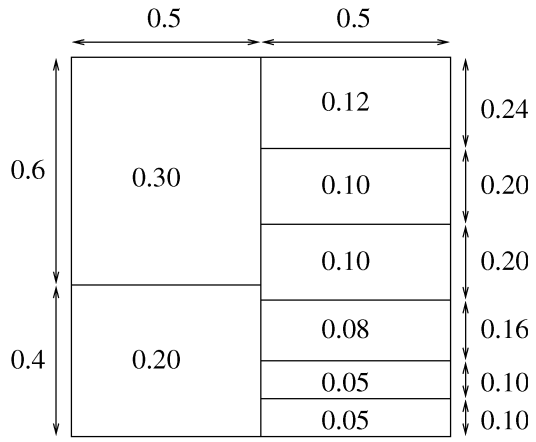


Fig. 7. The distribution of Kalinov and Lastovetky. Both columns are partitioned independently. Then, the final partition is made according to the total column weights: 0.5 versus 0.5. The total cost is $\hat{C} = 6$, which is to be compared to the cost $\hat{C} = 8$ of a partitioning into eight horizontal slices.

they take a fixed layout of processors arranged as a collection of processor columns; then, the load is evenly balanced *within* each processor column independently; next, the load is balanced *between* columns; this is the "heterogeneous block cyclic distribution" of [26], which we illustrate in Fig. 7, where we have $p = 8$ areas of values $(0.05, 0.05, 0.08, 0.1, 0.1, 0.12, 0.2, 0.3)$. This simple scheme is likely to give better results than a straightforward partitioning into horizontal slices.

Another approach is proposed by Crandall and uinn [15]. First, they compare a contiguous block allocation (see Fig. 8) to horizontal slicing; next, they introduce a better processor arrangement: They introduce a recursive algorithm to tile the iteration space (i.e., partitioning the unit square) into p rectangles of prescribed area $s_1, s_2, \ldots, s_p$ with $\sum_{i=1}^{p} s_i = 1$ so that the total communication volume is kept small. The algorithm works recursively as follows: If at some stage their remains some rectangle R of size $h.v$ to partition into q rectangles of prescribed area $s_{i_1}, s_{i_2}, \ldots, s_{i_q}$, where $\sum_{j=1}^{q} s_{i_j} = h \times v$, then partition R along its shortest dimension into two rectangles R_1 and R_2, where R_1 contains the largest $\lceil \frac{q}{2} \rceil$ rectangles and R_2 the other ones. For instance, if $h \leq v$ and $s_{i_1} \geq s_{i_2} \geq \ldots \geq s_{i_q}$, we obtain a rectangle R_1 for the $\lceil \frac{q}{2} \rceil$ largest rectangles of area $h \times v_1$, where

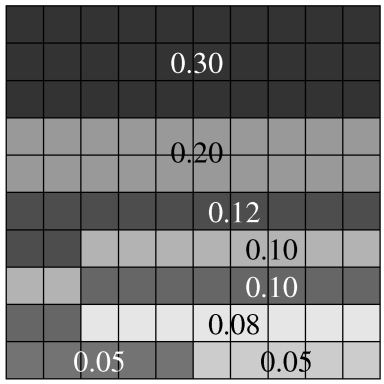


Fig. 8. Illustrating contiguous block allocation. The cost is as high as $\hat{C} = 8.2$.

$$v_1 = \frac{\sum_{j=1}^{\lceil \frac{q}{2} \rceil} s_{i_j}}{\sum_{j=1}^{q} s_{i_j}} \times v;$$

rectangle R_2 is for the remaining rectangles and of area $h \times v_2$, where $v_2 = v - v_1$ (see Fig. 9 for an illustration).

Kaddoura et al. [25] refine the previous recursive algorithm and provide several variations. They report several numerical simulations.

However, we are not aware of any theoretical result nor of any approximation bound stating some performance guarantee α (e.g., that the value $\hat{C}$ obtained by an algorithm is not α times larger than the optimal value $\hat{C}_{opt}$, where α is some constant). We have established the complexity of the problem in Section 3.4 and we provide approximation bounds in Section 5.3.

Finally, note that preliminary experimental results on implementing MM and linear system solvers on a heterogeneous network are reported in our previous papers [9], [8], [5], [10].

4.2 Problems Similar to MM-OPT(s)

There are several problems related to MM-OPT(s) in the literature:

- The most similar problem is the following: How to tile the unit square into p rectangles of same area so as to minimize the maximum perimeter of these rectangles? This problem is shown to be polynomial

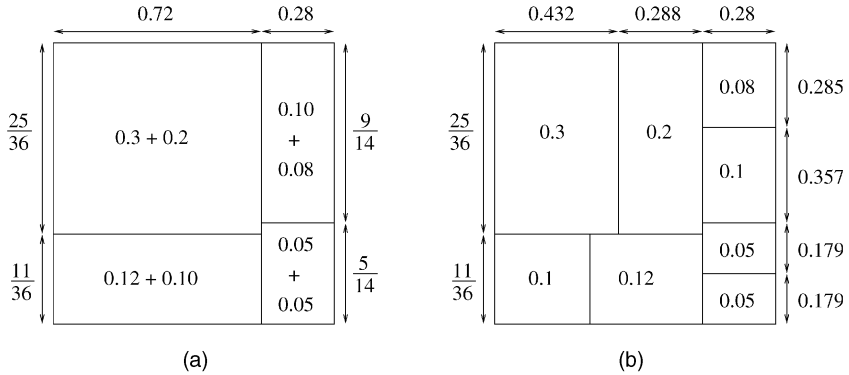


Fig. 9. Illustrating the recursive partitioning of Crandall and uinn (obtained in three steps in this example). (a) First two steps. (b) Final partitioning : $\hat{C} = 5.56$.

by Kong et al. [30], [29]. The optimal solution is one of the following two arrangements: Let either $m = \lfloor \sqrt{p} \rfloor$ or $m = \lceil \sqrt{p} \rceil$ and use m columns composed of $\lfloor \frac{p}{m} \rfloor$ or $\lceil \frac{p}{m} \rceil$ rectangles. This problem is motivated by a data-allocation problem which is related to ours in the following sense: Assume that we have p equal-speed processors and that we aim at minimizing the largest amount of communications made by one processor. Because the above arrangements are optimal, we have a polynomial solution to this problem.

The heterogeneous counterpart of this problem is the following: Given p different-speed processors, how to allocate data so that the length of the largest communication is optimized? In terms of tiling, how to tile the unit square into p nonoverlapping rectangles of prescribed area $s_1, \ldots, s_p$ whose sum is 1 so that the largest perimeter is minimized? This interesting problem is NP-complete too [3], which again shows the intrinsic difficulty of designing heterogeneous parallel algorithms!

- Another related problem is to find the minimum partition of a rectangle with interior points. Given a rectangle R and a finite set P of points located inside R, find a set of line segments that partition R into rectangles such that every point in P is on the boundary of some rectangle. The goal is to minimize the total length of the introduced line segments. This problem is shown NP-complete in [33], [22], [23], where approximation algorithms are given. The link with our problem is that the objective function is the same, but the original motivation in [22], [23] was a VLSI routing problem (and the constraints are quite different).
- There are several NP-complete geometric optimization problems that are listed in [16]. One example is the minimum rectangle tiling problem [28]: Given an $n \times n$ array A of nonnegative numbers and a positive integer p, find a partition of A into p nonoverlapping rectangular subarrays such that the maximum weight of any rectangle in the partition is minimized (the weight of a rectangle is the sum of its elements).

5 Heuristics

In this section, we introduce a polynomial heuristic to solve the MM-OPT problem. After describing the heuristic and proving its optimality among all column-based approaches, we report experimental results that nicely demonstrate its efficiency. Finally, we provide a theoretical guarantee for the heuristic and we discuss possible extensions.

5.1 Optimal Column-Based Tiling

As outlined in Section 3.3, the MM-OPT(s) problem is the following: Given p real positive variables $s_1, \ldots, s_p$ such that $\sum_{i=1}^{p} s_i = 1$, tile the unit square into p nonoverlapping rectangles $R_1, \ldots, R_p$ of respective areas $s_1, \ldots, s_p$ so as to minimize the sum of the (half) perimeters of these rectangles. Because the associated decision problem MM-DEC(s,K) is NP-complete (Section 3.4), we consider the more constrained problem MM-COL(s), where we impose that the tiling is made up of processor columns, as illustrated in Fig. 10. In other words, MM-COL(s) is the restriction of MM-OPT(s) to column-based partitions. In this section, we give a polynomial solution to the MM-COL(s), which will be used as a heuristic to solve MM-OPT(s).

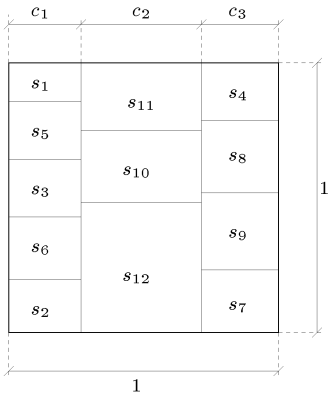


Fig. 10. Column-based partitioning of the unit square: $\mathcal{C} = 3$, $k_1 = 5$, $k_2 = 3$, and $k_3 = 4$.

Framework. We describe the MM-COL(s) problem more formally. We aim at tiling the unit square into $\mathcal{C}$ columns (where $\mathcal{C}$ is yet to be determined) of width $c_1, \ldots, c_{\mathcal{C}}$. Each column C_i is partitioned itself into k_i rows (to be determined too) of respective area $s_{\sigma(i,1)}, \ldots, s_{\sigma(i,k_i)}$. Of course, the final partitioning has $\sum_{i=1}^{\mathcal{C}} k_i = p$ rectangles and all the areas $s_1, \ldots, s_p$ are represented once and only once. The goal is to build such a partitioning subject to the minimization of the sum of the rectangle perimeters.

lgorithm 1. We describe our algorithm which is based upon the dynamic programming paradigm; the optimality proof will be presented later. The main points are the following:

1. Reindex the variables $s_1, \ldots, s_p$ such that
$$s_1 \leq s_2 \leq \ldots \leq s_p.$$
2. Iteratively build the function $f_{\mathcal{C}}$, by incrementing the value of $\mathcal{C}$ from 1 to the desired value. For $q \in \{1, \ldots, p\}$, $f_{\mathcal{C}}(q)$ represents the total perimeter of an optimal column-based partitioning of a rectangle of height 1 and width $\left(\sum_{i=1}^{q} s_i\right)$ into q rectangles of respective area $s_1, \ldots, s_q$ using $\mathcal{C}$ columns.

To help understand the derivation, we apply the algorithm on the example we have used throughout Section 4.1. We have $p = 8$ areas of values

$$(0.05, 0.05, 0.08, 0.1, 0.1, 0.12, 0.2, 0.3).$$

The results of the algorithm are described in Table 1 and the resulting partitioning is depicted in Fig. 11. Each column $\mathcal{C}_i$ contributes to the sum of the half-perimeters as follows: 1 for the vertical line and $k_i \times c_i$ for the k_i horizontal lines of length c_i.

In the example, the optimal partitioning is obtained for three columns ($f_3(8) = 5.5$). The first column of width $c_3 = s_7 + s_8 = 0.5$ is composed of 2 elements. The second column of width $c_2 = s_4 + s_5 + s_6 = 0.32$ is composed

TABLE 1

Table Containing the Values of the Couples $f_{\mathcal{C}}(q)/r$ where $f_{\mathcal{C}}(q) = \min_{r\in[\mathcal{C}-1,q-1]}\left(1 + \left(\sum_{r<i\leq q} s_i\right) \times (q-r) + f_{\mathcal{C}-1}(r)\right)$ and r is the Value Minimizing the Previous Expression

	q=1	q=2	q=3	q=4	q=5	q=6	q=7	q=8
$\mathcal{C}=1$	1.05 / 0	1.2 / 0	**1.54 / 0**	2.12 / 0	2.9 / 0	4 / 0	5.9 / 0	9 / 0
$\mathcal{C}=2$		2.1 / 1	2.28 / 2	2.56 / 2	2.94 / 3	**3.5 / 3**	4.38 / 4	5.76 / 5
$\mathcal{C}=3$			3.18 / 2	3.38 / 3	3.66 / 4	4 / 4	4.58 / 5	**5.5 / 6**
$\mathcal{C}=4$				4.28 / 3	4.48 / 4	4.78 / 5	5.2 / 6	5.88 / 7
$\mathcal{C}=5$					5.38 / 4	5.6 / 5	5.98 / 6	6.5 / 7
$\mathcal{C}=6$						6.5 / 5	6.8 / 6	7.28 / 7
$\mathcal{C}=7$							7.7 / 6	8.1 / 7
$\mathcal{C}=8$								9 / 7

of three elements. Then, the last column of width $c_1 = s_1 + s_2 + s_3 = 0.18$ is made up with the smallest three elements (see Fig. 11).

The algorithm is outlined in Table 2. ($f_{\mathcal{C}}^{perimeter}$ corresponds to the $f_{\mathcal{C}}(q)$ previously used. $f_{\mathcal{C}}^{cut}(q)$ corresponds to the total number of blocs in the first $\mathcal{C}-1$ columns so that there remains $q - f_{\mathcal{C}}^{cut}(q)$ blocs in the column $\mathcal{C}$).

The worst case complexity of the algorithm is $O(p^3)$. Note that, in practice, the complexity will be lower than the worst-case analysis shows because $f_{\mathcal{C}}(p)$ is a function that is first decreasing and then increasing as $\mathcal{C}$ varies. All the functions $f_{\mathcal{C}}$ will not be built and the expected cost will be $p^2\mathcal{C}_{opt} \approx p^{2.5}$.

The final partitioning corresponding to the function $f_{\mathcal{C}_{opt}}(p) = \min_{1\leq\mathcal{C}\leq p} f_C(p)$ is found using the algorithm in Table 3, which corresponds to tracking (backwards) the bold entries in Table 1. The unit square is partitioned into $\mathcal{C}_{opt}$ columns. The ith column contains the rectangles $s_{d+1}, \ldots, s_{d+k_i}$ with $d = k_1 + k_2 + \ldots + k_{i-1}$.

Correctness. To prove the optimality of the algorithm, we show that the optimum solution can be achieved with a *well-ordered partitioning*. A partitioning is said to be well

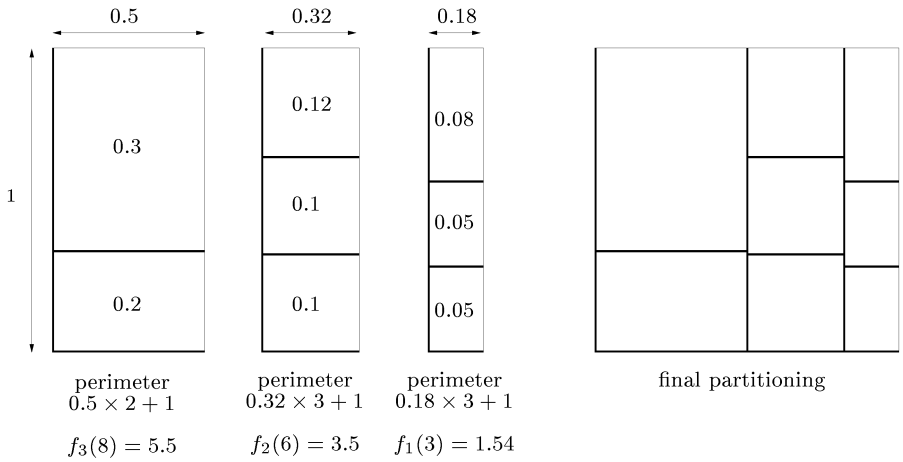


Fig. 11. Optimal column-based partitioning for the example. Thicker lines correspond to the sum of the half-perimeters. We obtain $\hat{C} = 5.5$.

TABLE 2
Algorithm 1

$S = 0$
for q=1 to p
  $S = S + s_q$
  $f_1^{perimeter}(q) = 1 + Sq$
  $f_1^{cut}(q) = 0$
endfor
for $\mathcal{C}$=2 to p
  for q=$\mathcal{C}$ to p
    $f_{\mathcal{C}}^{perimeter}(q) = \min_{r\in[\mathcal{C}-1,q-1]}\left(1 + \sum_{q-r<i\leq q} s_i(q-r) + f_{\mathcal{C}-1}^{perimeter}(r)\right)$
    $f_{\mathcal{C}}^{cut}(q) = r_{opt}$ {where r_{opt} reaches the minimum in the previous expression}
  endfor
endfor

TABLE 3
Algorithm 2

$q = p$
for $\mathcal{C} = \mathcal{C}_{opt}$ **downto** 2
  $k_{\mathcal{C}} = q - f_{\mathcal{C}}^{cut}(q)$
  $q = f_{\mathcal{C}}^{cut}(q)$
endfor
$k_1 = q$

ordered if for every pair of columns $\mathcal{C}_i$ and $\mathcal{C}_j$, either all the elements of $\mathcal{C}_i$ are smaller than (or equal to) all the elements of $\mathcal{C}_j$, or the other way round. See Fig. 12 for an illustration.

We start from a given partitioning made up of, say, $\mathcal{C}$ columns of size $k_1 \geq k_2 \geq \ldots \geq k_{\mathcal{C}}$. Suppose, for convenience, that $s_1 \leq s_2 \leq \ldots \leq s_p$; τ is a permutation of $\{1, 2, \ldots, p\}$ such that the ith column of this partitioning contains the rectangles $s_{\tau(d+1)}, \ldots, s_{\tau(d+k_i)}$ with

$$d = k_1 + k_2 + \ldots + k_{i-1}.$$

Now, recall that the cost of column C_i is $1 + k_i \sum_{j=d+1}^{d+k_{i-1}} s_{\tau(j)}$. Hence, the total "perimeter" is

$$\begin{aligned}
&\mathcal{C} + k_1 s_{\tau(1)} + k_1 s_{\tau(2)} + \ldots + k_1 s_{\tau(k_1)} \\
&+ k_2 s_{\tau(k_1+1)} + k_2 s_{\tau(k_1+2)} + \ldots + k_2 s_{\tau(k_1+k_2)} \\
&+ \ldots \\
&+ k_{\mathcal{C}} s_{\tau(k_1+\ldots+k_{\mathcal{C}-1}+1)} + k_{\mathcal{C}} s_{\tau(k_1+\ldots+k_{\mathcal{C}-1}+2)} + \ldots + k_{\mathcal{C}} s_{\tau(k_1+\ldots+k_{\mathcal{C}})}.
\end{aligned}$$

Since $k_1 \geq k_2 \geq \ldots \geq k_{\mathcal{C}}$, this expression is minimized for $\tau = Identity$, which corresponds to a "well-ordered" partitioning.[2] Hence, for each partitioning, there exists a corresponding better or equivalent partitioning that is "well ordered." This achieves the proof of correctness.

5.2 Experimental Comparison with the Lower Bound

As shown in Section 3.3, a lower bound for the sum $\hat{C}$ of the half-perimeters is twice the sum of the square roots of the areas $LB = 2\sum_{i=1}^{p} \sqrt{s_i}$. Of course, this bound cannot always be met. Consider an instance of MM-OPT(s) with only two processors, $s_1 = 1 - \epsilon$ and $s_2 = \epsilon$, where $\epsilon > 0$ is an arbitrarily small number. Partitioning into two rectangles requires to draw a line of length 1, hence, $\hat{C} = 3$. However, $LB = 2(\sqrt{1-\epsilon} + \sqrt{\epsilon}) > 2$ can be arbitrarily close to 2.

In this section, we experimentally compare, using a large number of random tests, the value $\hat{C}$ given by our partitioning against the absolute lower bound LB.

- Because the ratio between the processor speeds is not likely to be very large in practice (who would use a machine 100 times slower than another one?),[3] we assume that a uniform repartition of the processor speeds might be significant. The goal of Fig. 13 is to show that the column based partitioning is efficient in most reasonable situations. We randomly generate a large number of set of speeds with a uniform repartition. We represent two curves for a number of processors varying from 1 to 40. The first curve corresponds to the mean value of the ratio $\frac{\hat{C}}{LB}$ while the second curve gives the minimum values of this ratio. We see that on average, the optimal column-based tiling given by our algorithm gives a solution that is "almost" optimal, so that we can be satisfied with the results for all practical purposes.
- Next, we adopt a theoretical point of view and concentrate on worst cases. A uniform repartition is no longer acceptable with such a purpose. Hence, in Fig. 14, we generate a large number of sets of speeds using an exponential repartition. Because the ratio $r = \frac{\max s_i}{\min s_i}$ plays an important role in the cost of the worst case, we display the tests for different values of r varying from 2 to ∞.

2. The proof can easily be done by induction on the number of inversions in the permutation τ.

3. In fact, using 100 slower machines in conjunction to a fast one does make sense in some cases!

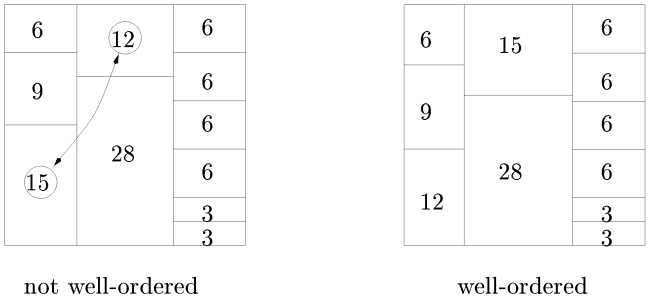


Fig. 12. Two partitionings of the same problem instance. The right one is well ordered while the left one is not.

5.3 Theoretical Comparison with the Lower Bound

The column-based heuristic appears to be quite satisfactory in practice. In this section, we prove it is (theoretically) not far from being optimal, especially when the ratio r between $\max s_i$ and $\min s_i$ is small. In other words, we are able to give the following guarantee to the column-based heuristic:

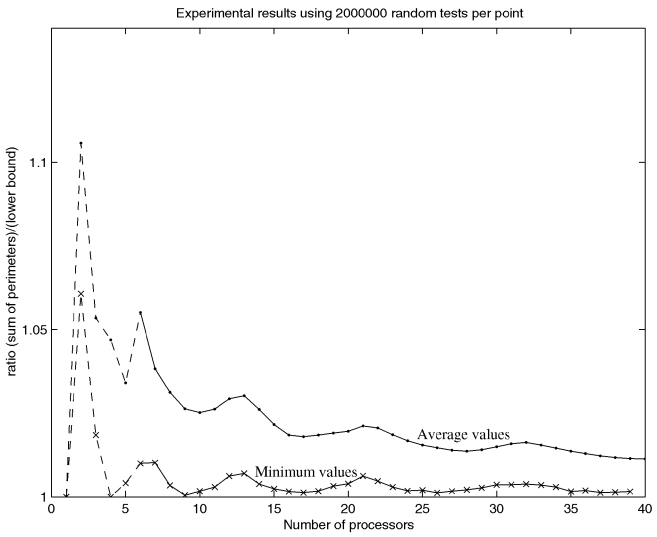


Fig. 13. For each number of processors (varying from 1 to 40), 2,000,000 values for the s_i have been randomly generated. For each case, we compute the ratio of the sum $\hat{C}$ of the half-perimeters of our partitioning over the absolute lower bound LB. The *average* and *minimum* values of this ratio are reported in the two curves.

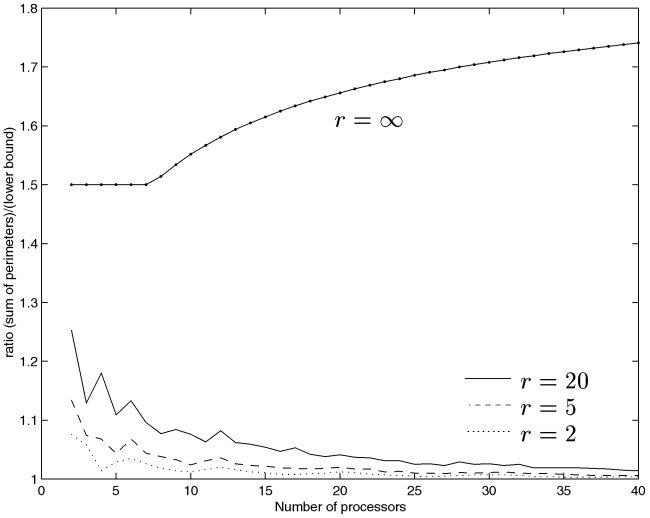


Fig. 14. For each number of processors (varying from 2 to 40) and for different values of $\frac{r=\max s_i}{\min s_i}$, 10,000 values for the s_i have been randomly generated using an exponential distribution. For each case, we compute the ratio of the sum $\hat{C}$ of the half-perimeters of our partitioning over the absolute lower bound LB. The *maximum* values of this ratio are reported in the four curves.

Proposition 1. *Let $r = \frac{\max s_i}{\min s_i}$ and let $\hat{C}$ denote the sum of the half-perimeters of the rectangles obtained with the optimal column-based partitioning. Then,*

$$\frac{\hat{C}}{2\sum\sqrt{s_i}} \leq \sqrt{r}\left(1 + \frac{1}{\sqrt{p}}\right).$$

Proof. Consider a column based partitioning with

$$\mathcal{C} = \lceil \sqrt{r} \sum \sqrt{s_i} \rceil$$

columns. Rectangles are evenly distributed amongst columns so that the numbers of rectangles in each column is either $\lfloor \frac{p}{\mathcal{C}} \rfloor$ or $\lceil \frac{p}{\mathcal{C}} \rceil$. Letting $\hat{C}^*$ denote the sum of the half-perimeters of the rectangles obtained with this column partitioning, we have:

$$\begin{aligned}\hat{C}^* \leq & \lceil \sqrt{r} \sum \sqrt{s_i} \rceil + \left\lceil \frac{p}{\lceil \sqrt{r} \sum \sqrt{s_i} \rceil} \right\rceil \\ \leq & 2 + \sqrt{r} \sum \sqrt{s_i} + \frac{p}{\sqrt{r} \sum \sqrt{s_i}}.\end{aligned}$$

Thus,

$$\frac{\hat{C}^*}{2\sum\sqrt{s_i}} \leq \frac{1}{\sum\sqrt{s_i}} + \frac{\sqrt{r}}{2} + \frac{p}{2\sqrt{r}\sum\sqrt{s_i}}.$$

Moreover,

$$\begin{aligned}\sum s_i = 1 \Longrightarrow & p \max s_i \geq 1 \\ \Longrightarrow & \min s_i \geq \frac{1}{pr}\end{aligned}$$

and, thus,

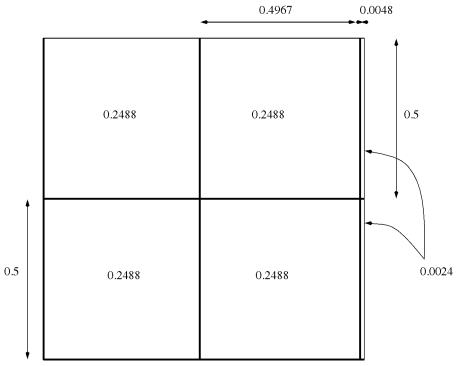


Fig. 15. Optimal column-based partitioning. The sum of the half-perimeters is 5.

$$\sum \sqrt{s_i} \geq p\sqrt{\min s_i} \geq \sqrt{\frac{p}{r}}.$$

Therefore,

$$\begin{aligned}\frac{\hat{C}^*}{2\sum\sqrt{s_i}} \leq & \sqrt{\frac{r}{p}} + \frac{\sqrt{r}}{2} + \frac{\sqrt{r}}{2} \\ \leq & \sqrt{r}(1 + \frac{1}{\sqrt{p}}).\end{aligned}$$

Since $\widehat{C}$ corresponds to the best solution among all possible column-based partitioning, $\hat{C}$ satisfies to $\hat{C} \leq \hat{C}^*$, which concludes the proof. □

If $r = 1$, i.e., all the processors have the same speed, the column-based partitioning is asymptotically optimal. On the other hand, if r is large, i.e., one processor is much faster than another, the bound is very pessimistic. In [4], we have considered a recursive heuristic, very complicated to describe, but for which a better approximation bound is provided. For all practical purposes, we believe that the column-based algorithm is the best trade-off between efficiency and simplicity.

5.4 Looking for a Better Solution

As already said, this section is mostly theoretical. We investigate new algorithms for the sake of improving the column-based solution, which is very satisfactory except in some "degenerate" artificial cases. To illustrate the point, consider the following partitioning problem into $p = 6$ rectangles of respective areas

$$(0.2488, 0.2488, 0.2488, 0.2488, 0.0024, 0.0024)$$

(the relative cycle-times of the six processors are approximately $(1, 1, 1, 1, 100, 100)$). The absolute lower-bound for this example is $LB = 2\sum_{i=1}^{6}\sqrt{s_i} = 4.19$. Consider the following solutions, which have different degrees of freedom:

1. The partitioning is constrained to be a column-based partitioning. Using the column-based algorithm, we obtain the solution depicted in Fig. 15.
2. The partitioning is constrained to be recursively defined as follows: The unit square is divided into several columns. Each column is in turn divided into

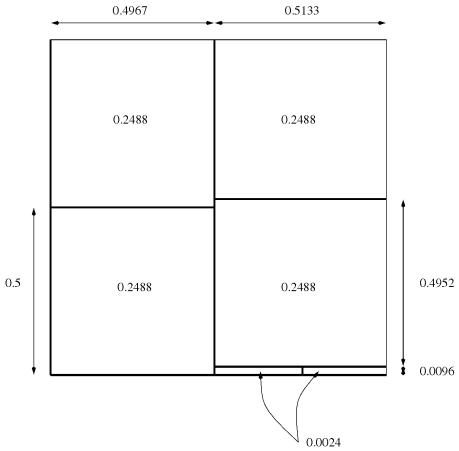


Fig. 16. Optimal recursively defined partitioning. The sum of the half-perimeters is 4.51.

several rows and so on. Of course, there are multiple choices for the number of columns and for the number of rows within each column and so on. In Fig. 16, we give an example with two columns divided into two and three rows, respectively. Finally, the last row of the second column is split into two rectangles. In the example, this partitioning is optimal amongst recursively defined partitionings (proof by exhaustive case analysis).

3. The partitioning is only constrained to be made out of rectangles. An example is given in Fig. 17. Note that this solution is neither column-based nor recursively defined. This partitioning is optimal among all rectangle-based partitionings.

Clearly, the less constrained the partitioning, the better the solution. Note that the improvement over the column-based partitioning can be important, roughly 16 percent for the rectangle-based solution. Again, in a realistic experiment, we would never use a processor in conjunction with another one which is 100 times faster! Also, with a library designer's perspective, a simple column-based partitioning has many advantages regarding code generation issues.

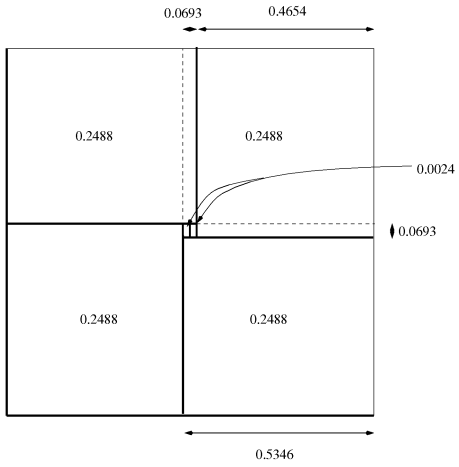


Fig. 17. Optimal rectangle-based partitioning. The sum of the half-perimeters is 4.19.

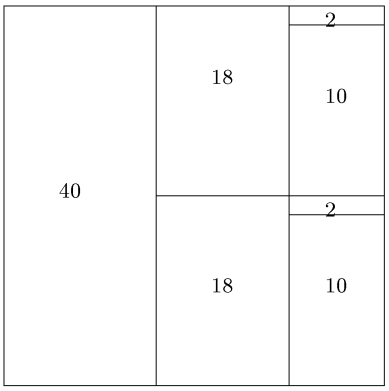


Fig. 18. Partition given by the column-based heuristic (Cost $\hat{C} = 5.1$).

6 MPI Experiments

To provide a preliminary experimental validation of our approach, we have implemented the heterogeneous MM algorithm using the MPI library [35]. In this section, we report a few experiments performed on a HNOW and on a (very small!) collection of two clusters.

6.1 Using a Single HNOW to Compare Different Partitions

In this section, we use a cluster of seven heterogeneous machines of relative cycle-times equal to $(1, 1, \frac{1}{5}, \frac{1}{5}, \frac{1}{9}, \frac{1}{9}, \frac{1}{20})$. These seven machines are SUN workstations of our laboratory, linked by a simple Ethernet network. We compare the partition given by the optimal column-based heuristic (see Fig. 18) with four different partitions of the same matrices which are shown in Fig. 19.

The measures were realized for matrices of size $n = 640$ using a blocksize $r = 32$ and for matrices of size $n = 1280$ using two blocksize values, $r = 32$ and $r = 64$. Table 4 gives the average time to compute the MM product for the five partitions. In the case of a matrix of size $n = 1280$, we see that the time is slightly smaller if we increase the blocksize because there are fewer communications. We check that the execution time does grow with the cost of the partition, which shows that our modeling of the communication costs is very reasonable and is in good adequation with these experiments. Note that (for fairness) we have not compared the results with the homogeneous block-cyclic distribution. Because the processor speeds are very different, the performances would have been disastrous.

6.2 Experimenting with Two Clusters

In this section, the target platform is made up of two clusters. The first cluster is a pile of Pentium Pros and the second cluster is a pile of Power PCs. The interconnection network within both clusters is a Myrinet network. There is also a Myrinet link between the two clusters. Hence, all communications are very fast. In the experiments, either we allocate to each cluster a fraction of the matrix which is proportional to its computing power, according to Section 3.1, or we give the same fraction to each processor, as in the homogeneous case. When we use the load-balancing strategy, we use each cluster as a farm of processors and equally distribute the workload inside the farm.

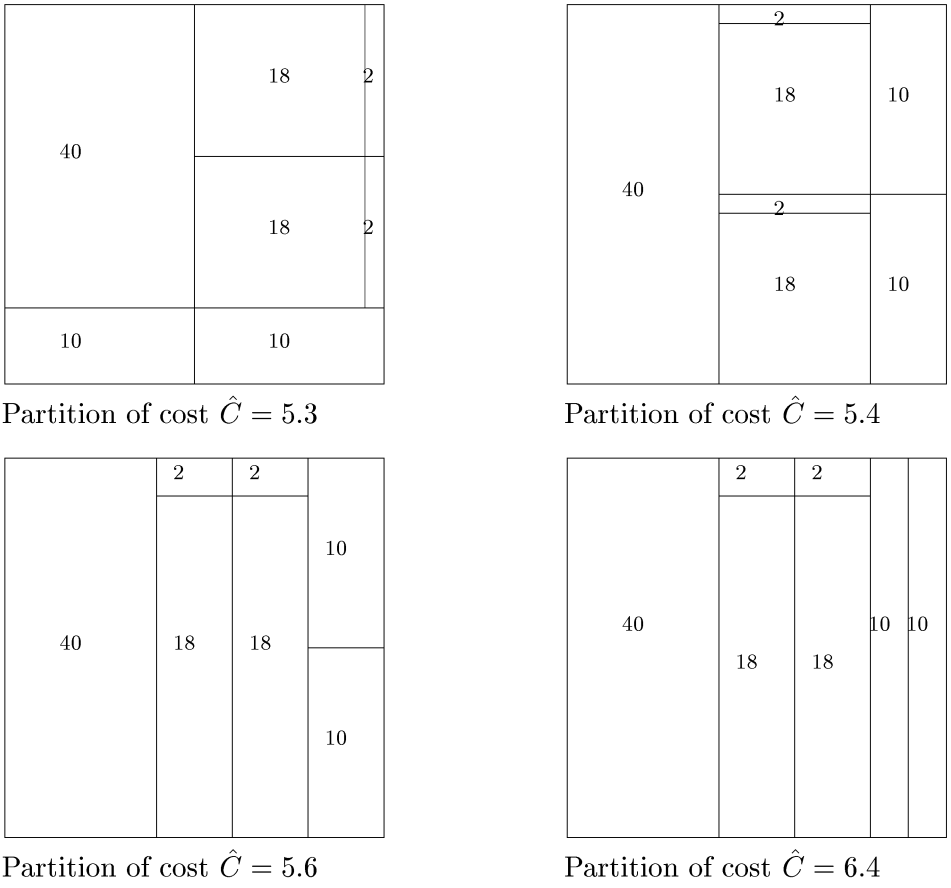


Fig. 19. Four different column-based partitions.

We use two configurations, one with five processors in the first cluster and three in the second one and the other one with only four processors in the first cluster and two processors in the second one. In both cases, the gain of the load-balancing heuristic over the homogeneous block-cyclic distribution (a meaningful comparison here because the processor speeds are rather similar) is very important (see Fig. 20).

7 Conclusion

In this paper, we have dealt with the implementation of MM algorithms on heterogeneous platforms. The bad news is that minimizing the total communication volume is NP-complete. The good news is that efficient polynomial heuristics can be provided, as we have shown, both theoretically (by guaranteeing their performance) and through simulations and MPI experiments. At the very least, the MPI experiments fully demonstrate the importance of using a good load-balancing strategy.

TABLE 4
Average Times for a Matrix Multiplication

Partition	n=640 r=32	n=1280 r=32	n=1280 r=64
$\hat{C}$=5.1	T=33	T=269	T=258
$\hat{C}$=5.3	T=34	T=287	T=265
$\hat{C}$=5.4	T=48	T=289	T=264
$\hat{C}$=5.6	T=34	T=290	T=267
$\hat{C}$=6.4	T=72	T=367	T=302

Future work could aim at testing a larger collection of clusters with slower intercluster links. The Globus system [18] provides a perfect framework for such experiments because hardware resources are used in a dedicated mode through a remote batch system so that static load-balancing strategies such as the one presented in this paper have all their significance.

The MM algorithm is the prototype of tightly-coupled kernels with a high spatial locality that need to be implemented efficiently on distributed and heterogeneous platforms. We view it as a perfect testbed before experimenting more challenging computational problems on the grid.

It is not clear which is the good level to program metacomputing platforms. Data-parallelism seems unrealistic due to the strong heterogeneity. Explicit message passing is too low-level. Despite their many advantages, object-oriented approaches (e.g., [24], [2]) still request the user to have a deep knowledge and understanding of both its application behavior and the underlying hardware and network. Remote computing systems such as NetSolve [11] face severe limitations to efficiently load-balance the work to processors. For the inexperienced user, relying on specialized but highly-tuned libraries of all kinds (communication, scheduling, application-dependent data decompositions) may prove a good trade-off until the programming environments evolve into high-level general-purpose yet efficient solutions.

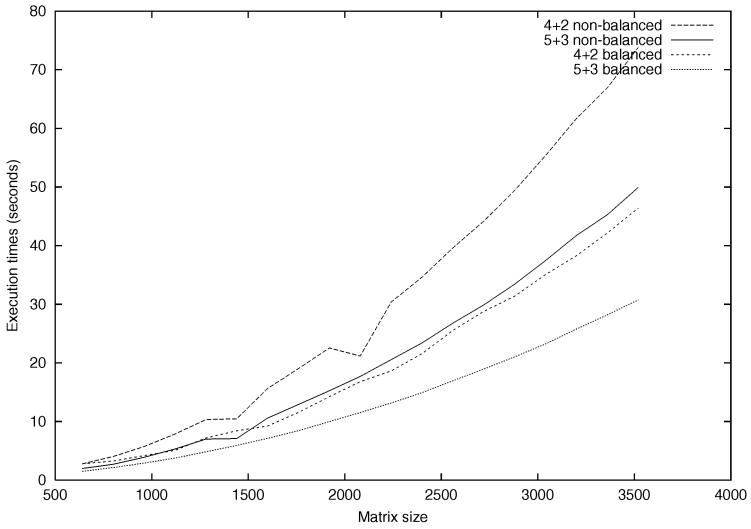


Fig. 20. MPI experiments with two clusters.

APPENDIX A
PROOF OF THEOREM 1

Definition 2 MM-DEC(s,). *Given p real positive numbers $s_1, \ldots, s_p$ s.t. $\sum_{i=1}^{p} s_i = 1$ and a positive real bound K, is there a partition of the unit square into p rectangles R_i of area s_i and of size $h_i \times v_i$ so that $\sum_{i=1}^{p}(h_i + v_i) \leq K$?*

Theorem 1. *MM-DEC(s,K) is NP-complete.*

Proof. Obviously, MM-DEC(s,K) $\in$ NP. In this section, we prove the following lemma. □

Lemma 2. *$2P - eq \leq_P \ ASP \leq_P \ MM - DEC$, where 2P-eq and ASP are defined as follows:*

Definition 3. 2-Partition-E ual (2P-e). *Given a set of p integers $\mathcal{A} = \{a_1, \ldots, a_p\}$, is there a partition of $\{1, \ldots, p\}$ into two subsets $\mathcal{A}_1$ and $\mathcal{A}_2$ such that*

$$\sum_{i \in \mathcal{A}_1} a_i = \sum_{i \in \mathcal{A}_2} a_i \text{ and } card(\mathcal{A}_1) = \ card(\mathcal{A}_2) \ ?$$

Definition 4. ll-S uares-Partition (SP). *Given a set*

$$\mathcal{L} = \{l_1, \ldots, l_p\}$$

of p real positive numbers such that $\sum_{i=1}^{p} l_i^2 = 1$, is there a partition of the unit square into p squares S_i of width l_i?

Since 2P-eq is known to be NP-Complete [21], Lemma 2 will complete the proof of Theorem 1.

A.1 Reduction: ASP $\leq_P$ MM-DEC(s,K)

We start by proving the easy part of Lemma 2, i.e., ASP $\leq_P$ MM $-$ DEC(s, K). Let $\mathcal{L} = \{l_1, \ldots, l_p\}$ be a set of p real positive numbers s.t. $\sum_{i=1}^{p} l_i^2 = 1$. Solving ASP is equivalent to solving MM-DEC(s,K) with

$$\begin{cases} K = \sum_{i=1}^{p} 2l_i \\ \forall i, \ s_i = l_i^2 \end{cases}$$

and, therefore, ASP $\leq_P$ MM $-$ DEC.

A.2 Reduction: 2P-eq $\leq_P$ ASP

In this section, we consider an arbitrary instance of the 2-Partition-Equal problem, i.e., a set $\mathcal{A} = \{a_1, \ldots, a_n\}$ of n integers. We assume that $n \geq 400$ without loss of generality. We have to polynomially transform this instance into an instance of the ASP problem which has a solution iff the original instance of 2-Partition-Equal has one solution.

Define $\{b_1, \ldots, b_n\}$ as

$$\forall i, b_i = 2(a_i + 2n \max_k a_k).$$

Let $N = \max_k b_k$. Then, $b_i > \frac{2N}{3}$ and b_i is even. Moreover, if we let $M = \frac{4N}{3}$ and

$$S = \frac{\sum_i b_i}{2},$$

then $S \geq 100M$. We also have $\frac{M}{2} < b_i \leq \frac{3M}{4}$ for all i. The reason for introducing M is that we will tile the n rectangles $\overline{R_i}$ of size $b_i \times (M - b_i)$ into a minimal number of squares $KS(i)$, following the procedure of Kenyon [27]. Here, KS stands for *Kenyon's squares*. To get a logarithmic number of squares $KS(i)$, the rectangle $\overline{R_i}$ must not be too elongated, which is ensured by the inequality $M - b_i < b_i \leq 3(M - b_i)$. We obtain from [27] that $KS(i) \leq 3 + C \log b_i$, where C is a universal constant. In the following, for $1 \leq i \leq n$, we let $w(\overline{b_i}, j)$, $1 \leq j \leq KS(i)$, denote the widths of the $KS(i)$ squares obtained by the procedure in [27] to tile the rectangle $\overline{R_i}$ of size $b_i \times (M - b_i)$.

We build the following instance $\mathcal{L}$ of the ASP problem (ASP($b_1, \ldots, b_n$)): Is there a partition of the unit square into $14 + n + \sum_{i=1}^{n} KS(i)$ squares of respective width

$$\begin{array}{llll} \frac{(13S+11M)}{l} & (\times 1), & \frac{(7S+6M)}{l} & (\times 3), \\ \frac{(3S+2M)}{l} & (\times 2), & \frac{(2S+2M)}{l} & (\times 4), \\ \frac{(4S+3M)}{l} & (\times 2), & \frac{(3S+3M)}{l} & (\times 2), \\ \frac{b_i}{l} & (\forall i), & \frac{w(\overline{b_i},j)}{l} & (\forall i, \ \forall j \leq KS(i)), \end{array}$$

where $l = (20S + 17M)$?

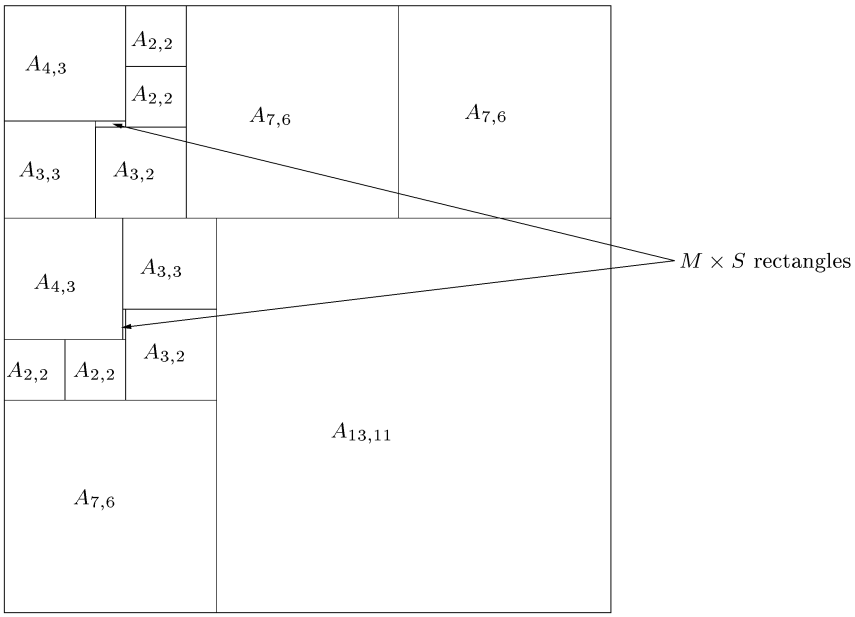


Fig. 21. General position of the squares.

For convenience, in what follows, we consider the (equivalent) scaled problem: Is there a partition of the $(20S + 17M) \times (20S + 17M)$ square into $14 + n + \sum_i KS(i)$ squares of respective width

$$\begin{cases} 13S + 11M & (\times 1), \\ 7S + 6M & (\times 3), \\ 4S + 3M & (\times 2), \\ 3S + 3M & (\times 2), \\ 3S + 2M & (\times 2), \\ 2S + 2M & (\times 4), \\ b_i & (\forall i,\ 1 \le i \le n), \\ w(\overline{b_i}, j) & (\forall i,\ 1 \le i \le n,\ \forall j,\ 1 \le j \le KS(i))\ ? \end{cases}$$

From now on, $A_{x,y}$ denotes a square of width $(xS + yM)$, A_{b_i} denotes a square of width b_i, and $A_{\overline{b_i},j}$ denotes a Kenyon's square of width $w(\overline{b_i}, j)$. In what follows, we prove that such a partition is necessarily the one depicted in Fig. 21, where the two small $M \times S$ rectangle areas are shown by arrows in Fig. 21 and fully described in Fig. 22. The intuitive idea of the proof is the following: The large squares are used to prevent the two small $M \times S$ rectangular zones to be neighbors. Hence, these areas must be filled separately by the remaining squares, namely the squares A_{b_i} and the Kenyon's squares. This will be possible iff the b_i can be partitioned into two subsets of same sum, which in turn will be possible iff the a_i can be partitioned into two subsets of same sum and same cardinal. The Kenyon's squares are introduced to fill the holes in the two rectangular zones and to obtain a true tiling of the whole area.

.2.1 Position of the argest Four S uares

The general position of the largest four squares is shown in Fig. 23a. Obviously, if we can tile the remaining area with the remaining squares, this will also be the case for the configuration shown in Fig. 23b. Therefore, from now on,

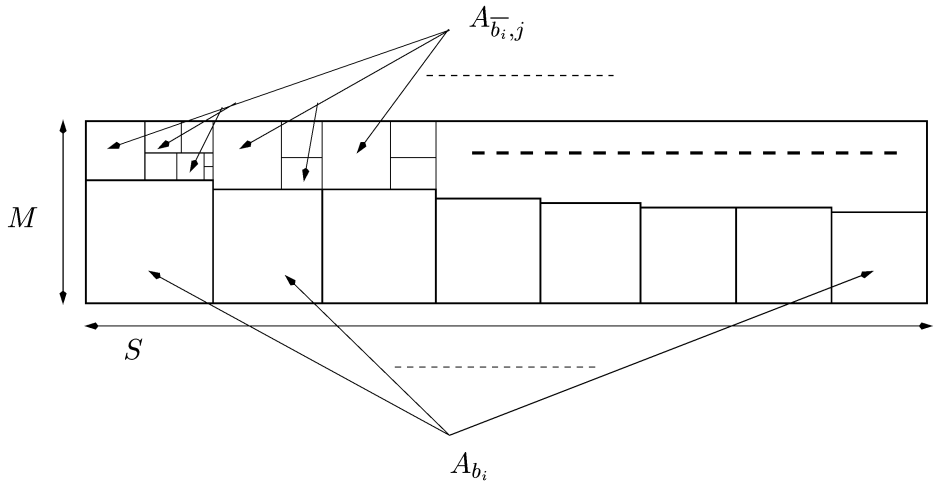


Fig. 22. oom on the $M \times S$ rectangular zones.

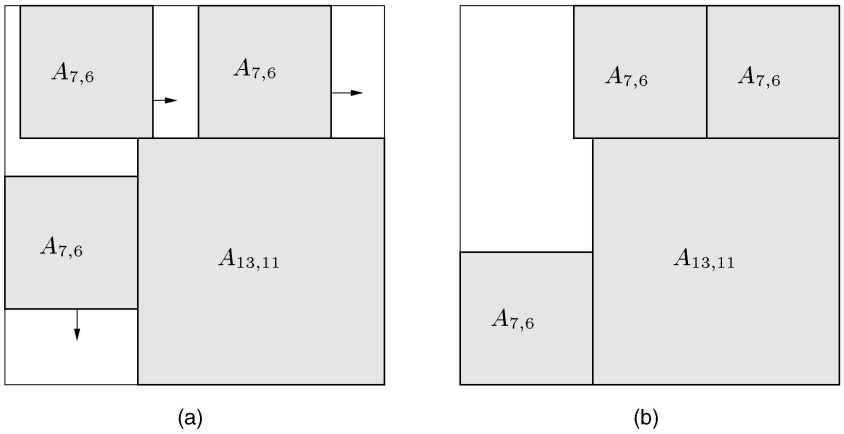


Fig. 23. General position of the largest four squares.

we assume (without loss of generality) that the largest four rectangles are arranged as shown in Fig. 23b.

.2.2 Tiling the Remaining Surface

Now, we discuss the tiling of the remaining surface (the white area of Fig. 23b). We give all dimensions in Fig. 24). We proceed by an exhaustive case analysis:

- First case: We start by tiling the $6S + 5M$ basis with a $4S + 3M$ square.
  The different situations to consider are shown in Fig. 25, Fig. 26, and Fig. 27, respectively. The only correct configuration is the one depicted in Fig. 27.
- Second case: We start by tiling the $6S + 5M$ basis with a $3S + 3M$ square.
  The different situations to consider are shown in Fig. 28, Fig. 29, and Fig. 30. The only correct configuration is the one depicted in Fig. 30.
- Moreover, any solution requires either a $4S + 3M$ square or a $3S + 3M$ square to be on the $6S + 5M$ basis.

Therefore, Fig. 27 and Fig. 30 describe the only two possibilities to start the tiling of the remaining surface described in Fig. 24. By symmetry, we can complete these partial tilings into the solution described in Fig. 31 (other equivalent solutions are also possible).

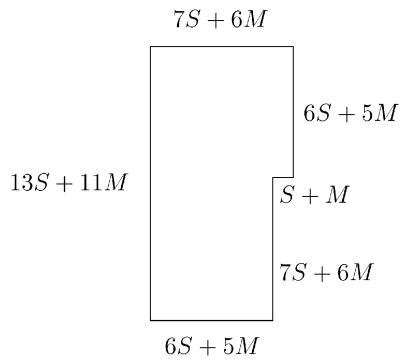


Fig. 24. Remaining surface.

.2.3 Partial onclusion

We have proved that any tiling of the remaining surface (see Fig. 24) is similar to the one depicted in Fig. 31. After using all the large rectangles $A_{x,y}$, there remains two nonadjacent rectangle areas of area $M \times S$ to be tiled. Therefore, we can solve the ASP problem iff we can tile these two areas with the remaining squares, i.e., n squares A_{b_i} of width b_i and $\sum_{i=1}^{n} KS(i)$ Kenyon's squares of width $w(\overline{b_i}, j)$. Since $\min_i b_i > \frac{M}{2}$, we cannot superpose two rectangles A_{b_i} and A_{b_j}, $i \neq j$, on top of each other. Since $\sum_i b_i = 2S$, one can easily check that both $M \times S$ rectangle areas have to be tiled as depicted in Fig. 22. Necessarily the A_{b_i} squares of size $b_i \times b_i$ have to be aligned. Therefore, for each rectangle the sum of the b_i is equal to S. Consequently, our instance $\mathcal{L}$ of the ASP problem has a solution iff there exists a partition of $\{b_1, \ldots, b_n\}$ into two subsets of same sum S.

.2. Final Reduction

To complete the reduction, we have to show that there exists a partition of $\{b_1, \ldots, b_n\}$ into two subsets of same sum iff the original instance $\mathcal{A}$ of the 2-Partition-Equal problem has a solution.

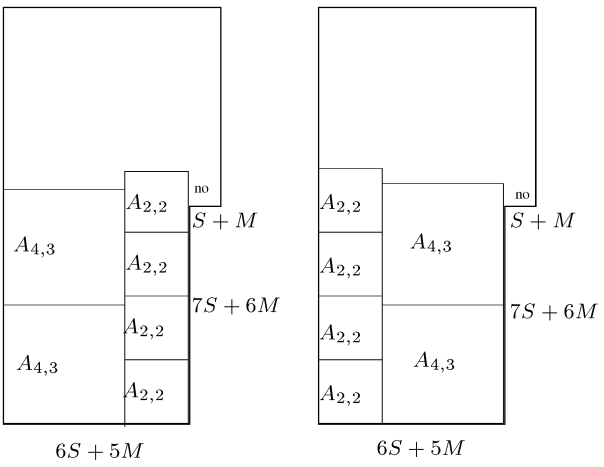


Fig. 25. Some impossible configurations with a $4S + 3M$ square.

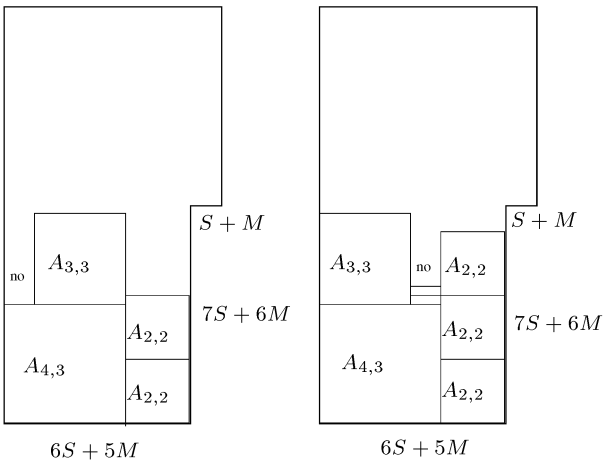


Fig. 26. Some impossible configurations with a $4S+3M$ square.

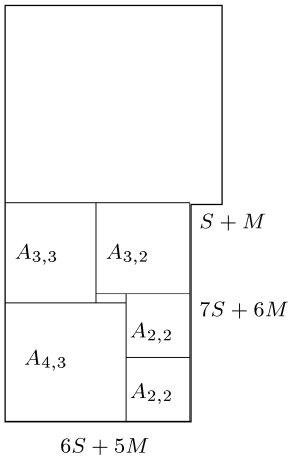


Fig. 27. The only correct configuration with a $4S+3M$ square.

First, suppose that the original instance of the 2-Partition-Equal problem has a solution, i.e. there exists a partition of $\{1,\ldots,n\}$ into two subsets $\mathcal{A}_1$ and $\mathcal{A}_2$ satisfying

$$\sum_{i\in\mathcal{A}_1} a_i = \sum_{i\in\mathcal{A}_2} a_i \text{ and card } (\mathcal{A}_1) = \text{ card } (\mathcal{A}_2).$$

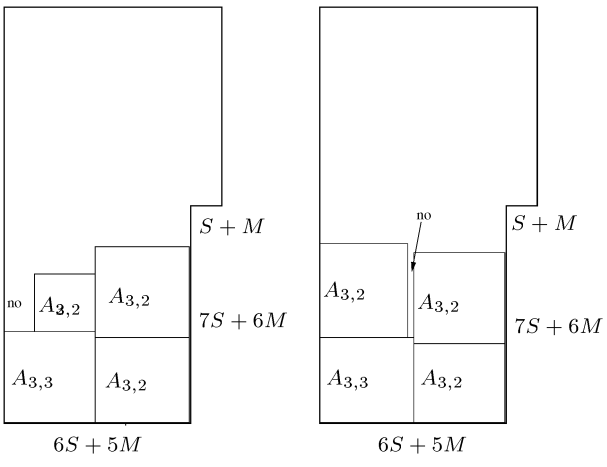


Fig. 28. Some impossible configurations with a $3S+3M$ square.

Recall that $b_i = 2(a_i + 2n\ \text{MAX})$, where $\text{MAX} = \max_i a_i$. Then,

$$\begin{aligned}\sum_{i\in\mathcal{A}_1} b_i &= \sum_{i\in\mathcal{A}_1} a_i + 2n\ \text{MAX card } (\mathcal{A}_1)\\ &= \sum_{i\in\mathcal{A}_2} a_i + 2n\ \text{MAX card } (\mathcal{A}_2)\\ &= \sum_{i\in\mathcal{A}_2} b_i.\end{aligned}$$

Therefore, there exists a suitable partition of $\{b_1,\ldots,b_n\}$.

Conversely, suppose that there exists a partition of $\{1,\ldots,p\}$ into two subsets $\mathcal{A}_1$ and $\mathcal{A}_2$ such that

$$\sum_{i\in\mathcal{A}_1} b_i = \sum_{i\in\mathcal{A}_2} b_i.$$

Thus,

$$\begin{aligned}\sum_{i\in\mathcal{A}_1} a_i &= \sum_{i\in\mathcal{A}_1} b_i - 2n\ \text{MAX card } (\mathcal{A}_1)\\ \sum_{i\in\mathcal{A}_2} a_i &= \sum_{i\in\mathcal{A}_2} b_i - 2n\ \text{MAX card } (\mathcal{A}_2)\\ \sum_{i\in\mathcal{A}_1} a_i - \sum_{i\in\mathcal{A}_2} a_i &= 2n\ \text{MAX card } (\mathcal{A}_2 - \mathcal{A}_1).\end{aligned}$$

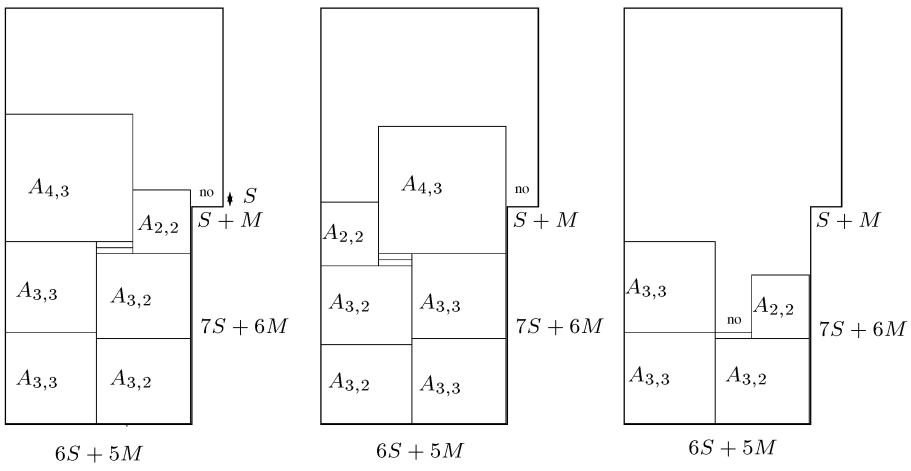


Fig. 29. Some impossible configurations with a $3S+3M$ square.

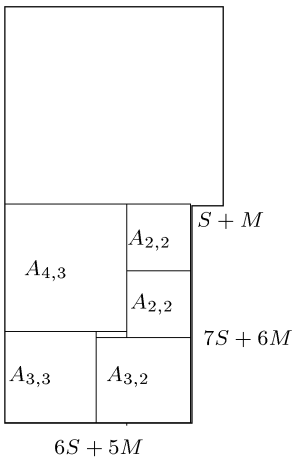


Fig. 30. The only correct configuration with a $3S+3M$ square.

Moreover, since

$$\sum_{a_i \in \mathcal{A}_1} a_i - \sum_{a_i \in \mathcal{A}_2} a_i \leq n \text{ MAX},$$

we obtain

$$\text{card}(\mathcal{A}_1) = \text{ card } (\mathcal{A}_2) \text{ and} \sum_{i \in \mathcal{A}_1} a_i = \sum_{i \in \mathcal{A}_2} a_i.$$

Therefore, the original instance of the 2-Partition-Equal problem has a solution.

A.3 Conciseness of the Transformation

The last element of the proof is the conciseness of the transformation. We have to proof that our instance of the ASP problem has a size polynomial in the size of the original instance of the 2-Partition-Equal problem.

Lemma 3. *Define* $\text{MAX} = \max_i a_i$ *as above and let* $c(\mathcal{A})$ *and* $c(\mathcal{L})$ *denote respectively the encoding of the data* $\mathcal{A}$ *and* $\mathcal{L}$. *Then,*

$$Length(c(\mathcal{L})) = O(Length(c(\mathcal{A}))^3).$$

Proof. We write $f(x) = O(g(x))$ if there exists a constant c such that $f(x) \leq c.g(x)$ for x large enough. Similarly, we write $f(x) = \Omega(g(x))$ if there exists a constant c such that $g(x) \leq c.f(x)$ for x large enough.

For the encoding of the initial instance $\mathcal{A}$, we have $\text{Length}(c(\mathcal{A})) = \Omega(\sum_i \log a_i)$. Since

$$\sum_i \log a_i \geq \log \text{ MAX} + (n-1)\log(\min_i a_i)$$
$$\geq (n-1)\log 2 + \log \text{ MAX},$$

we derive that

$$\text{Length}(c(\mathcal{A})) = \Omega(n + \log \text{ MAX}).$$

For the encoding of the ASP instance $\mathcal{L}$, we have

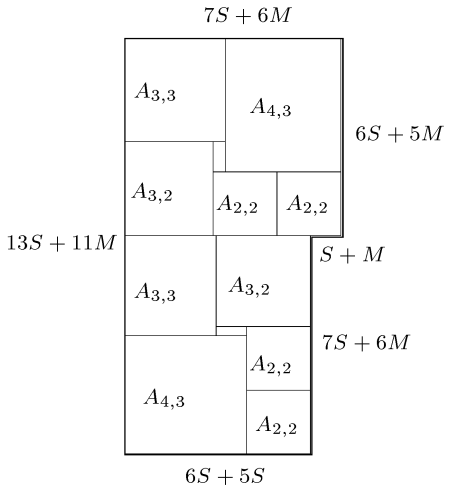


Fig. 31. One possible tiling of the remaining surface.

$$\begin{cases} \log b_i \leq \log((4n+2)\text{ MAX}) = O(\log n + \log \text{ MAX}), \\ \log M = O(\log n + \log \text{ MAX}), \\ \log S = O(n(\log n + \log \text{ MAX}), \\ \sum_i KS(i)\log b_i \leq \sum_i (3 + C\log b_i)\log b_i \\ \qquad = O(n(\log n + \log \text{ MAX})^2), \end{cases}$$

where C is the universal constant given by Kenyon [27]. Therefore,

$$\text{Length}(c(\mathcal{L})) = O(\text{Length}(c(\mathcal{A}))^3).$$

□

This achieves the proof of the NP-completeness of ASP and, therefore, the proof of the NP-completeness of MM-DEC. □

ACKNOWLEDGMENTS

The authors would like to thank the reviewers whose comments and suggestions have greatly improved the presentation of the paper. They would also like to thank Claire Kenyon for her careful reading of the proofs.

REFERENCES

[1] R. Agarwal, F. Gustavson, and M. Zubair, "A High Performance Matrix Multiplication Algorithm on a Distributed-Memory Parallel Computer, Using Overlapped Communication," *IBM J. Research and Development*, vol. 38, no. 6, pp. 673-681, 1994.

[2] H.E. Bal, A. Plaat, T. Kielmann, J. Maassen, R. van Nieuwpoort, and R. Veldema, "Parallel Computing on Wide-Area Clusters: The Albatross Project" *Proc. Extreme Linux Workshop*, pp. 20-24, 1999.

[3] O. Beaumont, V. Boudet, A. Legrand, F. Rastello, and Y. Robert, "Heterogeneity Considered Harmful to Algorithm Designers," Technical Report RR-2000-24, LIP, ENS Lyon, June 2000. Also available at www. ens-lyon. fr/LIP/.

[4] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert, "Matrix-Matrix Multiplication on Heterogeneous Platforms," Technical Report RR-2000-02, LIP, ENS Lyon, Jan. 2000.

[5] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert, "Matrix-Matrix Multiplication on Heterogeneous Platforms," *Proc. 2000 Int'l Conf. Parallel Processing (ICPP 2000)*, 2000.

[6] F. Berman, "High-Performance Schedulers," *The Grid: Blueprint for a New Computing Infrastructure*, I. Foster and C. Kesselman, eds., Morgan-Kaufmann, pp. 279-309, 1999.

[7] L.S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley, *ScaLAPACK Users' Guide.* SIAM, 1997.

[8] V. Boudet, F. Rastello, and Y. Robert, "Algorithmic Issues for (Distributed) Heterogeneous Computing Platforms," *Proc. Cluster Computing Technologies, Environments, and Applications (CC-TEA '99),* R. Buyya and T. Cortes, eds., 1999.

[9] P. Boulet, J. Dongarra, F. Rastello, Y. Robert, and F. Vivien, "Algorithmic Issues on Heterogeneous Computing Platforms," *Proc. Clusters and Computational Grids Workshop,* 1998.

[10] P. Boulet, J. Dongarra, F. Rastello, Y. Robert, and F. Vivien, "Algorithmic Issues on Heterogeneous Computing Platforms," *Parallel Processing Letters,* vol. 9, no. 2, pp. 197-213, 1999.

[11] H. Casanova and J. Dongarra, "Netsolve: A Network Server for Solving Computational Science Problems," *Int'l J. Supercomputer Applications and High Performance Computing,* vol. 11, no. 3, pp. 212-223, 1997.

[12] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley, "ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers—Design Issues and Performance," *Computer Physics Comm.,* vol. 97. pp. 1-15, 1996.

[13] M. Cierniak, M.J. Zaki, and W. Li, "Customized Dynamic Load Balancing for a Network of Workstations," *J. Parallel and Distributed Computing,* vol. 43, pp. 156-162, 1997.

[14] M. Cierniak, M.J. Zaki, and W. Li, "Scheduling Algorithms for Heterogeneous Network of Workstations," *Computer J.,* vol. 40, no. 6, pp. 356-372, 1997.

[15] P.E. Crandall and M.J. uinn, "Block Data Decomposition for Data-Parallel Programming on a Heterogeneous Workstation Network," *Proc. Second Int'l Symp. High Performance Distributed Computing,* pp. 42-49, 1993.

[16] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, *Complexity and Approximation,* Springer, 1999.

[17] J.J. Dongarra and D.W. Walker, "Software Libraries for Linear Algebra Computations on High Performance Computers," *SIAM Rev.,* vol. 37, no. 2, pp. 151-180, 1995.

[18] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *Int'l J. Supercomputer Applications,* vol. 11, no. 2, pp. 115-128, 1997.

[19] *The Grid: Blueprint for a New Computing Infrastructure.* I. Foster and C. Kesselman, eds., Morgan-Kaufmann, 1999.

[20] G. Fox, S. Otto, and A. Hey, "Matrix Algorithms on a Hypercube i: Matrix Multiplication," *Parallel Computing,* vol. 3, pp. 17-31, 1987.

[21] M.R. Garey and D.S. Johnson, *Computers and Intractability, a Guide to the Theory of NP-Completeness.* W.H. Freeman and Co., 1991.

[22] T.F. Gonzalez and S. Zheng, "Improved Bounds for Rectangular and Guilhotine Partitions," *J. Symbolic Computation,* vol. 7, pp. 591-610, 1989.

[23] T.F. Gonzalez and S. Zheng, "Approximation Algorithm for Partitioning a Rectangle with Interior Points," *Algorithmica,* vol. 5, pp. 11-42, 1990.

[24] A.S. Grimshaw and W.A. Wulf, "The Legion Vision of a Worldwide Virtual Computer," *Comm. ACM,* vol. 40, no. 1, pp. 39-45, 1997.

[25] M. Kaddoura, S. Ranka, and A. Wang, "Array Decomposition for Nonuniform Computational Environments," *J. Parallel and Distributed Computing,* vol. 36, pp. 91-105, 1996.

[26] A. Kalinov and A. Lastovetsky, "Heterogeneous Distribution of Computations While Solving Linear Algebra Problems on Networks of Heterogeneous Computers," *Proc. High Performance Computing and Networking Europe 1999,* P. Sloot, M. Bubak, A. Hoekstra, and B. Hertzberger, eds., pp. 191-200, 1999.

[27] R.W. Kenyon, "Tiling a Rectangle with the Fewest Squares," *J. Combination Theory A,* vol. 76, pp. 272-291, 1996.

[28] S. Khanna, S. Muthukrishnan, and M. Paterson, "On Approximating Rectangle Tiling and Packing," *Proc. Ninth Ann. ACM-SIAM Symp. Discrete Algorithms,* pp. 384-393, 1998.

[29] T.Y. Kong, D.M. Mount, and W. Roscoe, "The Decomposition of a Rectangle into Rectangles of Minimal Perimeter," *SIAM J. Computing,* vol. 17, no. 6, pp. 1215-1231, 1988.

[30] T.Y. Kong, D.M. Mount, and M. Wermann, "The Decomposition of a Square into Rectangles of Minimal Perimeter," *Discrete Applied Mathematics,* vol. 16, pp. 239-243, 1987.

[31] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing.* Benjamin/Cummings Inc., 1994.

[32] A. Legrand, "Algorithmique Parallele: Environnements Hétérogenes et Non-Dédiés," Master's thesis, École Normale Supérieure de Lyon, June 2000. Also available at www.ens-lyon.fr/~yrobert.

[33] A. Lingas, R.Y. Pinter, R.L. Rivest, and A. Shamir, "Minimum Edge Length Partitioning of Rectilinear Polygons," *Proc. 20th Ann. Allerton Conf. Comm., Control and Computing,* 1982.

[34] C.D. Polychronopoulos, "Compiler Optimization for Enhancing Parallelism and Their Impact on Architecture Design," *IEEE Trans. Computers,* vol. 37, no. 8, pp. 991-1004, Aug. 1988.

[35] M. Snir, S.W. Otto, S. Huss-Lederman, D.W. Walker, and J. Dongarra, *MPI: The Complete Reference.* MIT Press, 1996.

Olivier Beaumont received the PhD degree from the Université de Rennes in January 1999. He is currently an associate professor in the Computer Science Laboratory LIP at École Normale Supérieure de Lyon. His main research interests are parallel algorithms on distributed memory architectures.

incent Boudet is currently a PhD student in the Computer Science Laboratory LIP at École Normale Supérieure de Lyon. He is mainly interested in algorithm design and in compilation-parallelization techniques for distributed memory architectures.

Fabrice Rastello received the PhD degree from École Normale Supérieure de Lyon in September 2000. He is currently working as an engineer for ST Microelectronics in Grenoble. His main research interests are parallel algorithms for distributed memory architectures and automatic compilation/parallelization techniques.

Yves Robert received the PhD degree from the Institut National Polytechnique de Grenoble in January 1986. He is currently a full professor in the Computer Science Laboratory LIP at École Normale Supérieure de Lyon. He is the author of three books, more than 60 papers published in international journals, and more than 70 papers published in international conferences. His main research interests are parallel algorithms for distributed memory architectures and automatic compilation/parallelization techniques. He is a member of the ACM and of the IEEE.

Algorithmica (2002) 34: 217–239
DOI: 10.1007/s00453-002-0962-9

Algorithmica

Partitioning a Square into Rectangles: NP-Completeness and Approximation Algorithms

Olivier Beaumont,[1] Vincent Boudet,[1] Fabrice Rastello,[1] and Yves Robert[1]

Abstract. In this paper we deal with two geometric problems arising from heterogeneous parallel computing: how to partition the unit square into p rectangles of given area $s_1, s_2, \ldots, s_p$ (such that $\sum_{i=1}^{p} s_i = 1$), so as to minimize either (i) the sum of the p perimeters of the rectangles or (ii) the largest perimeter of the p rectangles? For both problems, we prove NP-completeness and we introduce a $\frac{7}{4}$-approximation algorithm for (i) and a $(2/\sqrt{3})$-approximation algorithm for (ii).

Key Words. NP-completeness, Approximation algorithms, Geometric problems, Heterogeneous resources, Parallel computing.

1. Introduction. In this paper we deal with two simple geometric problems: how to partition the unit square into p rectangles of given area $s_1, s_2, \ldots, s_p$ (such that $\sum_{i=1}^{p} s_i = 1$), so as to minimize

- either the sum of the p half-perimeters of the rectangles,
- or the largest half-perimeter of the p rectangles?

Note that there always exist solutions to these problems: e.g. tile the unit square into p horizontal slices of height $s_1, s_2, \ldots, s_p$. The difficulty is to minimize the objective function.

Consider the following example with $p = 5$ rectangles $R_1, \ldots, R_5$ of areas $s_1 = 0.36$, $s_2 = 0.25$, $s_3 = s_4 = s_5 = 0.13$. A possible partition is shown in Figure 1. The size of each rectangle is the following: $0.61 \times \frac{36}{61}$ for R_1, $0.61 \times \frac{25}{61}$ for R_2 and $0.39 \times \frac{1}{3}$ for R_3, R_4 and R_5. The maximum half-perimeter is that of R_1, approximately 1.2002, which is very close to the absolute lower bound 1.2 obtained when the largest rectangle is a square (this is not achievable in this example). As for the second objective function, we compute that the sum of the half-perimeters is 4.39, while the absolute lower bound is $\sum_{i=1}^{p} 2\sqrt{s_i} \approx 4.36$ (obtained when all rectangles are squares, which is not achievable in this example either). Hence the partition turns out to be very satisfactory for both objective functions. The geometric interpretation for the sum of the half-perimeters is nice: it is the length of the lines drawn to make the partition, plus 2 corresponding to the right and bottom edge of the unit square.

The main objective of the paper is the design of approximation algorithms for both optimization problems (which turn out to be NP-complete). Beforehand, we explain the initial motivation for this work, which arises from minimizing communications in

[1] LIP, UMR CNRS-ENS Lyon-INRIA 5668, Ecole Normale Supérieure de Lyon, F-69364 Lyon Cedex 07, France. Contact: Yves.Robert@ens-lyon.fr.

Received February 5, 2001; revised August 22, 2001. Communicated by J.-D. Boissonnat.
Online publication June 21, 2002.

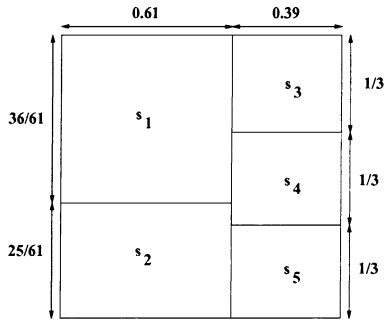


Fig. 1. A simple example with five rectangles.

the design of parallel algorithms targeted to heterogeneous platforms. The rest of the paper is organized as follows. In Section 2 we explain the motivation from heterogeneous parallel computing. In Section 3 we formally state the optimization problems PERI-SUM (minimize the sum of the perimeters of the rectangles) and PERI-MAX (minimize the largest perimeter), and we establish the NP-completeness of PERI-MAX (the proof of the NP-completeness of PERI-SUM is available in [3]). Section 4 is devoted to the design of approximation algorithms for PERI-SUM; Section 5 is its counterpart for PERI-MAX. In Section 6 we briefly survey related optimization problems. We give some final remarks and conclusions in Section 7.

2. Problem Motivation. The motivation for this work is the design of parallel Matrix Multiplication (MM for short) algorithms targeted to heterogeneous platforms, such as heterogeneous clusters of workstations, or collections of such clusters.

Parallel MM algorithms work as follows: let $C = A \times B$ be the product to be computed, where A and B are square matrices of size $n \times n$. First, granularity is increased: matrix blocks rather than elementary matrix coefficients are allocated to processors, as in the ScaLAPACK library [4]. Hence, each "element" in A, B and C is a square $r \times r$ block, and the unit of computation is the updating of one block, i.e. a matrix–matrix multiplication of size r. Assume there are p processors. The three matrices A, B and C are partitioned into p (superposed) rectangles. There is a one-to-one mapping between these rectangles and the processors. Each processor is responsible for updating its rectangle: at each step, one pivot column and one pivot row are communicated to all processors, and independent updates take place; more precisely, each processor updates each block in its rectangle with one block from the pivot row and one block from the column row, as illustrated in Figure 2.

Using different-speed processors, we want to balance the computing load so that each processor receives an amount of work in accordance to its computing power. Because all C blocks require the same amount of arithmetic operations, each processor executes an amount of work which is proportional to the number of blocks that are allocated to it, hence proportional to the area of its rectangle. In Figure 2 we have 13 different-speed computing resources. We let s_i be the fraction of the total computing power represented by processor P_i, $1 \leq i \leq p$. Normalizing processor speeds, we have $\sum_{i=1}^{p} s_i = 1$. Normalizing the computing workload accordingly, we have to tile the unit square into

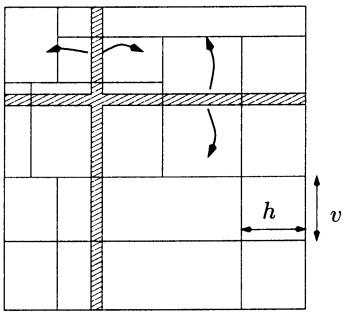


Fig. 2. The MM algorithm on a heterogeneous platform.

p rectangles R_i of prescribed area s_i, $1 \leq i \leq p$. The question is: how to compute the *shape* of these p rectangles so as to minimize the total execution time?

Let $h_i \times v_i$ be the size of rectangle R_i, where $h_i v_i = s_i$. At each step of the MM algorithm, communications take place between processors: the total volume of data exchanged is proportional to the sum $\hat{C} = \sum_{i=1}^{p} (h_i + v_i)$ of the half-perimeters of the p rectangles R_i. In fact, this is not exactly true: because the pivot row and columns are not sent to the processors that own them, we should subtract 2 from $\hat{C}$, 1 for the horizontal communications and 1 for the vertical ones. Since minimizing $\hat{C}$ or $\hat{C} - 2$ is equivalent, we keep the value of $\hat{C}$ as stated. Minimizing $\hat{C}$ seems to be a very natural goal, because it represents the total volume of communications. For instance it is natural to assume that communications will be mostly sequential on a heterogeneous network of workstations where processors are linked by a simple Ethernet network; also, there will be little or no computation/communication overlap on such a platform. In that context, minimizing the total communication volume is the main objective.

Conversely, some communications can occur in parallel, if the computing resources are linked through a dedicated high-speed network, and if parallel communication links are provided. In that context, we may want to minimize the maximal amount of communications to be performed by each processor, so that the objective function becomes $\hat{M} = \max_{1 \leq i \leq p} (h_i + v_i)$.

Once a solution to either optimization problem has been found, we derive the allocation of data elements to processor P_i by rounding up the values $n \times h_i$ and $n \times v_i$. Finally, note that both optimization problems have a wide potential applicability. Forgetting about MM algorithms, consider the implementation of any application (such as a finite-difference scheme) where heterogeneous processors communicate boundary elements at each step (the communication scheme need not be nearest-neighbor, it can be anything): minimizing the total communication volume, or the maximal amount of communications performed by one processor, while load-balancing the work, amounts to solving exactly the same optimization problems.

3. NP-Completeness

3.1. *Problem Formulation.* We formally state both optimization problems. We have to determine p rectangles R_i, of prescribed area s_i, $1 \leq i \leq p$, where $\sum_{i=1}^{p} s_i = 1$. The

shape of each R_i is the degree of freedom: we want to tile the unit square so as to solve the following optimization problems:

DEFINITION 1 (PERI-SUM(s)). Given p real positive numbers $s_1, \ldots, s_p$ s.t. $\sum_{i=1}^{p} s_i = 1$, find a partition of the unit square into p rectangles R_i of area s_i and of size $h_i \times v_i$, so that $\hat{C} = \sum_{i=1}^{p}(h_i + v_i)$ is minimized.

DEFINITION 2 (PERI-MAX(s)). Given p real positive numbers $s_1, \ldots, s_p$ s.t. $\sum_{i=1}^{p} s_i = 1$, find a partition of the unit square into p rectangles R_i of area s_i and of size $h_i \times v_i$, so that $\hat{M} = \max_{1 \le i \le p}(h_i + v_i)$ is minimized.

There is an obvious lower bound for PERI-SUM(s) and for PERI-MAX(s):

LEMMA 1. *For all solutions of PERI-SUM(s), $\hat{C} \ge 2\sum_{i=1}^{p} \sqrt{s_i}$. For all solutions of PERI-MAX(s), $\hat{M} \ge 2\max_{1 \le i \le p} \sqrt{s_i}$.*

PROOF. The half-perimeter of each rectangle R_i will always be larger that $2\sqrt{s_i}$, the value when it is a square. Of course, tiling the unit square into p squares of area s_i is not always possible, so the lower bound for PERI-SUM(s) is not always tight. The same observation holds for PERI-MAX(s), as shown by the example in Section 1. □

The decision problem associated to the optimization problem PERI-SUM has been shown to be NP-complete in [3].

3.2. *PERI-MAX(s).* The decision problem associated to the optimization problem PERI-MAX is the following:

DEFINITION 3 (PERI-MAX(s, K)). Given p real positive numbers $s_1, \ldots, s_p$ s.t. $\sum_{i=1}^{p} s_i = 1$ and a positive real bound K, is there a partition of the unit square into p rectangles R_i of area s_i and of size $h_i \times v_i$, so that $\max_{1 \le i \le p}(h_i + v_i) \le K$?

THEOREM 1. *PERI-MAX(s,K) is NP-complete.*

PROOF. The proof uses the following reduction:

LEMMA 2.

$$2P\text{-}0\text{-}4 \le_P MSP \le_P PERI\text{-}MAX,$$

where MSP and 2P-0-4 are defined as follows:

DEFINITION 4 (2-Partition-0-4 (2P-0-4)). Given a set of $p+2$ integers $\mathcal{A} = \{a_1, \ldots, a_p, a_{p+1} = 2, a_{p+2} = 2\}$ such that $\forall i \le p, \quad a_i > 4$, and $a_i = 0 \bmod 4$, is there a partition of $\{1, \ldots, p+2\}$ into two subsets $\mathcal{A}_1$ and $\mathcal{A}_2$ such that

$$\sum_{i \in \mathcal{A}_1} a_i = \sum_{i \in \mathcal{A}_2} a_i \quad \text{or} \quad \sum_{i \in \mathcal{A}_1} a_i = \sum_{i \in \mathcal{A}_2} a_i + 4?$$

DEFINITION 5 (Max-Square-Partition (MSP)). Given a set $\mathcal{A} = \{s_1, \ldots, s_p\}$ of p real positive numbers such that $\sum_{i=1}^{p} s_i = 1$ and $s_1 \geq s_2 \geq \cdots \geq s_p$, is there a partition of the unit square into p rectangles R_i of area s_i such that R_1 is a square and the half-perimeter of other rectangles is not larger than $2\sqrt{s_1}$?

Since 2P-0-4 is known to be NP-Complete (trivial reduction from 2-Partition [6]), Lemma 2 will complete the proof of Theorem 1.

3.2.1. *Reduction*: *MSP* $\leq_P$ *PERI-MAX*(s, K). We start by proving the easy part of Lemma 2, i.e. MSP $\leq_P$ PERI-MAX(s, K). Let $\mathcal{A} = \{s_1, \ldots, s_p\}$ be a set of p real positive numbers s.t. $\sum_{i=1}^{p} s_i = 1$ and $s_1 \geq s_2 \geq \cdots \geq s_p$. Solving MSP is equivalent to solving PERI-MAX(s, K) with

$$K = 2\sqrt{s_1}$$

and, therefore,

$$\text{MSP} \leq_P \text{PERI-MAX}.$$

3.2.2. *Reduction*: 2*P*-0-4 $\leq_P$ *MSP*. In this section we consider an arbitrary instance of the 2-Partition-0-4 problem, i.e., a set $\mathcal{A} = \{a_1, \ldots, a_n, a_{n+1} = 2, a_{n+2} = 2\}$ such that $\forall i \leq p,\ \ a_i > 4$ and $a_i = 0 \bmod 4$). We have to polynomially transform this instance into an instance of the MSP problem which has a solution iff the original instance of 2-Partition-0-4 has one solution. Let

$$S = \frac{\sum_{1 \leq i \leq n} a_i}{4}.$$

We build the following instance of the (scaled) MSP problem (MSP$(a_1, \ldots, a_n)$): is there a partition of the $(S + 2) \times (S + 2)$ square into $n + 3$ rectangles $R_S, R_1, \ldots, R_{n+2}$ of respective areas

$$\begin{aligned} R_S: &\quad A_S = S^2, \\ R_i: &\quad A_i = a_i \qquad (\forall i, 1 \leq i \leq n), \\ R_{n+1}: &\quad A_{n+1} = 2, \\ R_{n+2}: &\quad A_{n+2} = 2, \end{aligned}$$

where the rectangle R_S of area A_S is a square and the half-perimeter of other rectangles R_i is less than $2S$?

The general position of the largest square is depicted in Figure 3. We partition the set of the remaining rectangles into three disjoints sets:

- S_0: the rectangles whose intersection with Σ_0 has a non-zero area.
- S_1: the rectangles whose intersection with Σ_1 has a non-zero area.
- S_{rem}: the rest of the rectangles.

Since the area of Σ_{rem} is equal to 4, we can easily prove the following lemma:

LEMMA 3. $\forall i \leq n$, R_i *belongs either to* S_0 *or to* S_1 $(a_i > 4)$. *Moreover, we have* $|\sum_{R_i \in S_0} A_i - \sum_{R_i \in S_1} A_i| \leq 4$, *i.e.* $|\sum_{R_i \in S_0} A_i - \sum_{R_i \in S_1} A_i| \in \{0, 2, 4\}$.

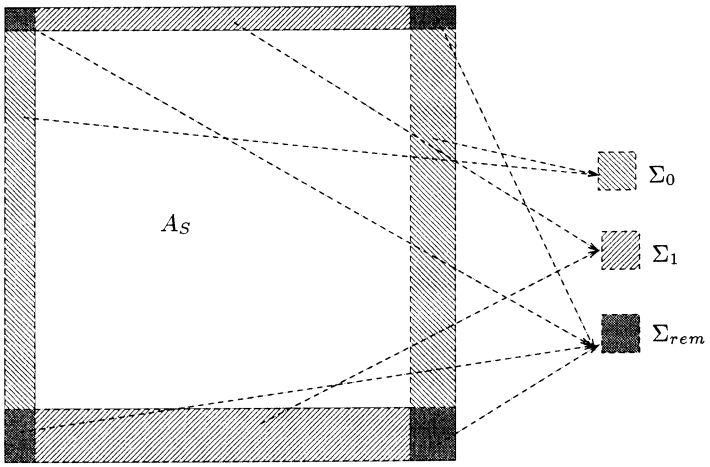


Fig. 3. General position of the largest square.

Without lost of generality, we suppose in what follows that $\sum_{R_i \in S_0} A_i \geq \sum_{R_i \in S_1} A_i$. Three different cases are to be considered according to the value of $\sum_{R_i \in S_0} A_i - \sum_{R_i \in S_1} A_i$:

- $\sum_{R_i \in S_0} A_i - \sum_{R_i \in S_1} A_i = 0$. In this case either $\exists i$, $R_{n+1} \in S_i$ and $R_{n+2} \in S_{1-i}$, or both R_{n+1} and R_{n+2} belong to S_{rem}. In the first case S_0 and S_1 represent a partition of $\mathcal{A} = \{a_1, \ldots, a_n, a_{n+1} = 2, a_{n+2} = 2\}$ into two subsets of same sum. In the second case $S_0 \cup R_{n+1}$ and $S_1 \cup R_{n+2}$ represent a partition of $\mathcal{A}$ into two subsets of same sum.
- $\sum_{R_i \in S_0} A_i - \sum_{R_i \in S_1} A_i = 2$. In this case exactly one rectangle out of R_{n+1} and R_{n+2} belongs to S_0 or S_1, and the other one belongs to S_{rem}. We suppose, without loss of generality, that $R_{n+1} \in S_i$. Then S_0 and $S_1 \cup R_{n+2}$ represent a partition of $\mathcal{A}$ into two subsets of same sum.
- $\sum_{R_i \in S_0} A_i - \sum_{R_i \in S_1} A_i = 4$. Again, in this case, either $\exists i$, $R_{n+1} \in S_i$ and $R_{n+2} \in S_{1-i}$, or both R_{n+1} and R_{n+2} belong to S_{rem}. In the first case S_0 and S_1 represent a partition of $\mathcal{A}$ into two subsets whose sums differ by 4. In the second case S_0 and $S_1 \cup R_{n+1} \cup R_{n+2}$ represent a partition of $\mathcal{A}$ into two subsets of the same sum.

Therefore, there exists a partition of $\mathcal{A} = \{a_1, \ldots, a_n, a_{n+1} = 2, a_{n+2} = 2\}$ into two subsets whose sums differ by 0 or 4 if our instance of the MSP problem has a solution. To complete the reduction, we have to show that our instance of the MSP problem has a solution if there exists a partition of $\mathcal{A} = \{a_1, \ldots, a_n, a_{n+1} = 2, a_{n+2} = 2\}$ into two subsets whose sums differ by 0 or 4.

- Suppose there is a partition of $\{1, \ldots, n+2\}$ into two subsets $\mathcal{A}_1$ and $\mathcal{A}_2$ such that

$$\sum_{i \in \mathcal{A}_1} a_i = \sum_{i \in \mathcal{A}_2} a_i + 4.$$

  We define $S_0 = \bigcup_{i \in \mathcal{A}_1} R_i$ where R_i denotes a $2 \times a_i/2$ rectangle, and $S_1 = \bigcup_{i \in \mathcal{A}_2} R_i$, where R_i denotes an $a_i/2 \times 2$ rectangle. We tile the $(S+2) \times (S+2)$ area as indicated

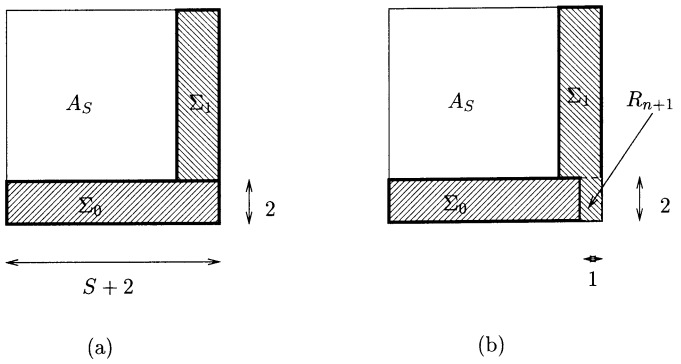


Fig. 4. Tiling the square using the MSP instance.

in Figure 4(a). Since it is possible to tile both Σ_0 and Σ_1 with S_0 and S_1, it is possible to solve the MSP problem.

- Suppose there is a partition of $\{1, \ldots, n+2\}$ into two subsets $\mathcal{A}_1$ and $\mathcal{A}_2$ such that

$$\sum_{i \in \mathcal{A}_1} a_i = \sum_{i \in \mathcal{A}_2} a_i.$$

We define $S_0 = \bigcup_{i \in \mathcal{A}_1} R_i$ where R_i denotes a $2 \times a_i/2$ rectangle, and $S_1 = \bigcup_{i \in \mathcal{A}_2} R_i$, where R_i denotes an $a_i/2 \times 2$ rectangle. Suppose, without loss of generality that R_{n+1} belongs to S_1. Then we tile the $(S+2) \times (S+2)$ area as indicated in Figure 4(b). Hence, since it is possible to tile both Σ_0 and Σ_1 with S_0 and S_1 as depicted in Figure 4(b), it is possible to solve the MSP problem.

Therefore, our instance of the MSP problem has a solution iff there exists a partition of $\mathcal{A} = \{a_1, \ldots, a_n, a_{n+1} = 2, a_{n+2} = 2\}$ into two subsets whose sums differ by 0 or 4. This achieves the proof of the NP-completeness of MSP, and therefore of the NP-completeness of PERI-MAX. □

4. Approximation Algorithms for PERI-SUM. There are several "natural" approximation algorithms to approximate PERI-SUM. However, proving approximation bounds turns out to be very technical. We start in Section 4.1 with a column-based approximation algorithm, very simple to implement, and which appears very efficient through extensive experimental comparisons. However, we have not been able to give a tight approximation bound: the bound of Section 4.1.3 depends on the relative size of the rectangles to be used in the tiling. In Section 4.2 we move to a recursive approximation algorithm, much more complicated to describe, but for which a nice approximation bound is provided.

4.1. *Column-Based Approximation Algorithm*

4.1.1. *Description.* Since PERI-SUM(s) is NP-complete, we consider the more constrained problem COL-PERI-SUM(s) where we impose that the tiling is made up with

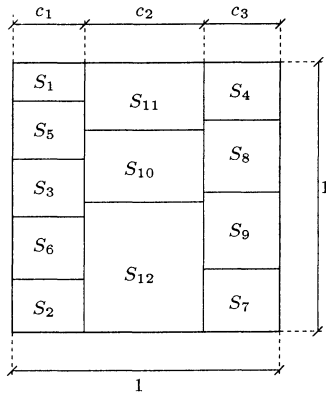


Fig. 5. Column-based partitioning of the unit square: $\mathcal{C} = 3$, $k_1 = 5$, $k_2 = 3$ and $k_3 = 4$.

processor columns, as illustrated in Figure 5. In other words, COL-PERI-SUM(s) is the restriction of PERI-SUM(s) to column-based partitions. In this section we give a polynomial solution to COL-PERI-SUM(s), which will be used as an approximation algorithm for PERI-SUM(s).

Framework. We describe the COL-PERI-SUM(s) problem more formally: we aim at tiling the unit square into $\mathcal{C}$ columns (where $\mathcal{C}$ is yet to be determined) of width $c_1, \ldots, c_{\mathcal{C}}$. Each column C_i is partitioned itself into k_i blocks (to be determined too) of respective area $s_{\sigma(i,1)}, \ldots, s_{\sigma(i,k_i)}$. Of course, the final partitioning has $\sum_{i=1}^{\mathcal{C}} k_i = p$ rectangles, and all the areas $s_1, \ldots, s_p$ are represented once and only once. The goal is to build such a partitioning, subject to the minimization of the sum of the rectangle perimeters.

Algorithm. The main points of the column-based tiling are the following:

1. Re-index the areas $s_1, \ldots, s_p$ such that $s_1 \leq s_2 \leq \cdots \leq s_p$.
2. Iteratively build the function $f_{\mathcal{C}}$, by incrementing the value of $\mathcal{C}$ from 1 to the desired value. For $q \in \{1, \ldots, p\}$, $f_{\mathcal{C}}(q)$ represents the total perimeter of an optimal column-based partitioning of a rectangle of height 1 and width $\left(\sum_{i=1}^{q} s_i\right) \times 1$ into q rectangles of respective area $s_1, \ldots, s_q$, using $\mathcal{C}$ columns.

In the Appendix we provide pseudo-codes for the algorithm together with a toy example.

The column-based partitioning is a simple dynamic programming algorithm. Since it is possible to reduce the search to sorted sequences $s_1 \leq s_2 \leq \cdots \leq s_p$ (see Section 4.1.1), the total perimeter $f_{\mathcal{C}}(q)$ of the optimal partitioning of a $\sum_1^q s_i \times 1$ rectangle into q rectangles with areas $s_1, s_2, \ldots, s_q$ using $\mathcal{C}$ columns can be obtained with the values $f_{\mathcal{C}-1}(q')$, $\forall 1 \leq q' \leq q$, i.e. the perimeters of the optimal partitioning with only

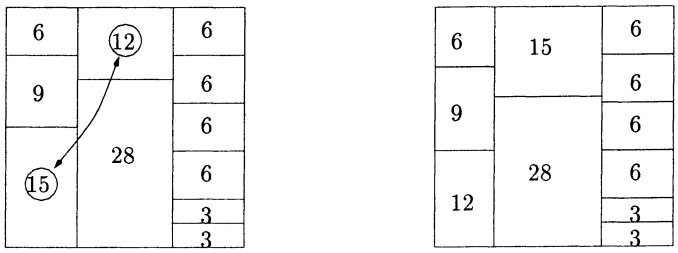


Fig. 6. Two partitionings of the same problem instance. The right one is well-ordered while the left one is not.

$\mathcal{C} - 1$ columns and less rectangles. More precisely,

$$f_{\mathcal{C}}(q) = \min_{q'} \left(1 + \left(\sum_{q'<i\leq q} s_i\right) \times (q - q') + f_{\mathcal{C}-1}(q')\right)$$
$$= 1 + \left(\sum_{r<i\leq q} s_i\right) \times (q - r) + f_{\mathcal{C}-1}(r).$$

Thus, $f_{\mathcal{C}}(q)$ can be computed in linear time (using the values $f_{\mathcal{C}-1}(1), f_{\mathcal{C}-1}(2), \ldots, f_{\mathcal{C}-1}(q)$), and the overall cost of the algorithm (the computation of $f_{\mathcal{C}}(q)$, $\forall 1 \leq q \leq p$, $\forall 1 \leq \mathcal{C} \leq q$) is of order $O(p^3)$. This cost can be reduced to $O(p^2 \log(p)$ using binary search, since $f_{\mathcal{C}}(q)$ is a convex function with respect to q.

Optimality. This algorithm provides the optimal column-based partitioning. To prove the optimality of the algorithm, we show that the optimum solution can be achieved with a *well-ordered partitioning*: a partitioning is said to be well-ordered if for every pair of columns $\mathcal{C}_i$ and $\mathcal{C}_j$, either all the elements of $\mathcal{C}_i$ are smaller than (or equal to) all the elements of $\mathcal{C}_j$, or the other way round. See Figure 6 for an illustration.

We start from a given partitioning, made up of, say, $\mathcal{C}$ columns of size $k_1 \geq k_2 \geq \cdots \geq k_{\mathcal{C}}$. Suppose for convenience that $s_1 \leq s_2 \leq \cdots \leq s_p$; τ is a permutation of $\{1, 2, \ldots, p\}$ such that the ith column of this partitioning contains the rectangles $s_{\tau(d+1)}, \ldots, s_{\tau(d+k_i)}$ with $d = k_1 + k_2 + \cdots + k_{i-1}$. Now, recall that the cost of column C_i is $1 + k_i \times \sum_{j=d+1}^{d+k_{i-1}} s_{\tau(j)}$. Hence, the total "perimeter" is

$$\begin{aligned}
\mathcal{C} &+ k_1 s_{\tau(1)} + k_1 s_{\tau(2)} + \cdots + k_1 s_{\tau(k_1)} \\
&+ k_2 s_{\tau(k_1+1)} + k_2 s_{\tau(k_1+2)} + \cdots + k_2 s_{\tau(k_1+k_2)} \\
&+ \cdots \\
&+ k_i s_{\tau(k_1+\cdots+k_{i-1}+1)} + k_i s_{\tau(k_1+\cdots+k_{i-1}+2)} + \cdots + k_i s_{\tau(k_1+\cdots+k_i)}.
\end{aligned}$$

Since $k_1 \geq k_2 \geq \cdots \geq k_{\mathcal{C}}$, this expression is minimized for $\tau = \textit{Identity}$, which corresponds to a "well-ordered" partitioning. Hence, for each partitioning, there exists a

corresponding better or equivalent partitioning that is "well-ordered." This achieves the proof of correctness.

4.1.2. *Experimental Comparison with the Lower Bound.* As shown in Section 3, a lower bound for the sum $\hat{C}$ of the half-perimeters is twice the sum of the square roots of the areas, i.e. $LB = 2\sum_{i=1}^{p}\sqrt{s_i}$. Of course this bound cannot always be met: consider an instance of PERI-SUM(s) with only two rectangles, $s_1 = 1 - \varepsilon$ and $s_2 = \varepsilon$, where $\varepsilon > 0$ is an arbitrarily small number. Partitioning into two rectangles requires drawing a line of length 1, hence $\hat{C} = 3$. However, $LB = 2\sqrt{1-\varepsilon} + \sqrt{\varepsilon} > 2$ can be arbitrarily close to 2.

In this section we experimentally compare, using a large number of random tests, the value $\hat{C}$ given by our partitioning against the absolute lower bound LB. Figure 7 represents two curves for a number of processors varying from 1 to 40. The top curve corresponds to the mean value of the ratio $\hat{C}/LB$ while the bottom curve gives the minimum values of this ratio. We see that in average, the optimal column-based tiling given by our algorithm gives a solution that is "almost" optimal, so that we can be satisfied with the results for all practical purposes.

4.1.3. *Theoretical Comparison with the Lower Bound.* In this section we prove that the column-based partitioning is a good approximation, especially when the ratio

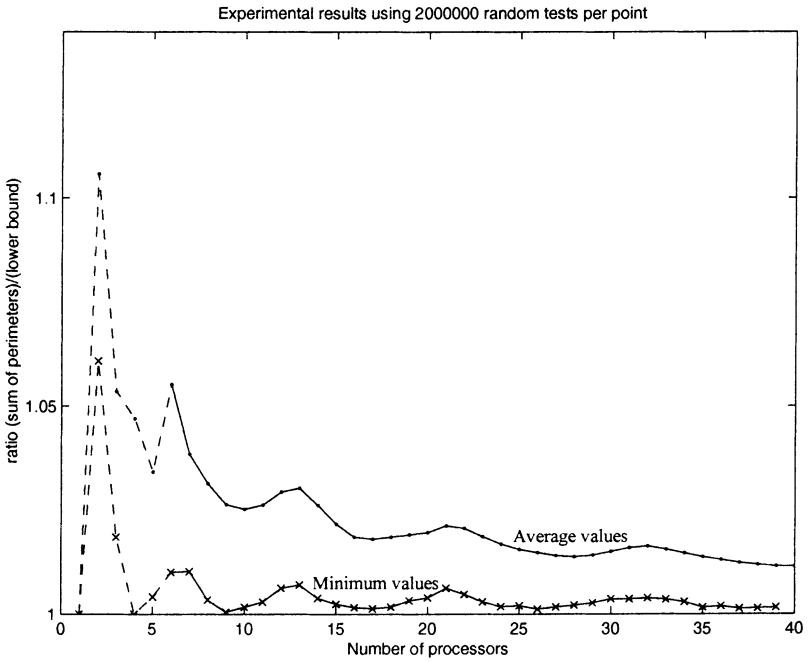


Fig. 7. For each number of processors (varying from 1 to 40), 2,000,000 values for the s_i have been generated. For each case, we compute the ratio of the sum $\hat{C}$ of the half-perimeters of our partitioning over the absolute lower bound LB. The worst case is a constant value equal to 1.5. The average and best cases are reported in the two curves.

between $\max s_i$ and $\min s_i$ is small:

PROPOSITION 1. *Let $r = \max s_i / \min s_i$, let $\hat{C}$ denote the sum of the half-perimeters of the rectangles obtained with the optimal column-based partitioning, and let $LB = 2\sum_{i=1}^{p} \sqrt{s_i}$. Then*

$$\frac{\hat{C}}{LB} \leq \sqrt{r}\left(1 + \frac{1}{\sqrt{p}}\right).$$

PROOF. Consider a column-based partitioning with $\mathcal{C} = \lceil \sqrt{r} \sum \sqrt{s_i} \rceil$ columns. Rectangles are evenly distributed amongst columns, so that the number of rectangles in each column is either $\lfloor p/\mathcal{C} \rfloor$ or $\lceil p/\mathcal{C} \rceil$. Letting $\hat{C}^*$ denote the sum of the half-perimeters of the rectangles obtained with this column partitioning, we have

$$\begin{aligned} \hat{C}^* &\leq \left\lceil \sqrt{r} \sum \sqrt{s_i} \right\rceil + \left\lceil \frac{p}{\lceil \sqrt{r} \sum \sqrt{s_i} \rceil} \right\rceil \\ &\leq 2 + \sqrt{r} \sum \sqrt{s_i} + \frac{p}{\sqrt{r} \sum \sqrt{s_i}}. \end{aligned}$$

Thus

$$\frac{\hat{C}^*}{2\sum \sqrt{s_i}} \leq \frac{1}{\sum \sqrt{s_i}} + \frac{\sqrt{r}}{2} + \frac{p}{2\sqrt{r} \sum \sqrt{s_i}}.$$

Moreover,

$$\begin{aligned} \sum s_i = 1 \quad &\Longrightarrow \quad p \max s_i \geq 1 \\ &\Longrightarrow \quad \min s_i \geq \frac{1}{pr} \end{aligned}$$

and thus

$$\sum \sqrt{s_i} \geq p\sqrt{\min s_i} \geq \sqrt{\frac{p}{r}}.$$

Therefore,

$$\begin{aligned} \frac{\hat{C}^*}{2\sum \sqrt{s_i}} &\leq \sqrt{\frac{r}{p}} + \frac{\sqrt{r}}{2} + \frac{\sqrt{r}}{2} \\ &\leq \sqrt{r}\left(1 + \frac{1}{\sqrt{p}}\right). \end{aligned}$$

The sum $\hat{C}$ of the half-perimeters of the optimal column-based partitioning is not greater than $\hat{C}^*$, which concludes the proof. □

If $r = 1$, i.e. all the processors have the same speed, the column-based partitioning is asymptotically optimal. On the other hand, if r is large, i.e. one processor is much faster than another, the bound is very pessimistic.

4.2. *Recursive Approximation Algorithm.* In this section we give a recursively defined approximation algorithm, that will lead to a good approximation factor. We introduce this approximation algorithm in two steps: first we define a column-based tiling which is different from that of the previous section; the idea here is to impose some ratio on the shapes of the rectangles. This column-based tiling is then used as a building block in the second step, where we derive the final tiling.

4.2.1. *Column-Based Partitioning.* We aim at tiling a rectangle R (not a square!) of size $h \times v$ such that $h \leq v \leq 4h$ into p rectangles of areas $s_1 \geq s_2 \geq \cdots \geq s_p$, where $\sum_{i=1}^{p} s_i = hv$. We further assume that $r = s_1/s_p \leq 2$. In this section we show that there exists a column-based tiling of R into rectangles of area $s_i = h_i \times v_i$ such that

$$(CR): \quad \tfrac{1}{4}v_i \leq h_i \leq 4v_i, \qquad \forall i, \quad 1 \leq i \leq p.$$

Note that this condition is equivalent to $\sqrt{s_i}/2 \leq h_i, v_i \leq 2\sqrt{s_i}$. For convenience, we define

$$\begin{cases} (CR)_h & \text{the condition} \quad h_i \geq \frac{1}{4}v_i, \\ (CR)_v & \text{the condition} \quad h_i \leq 4v_i. \end{cases}$$

The algorithm consists of two main phases: in the first phase the columns are filled in with rectangles. We stop adding rectangles to a given column, and create a new column, as soon as the criterion $(CR)_v$ is fulfilled for each rectangle in the column. Thus, at the end of the first phase, all the rectangles (except perhaps those in the last column) fulfill the criterion $(CR)_v$. Then, if the rectangles of the last column do not fulfill the condition, we dispatch those rectangles among other columns and delete the last column. In what follows we prove that the number of rectangles in the last column is less than the number of remaining columns, and that we can add one rectangle to any remaining column without loosing the criterion $(CR)_v$.

First_phase
  $i = 1$
  $C_i = \{s_1\}$
  for $j = 2$ **to** p
    if $(CR)_v$ is reached for all elements of C_i
      $i = i + 1$
      $C_i = s_j$
    endif
    else $C_i = C_i \cup s_j$
  endfor
  $\#columns = i$
endFirst_phase

Second_phase *Let* $s'_1 \geq s'_2 \cdots \geq s'_l$ *be the elements of the last column* C_i.
  if $(CR)_v$ is not reached by any element of C_i
    $\forall j \in \{1, \ldots, l\}$, $C_{j+i-l-1} = C_{j+i-l-1} \cup s'_j$
    $C_i = \emptyset$
    $\#columns = i - 1$
  endif
endSecond_phase

Figure 8 represents the partitioning of a 3×3.6 rectangle obtained with the following seven rectangle areas: 2, 2, 1.9, 1.5, 1.2, 1.2, 1. The proof of the correctness of the algorithm (that condition (CR) holds at the end) follows from the following three statements:

- [*Initial* $(CR)_h$]: consider any column (even the last one) after the first phase. Then any element in this column fulfills condition $(CR)_h$.
- [*Final* $(CR)_h$]: the condition $(CR)_h$ still holds if we add one element to any column.
- [*Enough columns*]: assume that there are $c + 1$ columns after the first phase. If the l elements in the last column do not fulfill condition $(CR)_v$, then $l \leq c$.

Indeed, consider a column composed of k elements $s'_1 \geq \cdots \geq s'_k$. Then for each element $s'_i = h'_i \times v'_i$ of this column, we have $h'_i = s'_i / \sum_{j=1}^{k} s'_j \times h$ and $v'_i = (\sum_{j=1}^{k} s'_j)/h$. Consequently, we have

$$(CR)_h: \quad \sqrt{s'_k} \geq \frac{1}{2} \frac{\sum_{j=1}^{k} s'_j}{h} \qquad \text{and} \qquad (CR)_v: \quad \sqrt{s'_1} \leq 2 \frac{\sum_{j=1}^{k} s'_j}{h}.$$

In particular, we see that as soon as $(CR)_v$ holds true for the elements of a column, then it will still hold true if we add a new element to this column.

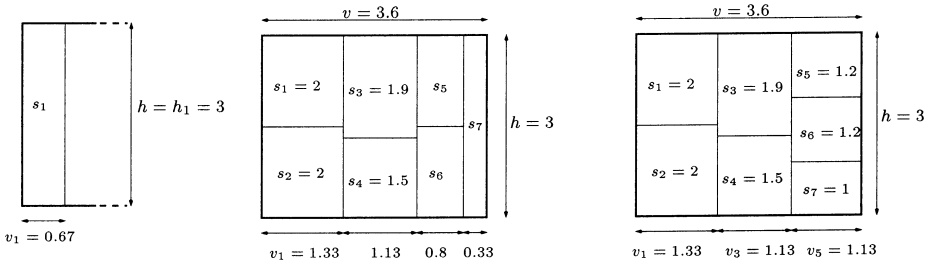


Fig. 8. The first two figures correspond to the first phase of the algorithm: columns are filled using $(CR)_v$ as a stopping criterion: for the first column, leaving s_1 alone would not fulfill the condition $(CR)_v$. Indeed, $h_1/4 = \frac{3}{4} = 0.75 > 0.67 = v_1$. Another element must be added to the column. Thus, we get $h_1 = h_2 = 1.5$ whereas $v_1 = 1.33$. The condition is then fulfilled and the algorithm goes through the next column. The last figure corresponds to the second phase of the algorithm: since the elements of the last column do not fulfill condition $(CR)_v$, they are distributed over other columns (in this case, only one element ($s_7 = 1$) has to be distributed).

Initial $(CR)_h$. We denote by $s'_1 \geq s'_2 \geq \cdots \geq s'_k$ the elements of the column. Let v'_1 denote its width and $\forall 1 \leq i \leq k$, let h'_i denote the height of the rectangle of area s'_i. Two situations may occur:

1. $[k = 1]$: we have $v'_1 \leq v \leq 4h = 4h'_1$
2. $[k > 1]$: because $(CR)_h$ does not hold if we remove the kth element, we have $\forall i \in \{1, \ldots, k\}$, $(\sum_{j=1}^{k-1} s'_j)/h < \frac{1}{2}\sqrt{s'_1} < (1/\sqrt{2})\sqrt{s'_i}$. Consequently, $v'_1 = (\sum_{j=1}^{k} s'_j)/h < k/(k-1) \times (1/\sqrt{2})\sqrt{s'_i} < 2\sqrt{s'_i}$.

Final $(CR)_h$. Let $s'_1 \geq s'_2 \geq \cdots \geq s'_k$ be the elements of a column, and let $s = s'_{k+1}$ be the element to be added to this column. Taking into account the new element, we define the values v'_1 and h'_i as previously. Three different cases may occur:

1. $[k = 1]$: we show that in the worst case, there is only one element in the last column. Indeed, consider a column $s''_1, \ldots, s''_l$ of $l > 1$ elements. Then $v''_1 = (\sum_{i=1}^{l} s''_i)/h \geq (l/2)(s'_1/h) \geq \frac{1}{4}h'_1 = \frac{1}{4}h > \frac{1}{4}h''_i$. Consequently, $s/h < h/4$. However, $s'_1 \leq 2s$. Hence, $v'_1 = (s'_1 + s)/h < \frac{3}{4}h$. With two elements in a column, $h'_i \geq h/3$, so finally $v'_1 < \frac{9}{4}h'_i$.
2. $[k = 2]$: in this case, $s'_1/h < h/4$ and $s'_1 \geq s'_2 \geq s$. Thus, $v'_1 = (s'_1 + s'_2 + s)/h < \frac{3}{4}h$. If there are three elements in the column, then $h'_i \geq h/5$. Consequently, $v'_1 < \frac{15}{4}h'_i$.
3. $[k \geq 3]$: in this case $v'_1 = (\sum_{i=1}^{k+1} s'_i)/h < (k+1)/(k-1) \times (\sum_{i=1}^{k-1} s'_i)/h < (k+1)/(k-1) \times \sqrt{s'_1}/2 < \sqrt{2}\sqrt{s'_i}$.

Enough Columns. Let v_i (for $1 \leq i \leq c+1$) be the width of the ith column after the first phase. Let h'_i (for $1 \leq i \leq l$) denote the heights of the elements of the last column. We have shown above (see paragraph *Initial* $(CR)_v$) that $\forall 1 \leq i \leq c,\ 1 \leq j \leq p$, $v_i < \sqrt{2}\sqrt{s_j}$. Since $(CR)_v$ is not fulfilled by the elements of the $(c+1)$th column we know that $v_{c+1} < (1/\sqrt{2})\sqrt{s_j} < \sqrt{2}\sqrt{s_j}$ and $\forall 1 \leq i \leq l,\ 1 \leq j \leq p, h'_i > \sqrt{2}\sqrt{s_j}$. Consequently, $\forall 1 \leq i \leq c+1,\ 1 \leq j \leq l, v_i < h'_j$. Moreover, since $v \geq h$, $\sum_{i=1}^{c+1} v_i = v$ and $\sum_{j=1}^{l} h'_j = h$, it is clear that $l \leq c$.

Remark. A consequence of this result is that we can improve the bound of Proposition 1 when $r \leq 2$: we get $\hat{C}/LB \leq \frac{5}{4}$.

4.2.2. *Recursively Defined Partitioning*. The main idea here is to split the list $s_1 \geq s_2 \geq \cdots \geq s_p$ of the rectangle areas into sub-lists so that the previous condition $r \leq 2$ holds within each sub-list. For instance, if $S = (0.49, 0.2, 0.2, 0.1, 0.01)$, then we get three sub-lists (0.49), (0.2, 0.2, 0.1) and (0.01). Then we compute the sum of the elements within each sub-list. Considering those results as new input values, we get a smaller problem. In the example, we get $S = (0.5, 0.49, 0.01)$). Then we restart the process recursively until no more merging is possible. In the example, we reach convergence after the third step, with $S = (0.99, 0.01)$). At the end of the process, since $\forall i,\ \ s_i > 2s_{i+1}$, the following inequality holds true:

$$\sum_{j>i} s_j < \frac{s_i}{2} + \frac{s_i}{4} + \cdots < s_i.$$

In what follows we denote by S_i the sub-lists obtained after convergence and by k_i the cardinality of S_i. The partitioning algorithm is recursively defined with two main functions, as outlined below:

Initial_square $(h, v, S = (s_1, \ldots, s_k))$
  where necessary, the s_i *should fulfill the condition* $s_i > \sum_{j>i} s_j$
  if $k > 1$
    if $v \geq h$
      partition $h \times v$ into $h \times v_1 = s_1$ and $h \times v_2 = \sum_{j>i} s_j$
      compute **Column_based**(h, v_1, S_1) and **Initial_square**$(h, v_2, (s_2, \ldots, s_k))$
    else
      partition $h \times v$ into $h_1 \times v$ and $h_2 \times v$ and compute similarly
    endif
  endif
endInitial_square

Column_based $(v, h, S = (s_1, \ldots, s_k))$
  where necessary, the s_i *should fulfill the condition* $\frac{1}{2} \leq s_i/s_j \leq 2$
  if $\frac{1}{4} \leq v/h \leq 4$
    apply steps 1 and 2 of the algorithm of Section 4.2.1
  else
    apply step 1 only of the algorithm of Section 4.2.1
  endif
  $\forall i$ such that $k_i > 1$, compute **Column_based**(v_i, h_i, S_i)
endColumn_based

PROPOSITION 2. *Let* $\hat{C}$ *denote the sum of the half-perimeters of the rectangles obtained with the recursively defined partitioning, and let* $LB = 2\sum_{i=1}^{p} \sqrt{s_i}$. *Then*

$$\hat{C} \leq 1 + \tfrac{5}{4} LB.$$

PROOF. We have to show that the additional cost to pay for rectangles that do not fulfill condition (*CR*) is less than 1. For the partitioning of a given rectangle, we call this quantity *the extra cost*. Depending on the depth of the recursion, two kinds of partitioning may arise:

- If *all* the elements differ by a ratio of *less* than 2, then a column-based partitioning is used.
- If *all* the elements differ by a ratio of *more* than 2, then the rectangle is partitioned into two parts and the smallest part is itself partitioned into two parts recursively.

Consequently, the proof is made of two parts:

1. [*Extra cost for the column-based partitioning*] In that case we show that the extra cost for the partitioning of a rectangle of size $h \times v$ $(v \geq h)$ is less than $v - h$.
2. [*Extra cost for the initial partitioning of the square*] In that case we show that the extra cost for the partitioning of a rectangle of size $h \times v$ $(v \geq h)$ is less than h.

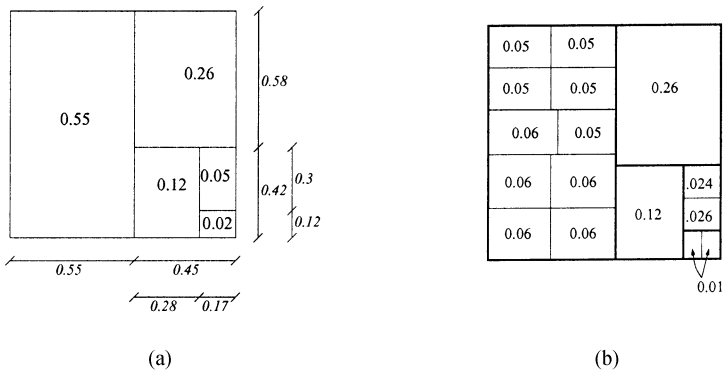


Fig. 9. Partitioning the square with the following initial list of rectangles: $S = (0.26, 0.12, 0.06, 0.06, 0.06, 0.06, 0.06, 0.06, 0.05, 0.05, 0.05, 0.05, 0.05, 0.026, 0.024, 0.01, 0.01)$. In this case convergence is obtained after one step of the algorithm, and then $S = (0.55, 0.26, 0.12, 0.05, 0.02)$. (a) The initial partitioning and (b) the final partitioning. Note that the rectangles of the initial partitioning have been recursively partitioned into smaller rectangles with the column-based algorithm.

Extra cost for the column-based partitioning.

1. If there is only one rectangle, then the extra cost is $v + h - 2\sqrt{vh} \leq v - h$.
2. Suppose that there are more than one rectangle and that the initial rectangle is so thin that its partitioning is made of one element only per column. In that case, for each rectangle, the extra cost is less than $v_i - h$, so that the overall extra cost is less than $\sum_{i=1}^{c} v_i - h = v - ch < v - h$.
3. Suppose that there are more than one rectangle per column so that the last column only is unbalanced (its elements do not fulfill condition (*CR*)). For each element of the last column, the extra cost is less than $h_i - v_c$. Hence, the overall extra cost is less than $\sum_{i=1}^{l} h_i - v_c = h - lv_c < h$ and, since $4h < v$, less than $v - h$.

Extra cost for the initial partitioning of the square. Suppose without loss of generality that the $h \times v$ (where $v \geq h$) rectangle is partitioned into two rectangles $h \times v_1$ and $h \times v_2$, where $v_1 > v_2$. Thus, $h \times v_1$ is tiled with the column-based algorithm, so that the extra cost for this rectangle is less than $v_1 - h$ if $v_1 > h$ and 0 otherwise, since $h \leq v < 2v_1$.

Two situations may occur for the remaining rectangle:

- If $v_2 \geq h$, its extra cost is less than v_2, so $v_1 > v_2 \geq h$. Hence, the overall extra cost is less than $v_1 - h + v_2 < v$.
- If $v_2 \leq h$, its extra cost is less than h. So, either $v_1 \geq h$, then the overall extra cost is less than $v_1 - h + h < v$; or $v_1 < h$, and then the overall extra cost is less than $h \leq v$.

As a consequence, the extra cost for the partitioning of the initial square is less than 1. □

Since $LB = 2\sum_i \sqrt{s_i} \geq 2$, Proposition 2 provides a $\frac{7}{4}$-approximation algorithm for PERI-SUM.

5. Approximation Algorithms for PERI-MAX. In this section we introduce a polynomial-time approximation algorithm to solve the PERI-MAX problem. Again, we consider a column-based partitioning of the unit square. We consider two different approximation algorithms, according to the area of the largest rectangle. Let $s_1 \geq s_2 \geq \cdots \geq s_p$ denote the given areas of the rectangles.

If s_1 is greater than $\frac{1}{3}$, we use a first approximation algorithm. In this case one column is created for each element. Therefore, the half-perimeter of one rectangle of area s_i is $1 + s_i$. In this case

$$\forall 1 \leq i \leq p, \qquad r_i = \frac{1 + s_i}{2\sqrt{s_1}} \leq r_1 \leq \frac{2}{\sqrt{3}}.$$

In the case where s_1 is less than $\frac{1}{3}$, we use a second approximation algorithm, which ensures that

$$\forall 1 \leq i \leq p, \qquad r_i = \frac{h_i + v_i}{2\sqrt{s_1}} \leq \frac{2}{\sqrt{3}}.$$

The algorithm can be stated as follows, where $\mathrm{Scol}(c)$ denotes the set of the rectangles belonging to the cth column:

Peri-max_column-based $(p, S = (s_1, \ldots, s_p))$
  First phase
  $c = 1$
  for $i = 1$ **to** p
    $\mathrm{Scol}(c) = \mathrm{Scol}(c) \cup \{i\}$
    if $\sum_{i \in \mathrm{Scol}(c)} s_i \geq 2\sqrt{s_1/3} - \sqrt{4s_1/3 - \max_{i \in \mathrm{Scol}(c)} s_i}$.
      $c = c + 1$
    endif
  endfor
  $c_{\max} = c$

  Second phase
  if $\sum_{i \in \mathrm{Scol}(c_{\max})} s_i \leq 2\sqrt{s_1/3} - \sqrt{4s_1/3 - \max_{i \in \mathrm{Scol}(c_{\max})} s_i}$
    $\mathrm{Scol}(1) = \mathrm{Scol}(1) \cup \mathrm{Scol}(c_{\max})$
    $c_{\max} = c_{\max} - 1$
  endif
endPeri-max_column-based

The algorithm works as follows. During the first phase we fill in columns with rectangles, and we create a new column as soon as the condition $\sum_{i \in \mathrm{Scol}(c)} s_i \geq 2\sqrt{s_1/3} - \sqrt{4s_1/3 - \max_{i \in \mathrm{Scol}(c)} s_i}$ is fulfilled, which ensures that all the rectangles belonging to the column satisfy the criterion $r_i \leq 2/\sqrt{3}$. Thus, at the end of the first phase, all the rectangles (except perhaps those belonging to the last column) fulfill this criterion. The second phase of the algorithm consists in deleting the last column (if its rectangles do not fulfill the criterion) and to add its rectangle to the first column. In what follows we will prove that this operation is always valid (i.e. there are at least two columns), and

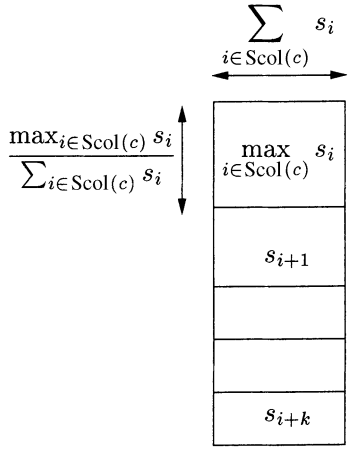


Fig. 10. $\mathrm{Scol}(c)$.

that the criterion $r_i \leq 2/\sqrt{3}$ still holds true for the rectangles of the first column after the second phase.

The configuration of one of the columns Scol is depicted in Figure 10. The largest perimeter of the rectangles in the column $\mathrm{Scol}(c)$ is

$$\sum_{i\in \mathrm{Scol}(c)} s_i + \frac{\max_{i\in \mathrm{Scol}(c)} s_i}{\sum_{i\in \mathrm{Scol}(c)} s_i}.$$

As shown in Figure 11, the condition

$$\forall i \in \mathrm{Scol}(c), \qquad \max(h_i + v_i) \leq \frac{4\sqrt{s_1}}{\sqrt{3}}$$

holds true iff

$$2\sqrt{\frac{s_1}{3}} - \sqrt{\frac{4s_1}{3} - \max_{i\in \mathrm{Scol}(c)} s_i} \leq \sum_{i\in \mathrm{Scol}(c)} s_i \leq 2\sqrt{\frac{s_1}{3}} + \sqrt{\frac{4s_1}{3} - \max_{i\in \mathrm{Scol}(c)} s_i}.$$

Therefore, the condition

$$\forall i \in \mathrm{Scol}(c), \qquad \max(h_i + v_i) \leq \frac{4\sqrt{s_1}}{\sqrt{3}}$$

holds true, except perhaps for the first column (since $\mathrm{Scol}(c_{\max})$ have possibly been added). Indeed, since

$$x_{\max}(\mathrm{Scol}(c)) - x_{\min}(\mathrm{Scol}(c)) = 2\sqrt{\frac{4s_1}{3} - \max_{i\in \mathrm{Scol}(c)} s_i} \geq \frac{2\sqrt{s_1}}{\sqrt{3}} \geq s_1,$$

we cannot jump from $x_{\min}(\mathrm{Scol}(c))$ to $x_{\max}(\mathrm{Scol}(c))$ by adding just one rectangle to $\mathrm{Scol}(c)$.

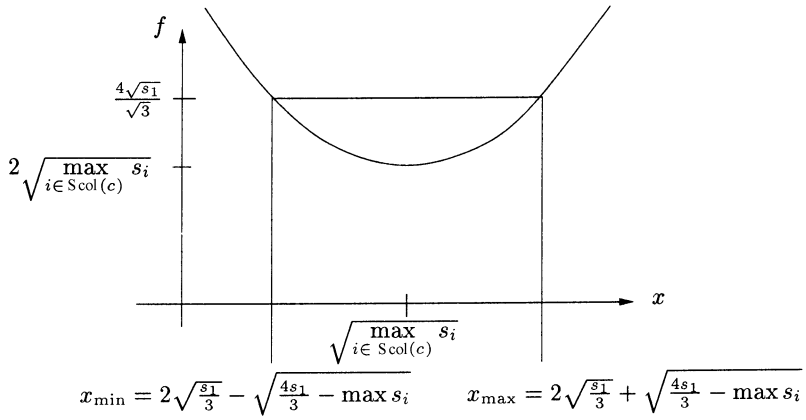


Fig. 11. Plot of the function $f(x) = x + (\max_{i\in \mathrm{Scol}(c)} s_i)/x$.

Hence, in order to prove the correctness of our algorithm, we need to prove the following two points:

- There are at least two columns at the end of the first step of the algorithm. Indeed,

$$\sum_{i\in\{1..n\}} s_i = 1 > \sqrt{s_1} \geq x_{\min}(\mathrm{Scol}(1)).$$

- Suppose that the last column $\mathrm{Scol}(c_{\max})$ does not fulfill the condition

$$\forall i \in \mathrm{Scol}(c_{\max}), \qquad \max(h_i + v_i) \leq \frac{4\sqrt{s_1}}{\sqrt{3}}.$$

Then the condition

$$\forall i \in \mathrm{Scol}(1), \qquad \max(h_i + v_i) \leq \frac{4\sqrt{s_1}}{\sqrt{3}}$$

still holds true if

$$\mathrm{Scol}(1) = \mathrm{Scol}(1) \cup \mathrm{Scol}(c_{\max}).$$

Indeed, in this case, we know that

$$\sum_{i\in\mathrm{Scol}(c_{\max})} s_i \leq x_{\min}(\mathrm{Scol}(c_{\max})) = 2\sqrt{\frac{s_1}{3}} - \sqrt{\frac{4s_1}{3} - \max_{i\in\mathrm{Scol}(c_{\max})} s_i} \leq \sqrt{\frac{s_1}{3}}$$

and

$$\sum_{i\in\mathrm{Scol}(1)} s_i \leq x_{\min}(\mathrm{Scol}(1)) + s_1 \leq \sqrt{\frac{s_1}{3}} + s_1.$$

Therefore, since $s_1 \leq \frac{1}{3}$,

$$\sum_{i\in\mathrm{Scol}(c_{\max})\cup\mathrm{Scol}(1)} s_i \leq 2\sqrt{\frac{s_1}{3}} + s_1 \leq \sqrt{3s_1} = x_{\max}(\mathrm{Scol}(1)).$$

In summary, by using one of the two approximation algorithms according to the value of s_1, we have proven the following proposition:

PROPOSITION 3. *Let $\hat{M}$ denote the maximum of the half-perimeters of the rectangles obtained with the above approximation algorithm, and let $LB = 2\sqrt{s_1}$. Then*

$$\frac{\hat{M}}{LB} \leq \frac{2}{\sqrt{3}}.$$

Note that it is impossible to obtain a better guarantee (without taking into account the actual values of the s_i's). Indeed, if we consider the following situation with $s_1 = s_2 = s_3 = \frac{1}{3}$, then the optimal solution satisfies to

$$\hat{M} = \max_i (h_i + v_i) = \frac{2}{\sqrt{3}} 2\sqrt{s_1} = \frac{2}{\sqrt{3}} LB.$$

6. Related Results. In this section we survey results on geometric optimization problems similar to PERI-SUM or PERI-MAX:

Covering a Square by Small Perimeter Rectangles. Alon and Kleitman [1] consider the tiling of the unit square into n rectangles. There is no constraint on the area of the rectangles. They show that one of the rectangles must have perimeter at least $4(2m + 1)/(n + m(m + 1))$, where m is the largest integer whose square is at most n. This result is exact for $n = m(m + 1)$ or $n = m^2$.

Decomposition of a Square into Rectangles of Minimal Perimeter. Kong et al. [12] determine how to tile the unit square into p rectangles of the same area so as to minimize the maximum perimeter of these rectangles. This is exactly our PERI-MAX problem constrained to same-area rectangles ($s_i = 1/p$ for $1 \leq i \leq p$). This problem is shown to be polynomial in [12]. The optimal solution is one of the following two arrangements: let either $m = \lfloor \sqrt{p} \rfloor$ or $m = \lceil \sqrt{p} \rceil$, and use m columns composed of $\lfloor p/m \rfloor$ or $\lceil p/m \rceil$ rectangles. This solution is extended to deal with the decomposition of a rectangle (instead of a square) onto same-area rectangles in [11].

Partitioning a Rectangle with Interior Points. Another related problem is to find the minimum partition of a rectangle with interior points: given a rectangle R and a finite set P of points located inside R, find a set of line segments that partition R into rectangles such that every point in P is on the boundary of some rectangle. The goal is to minimize the total length of the introduced line segments. This problem is shown to be NP-complete in [13] and approximation algorithms are given in [7] and [8]. The link with our PERI-MAX problem is that the objective function is the same, but the original motivation in [7] and [8] was a VLSI routing problem (and the constraints are quite different).

Array Partitioning. The minimum rectangle tiling problem [9] is partially related to our optimization problems PERI-MAX: given an $n \times n$ array A of non-negative numbers, and a positive integer p, find a partition of A into p non-overlapping rectangular subarrays, such that the maximum weight of any rectangle in the partition is minimized (the weight of a rectangle is the sum of its elements). This problem is NP-complete, and approximation algorithms are given in [10] and [9]

Finally, note that there are several NP-complete geometric optimization problems that are listed in the NP Compendium [5]. See also the survey book [2].

7. Conclusion. In this paper we have dealt with two geometric problems arising from heterogeneous parallel computing. Because both problems have been shown to be NP-complete, we have introduced approximation algorithms.

The original motivation for this work is very important: the MM algorithm is the prototype of tightly coupled kernels that need to be implemented efficiently on distributed and heterogeneous platforms: we view it as a perfect testbed before experimenting more challenging computational problems on the grid.

Appendix. Column-Based Approximation Algorithm for PERI-SUM. The algorithm is outlined as follows, where $f_{\mathcal{C}}(q)$ denotes the perimeter of the optimal column-based partitioning with q rectangles using $\mathcal{C}$ columns and $f_{\mathcal{C}}^{\text{cut}}(q)$ denotes the number of rectangles in the $\mathcal{C}-1$ first columns of the optimal partitioning with q rectangles and $\mathcal{C}$ columns. $f_{\mathcal{C}}^{\text{cut}}$ will be used to build the optimal partitioning during the second step of the algorithm.

$S = 0$
for $q = 1$ to p
  $S = S + s_q$
  $f_1(q) = 1 + S \times q$
  $f_1^{\text{cut}}(q) = 0$
endfor
for $\mathcal{C} = 2$ to p
  for $q = \mathcal{C}$ to p
    $f_{\mathcal{C}}(q) = \min_{1 \le r \le q-\mathcal{C}+1}(1 + S \times r + f_{\mathcal{C}-1}(q-r))$
    $f_{\mathcal{C}}^{\text{cut}}(q) = q - r_{\min}$
  endfor
endfor

The final partitioning corresponding to the function $f_{\mathcal{C}_{\min}}(p) = \min_{1 \le \mathcal{C} \le p} f_C(p)$ is found using the following algorithm:

$q = p$
for $\mathcal{C} = \mathcal{C}_{\min}$ **downto** 2
  $k_{\mathcal{C}} = q - f_{\mathcal{C}}^{\text{cut}}(q)$
  $q = f_{\mathcal{C}}^{\text{cut}}(q)$
endfor
$k_1 = q$

Table 1. The values of the couples $f_{\mathcal{C}}(q)/r$ where $f_{\mathcal{C}}(q) = \min_{q'}(1 + (\sum_{q'<i\leq q} s_i) \times (q - q') + f_{\mathcal{C}-1}(q')) = 1 + (\sum_{r<i\leq q} s_i) \times (q - r) + f_{\mathcal{C}-1}(r)$. (Bold entries correspond to the optimal solution.)

	q							
$\mathcal{C}$	1	2	3	4	5	6	7	8
1	1.02 / 0	1.12 / 0	1.36 / 0	**1.8 / 0**	3 / 0	4.6 / 0	6.6 / 0	9 / 0
2		2.06 / 1	2.18 / 2	2.4 / 2	2.92 / 3	**3.6 / 4**	4.6 / 4	5.8 / 5
3			3.12 / 2	3.26 / 3	3.6 / 4	4.12 / 5	4.72 / 5	**5.4 / 6**
4				4.2 / 3	4.46 / 4	4.8 / 5	5.32 / 6	5.92 / 7
5					5.4 / 4	5.66 / 5	6 / 6	6.52 / 7
6						6.6 / 5	6.86 / 6	7.2 / 7
7							7.8 / 6	8.06 / 7
8								9 / 7

which corresponds to tracking (backwards) the bold entries in Table 1. The unit square is partitioned into $\mathcal{C}_{\min}$ columns. The ith column contains the rectangles $s_d, s_{d+1}, \ldots, s_{d+k_i}$ with $d = k_1 + k_2 + \cdots + k_{i-1}$.

To help understand the derivation, we apply the algorithm on the following toy example: we have $p = 8$ areas of values $(0.02, 0.04, 0.06, 0.08, 0.2, 0.2, 0.2, 0.2)$. The results of the algorithm are described in Table 1. Each column $\mathcal{C}_i$ contributes to the sum of the half-perimeters as follows: 1 for the vertical line, and $k_i \times c_i$ for the k_i horizontal lines of length c_i.

In the example, the optimal partitioning is obtained for three columns ($f_3(8) = 5.4$). The last column of width $c_3 = s_7 + s_8 = 0.4$ is made of two elements. The second column of width $c_2 = s_5 + s_6 = 0.4$ is also made of two elements. Then the first column of width $c_1 = s_1 + s_2 + s_3 + s_4 = 0.2$ is made of the smallest four elements. Figure 12 represents this partitioning.

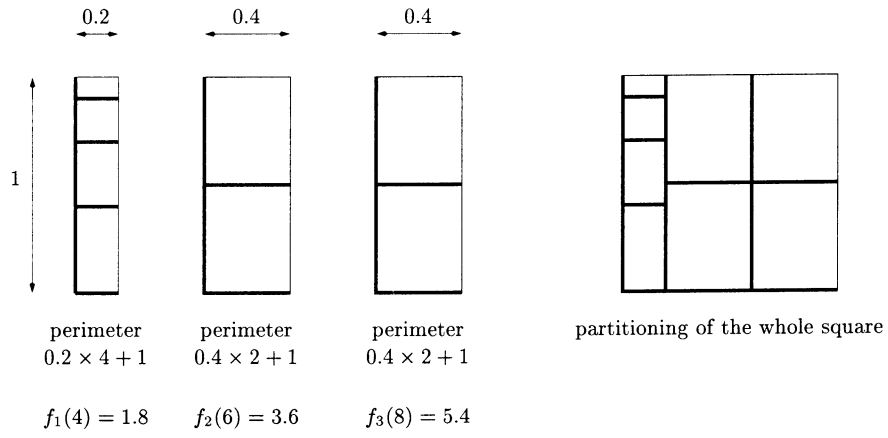


Fig. 12. Optimal column-based partitioning for the example. Thicker lines correspond to the sum of the half-perimeters.

References

[1] N. Alon and D.J. Kleitman. Covering a square by small perimeter rectangles. *Discrete Comput. Geom.*, 1:1–7, 1986.

[2] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation*. Springer-Verlag, Berlin, 1999.

[3] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert. Matrix multiplication on heterogeneous platforms. *IEEE Trans. Parallel Distrib. Systems*, 12(10):1033–1051, 2001.

[4] L.S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley. *ScaLAPACK Users' Guide*. SIAM, Philadelphia, PA, 1997.

[5] P. Crescenzi and V. Kann. A compendium of NP optimization problems. World Wide Web document, URL: http://www.nada.kth.se/~viggo/wwwcompendium/wwwcompendium.html.

[6] M.R. Garey and D.S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA, 1991.

[7] T.F. Gonzalez and S. Zheng. Improved bounds for rectangular and guillotine partitions. *J. Symbolic Comput.*, 7:591–610, 1989.

[8] T.F. Gonzalez and S. Zheng. Approximation algorithm for partitioning a rectangle with interior points. *Algorithmica*, 5:11–42, 1990.

[9] S. Khanna, S. Muthukrishnan, and M. Paterson. On approximating rectangle tiling and packing. In *Proc. 9th Ann. ACM–SIAM Symposium on Discrete Algorithms*, pages 384–393. ACM Press, New York, 1998.

[10] S. Khanna, S. Muthukrishnan, and S. Skiena. Efficient array partitioning. In *Proc. 24th Internat. Colloquium on Automata, Languages and Programming*, LNCS 1256, pages 616–626. Springer-Verlag, Berlin, 1997.

[11] T.Y. Kong, D.M. Mount, and W. Roscoe. The decomposition of a rectangle into rectangles of minimal perimeter. *SIAM J. Comput.*, 17(6):1215–1231, 1988.

[12] T.Y. Kong, D.M. Mount, and M. Wermann. The decomposition of a square into rectangles of minimal perimeter. *Discrete Appl. Math.*, 16:239–243, 1987.

[13] A. Lingas, R.Y. Pinter, R.L. Rivest, and A. Shamir. Minimum edge length partitioning of rectilinear polygons. In *Proc. 20th Ann. Allerton Conference on Communication, Control and Computing*, pages 53–63, 1982.

A Proposal for a Heterogeneous Cluster ScaLAPACK (Dense Linear Solvers)

Olivier Beaumont, Vincent Boudet, Antoine Petitet,
Fabrice Rastello, and Yves Robert, *Member*, *IEEE*

Abstract—In this paper, we study the implementation of dense linear algebra kernels, such as matrix multiplication or linear system solvers, on heterogeneous networks of workstations. The uniform block-cyclic data distribution scheme commonly used for homogeneous collections of processors limits the performance of these linear algebra kernels on heterogeneous grids to the speed of the slowest processor. We present and study more sophisticated data allocation strategies that balance the load on heterogeneous platforms with respect to the performance of the processors. When targeting unidimensional grids, the load-balancing problem can be solved rather easily. When targeting two-dimensional grids, which are the key to scalability and efficiency for numerical kernels, the problem turns out to be surprisingly difficult. We formally state the 2D load-balancing problem and prove its NP-completeness. Next, we introduce a data allocation heuristic, which turns out to be very satisfactory: Its practical usefulness is demonstrated by MPI experiments conducted with a heterogeneous network of workstations.

Index Terms—Heterogeneous network, heterogeneous grid, different-speed processors, load-balancing, data distribution, data allocation, numerical libraries, numerical linear algebra, heterogeneous platforms, cluster computing.

1 Introduction

Heterogeneous networks of workstations (HNOWs) are ubiquitous in university departments and companies. They represent the typical poor man's parallel computer: Running a large PVM or MPI experiment (possibly all night long) is a cheap alternative to buying supercomputer hours. The idea is to make use of *all* available resources, namely slower machines *in addition to* more recent ones.

The major limitation to programming heterogeneous platforms arises from the additional difficulty of balancing the load when using processors running at different speeds. This paper is devoted to providing the required framework to build an extension of the ScaLAPACK library [8] capable of running on top of HNOWs or nondedicated parallel machines. More precisely, we concentrate on dense linear algebra kernels, such as matrix multiplication, or LU and R decompositions. Our goal is to come up with an efficient implementation of such kernels within the framework of the library: The idea is not to rebuild ScaLAPACK kernels from scratch; instead, we take advantage of the deep modularity of the library and we modify only high-level routines related to data distribution. With processors running at different speeds, block-cyclic distribution is no longer enough; new data distribution schemes must be determined and analyzed. We show that deriving efficient distribution schemes is a rather simple task for linear networks, but turns out to be surprisingly difficult for two-dimensional grids. Technically, we prove that the underlying optimization problem is NP-hard for two-dimensional grids and we provide an efficient approximation of the optimal solution.

The rest of the paper is organized as follows: In Section 2, we discuss the framework for implementing our heterogeneous kernels and we briefly review the existing literature. The core of the paper is composed of Sections 3 and 4, where we outline our main results: Section 3 is devoted to data distribution schemes for unidimensional grids; Section 4 is its counterpart for two-dimensional grids. The practical usefulness of our tuned data distribution schemes is demonstrated in Section 5 through several MPI experiments run on two HNOWs configured as both unidimensional and two-dimensional grids. Finally, we give some remarks and conclusions in Section 6.

• *The authors are with LIP, UMR CNRS-ENS Lyon-INRIA 5668, Ecole Normale Supérieure de Lyon, F-69364 Lyon Cedex 07, France.*
E-mail: {Olivier.Beaumont, Vincent.Boudet, Antoine.Petitet, Fabrice.Rastello, Yves.Robert}@ens-lyon.fr.

Manuscript received 7 July 2000; accepted 23 May 2001.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 112416.

2 Framework

2.1 Static Distribution Schemes

Because we have a library designer's approach, we target static strategies to allocate data (and associated computations) to the heterogeneous processors. These processors will be configured either as a unidimensional or as a two-dimensional grids, according to the (virtual) hardware configurations currently supported by ScaLAPACK. For homogeneous platforms, ScaLAPACK uses a block-cyclic distribution approach. This means that subblocks (rather than single elements) of the matrices are distributed to processors in a wraparound fashion along the processor grid (this in both dimensions for a two-dimensional grid). Processors are then responsible for the computations to be performed on the data blocks that have been assigned to them.

The advantages of block-cyclic distribution for homogeneous platforms are easily understood. Blocked versions of the classical (systolic-like) parallel algorithms for matrix multiplication and linear system solvers [30] are used in ScaLAPACK to squeeze the most out of state-of-the-art processors with pipelined arithmetic units and multilevel memory hierarchy [19], [10]. Blocked algorithms naturally imply a distribution of matrix blocks to processors. Because matrix blocks are not accessed and operated upon evenly in dense system solvers (the matrix shrinks as the decomposition progresses), a cyclic distribution of blocks is used rather than a plain block distribution. Altogether, the block-cyclic distribution provides an efficient load-balancing of the work while enabling local (scalar) computations to be performed at the highest rate.

Blocked algorithms will be used for heterogeneous platforms, too, so matrix blocks will be distributed to processors as in the homogeneous case. However, a cyclic distribution of blocks is no longer likely to provide good load balancing. Intuitively, if a processor is, say, twice as fast as another one, it should be allocated twice as many blocks. Deriving efficient distribution schemes for heterogeneous machines is the main objective of this paper.

Our ScaLAPACK extension exposes a major difficulty when using heterogeneous platforms. Indeed, assume the platform is configured as a (virtual) two-dimensional grid: Such a configuration may well be used for a large cluster of workstations linked by a fast and dedicated network, such as Myrinet [17]. In that case, a two-dimensional grid would be preferred to a linear array for scalability reasons [10]: For kernels such as matrix multiplication or LU and R decompositions, a 2D block-cyclic distribution is superior to a 1D block-cyclic distribution, both theoretically [18], [11] and experimentally [8]. It turns out that configuring n heterogeneous processors into a $p \times q$ grid, with $pq \leq n$, is very difficult: To build up the grid, we have to choose the best layout of the processors among an exponential number of possible processor arrangements. We formally state this optimization problem and prove its NP-completeness in Section 4.3.2; then, we provide an efficient heuristic in Section 4.3.4.

2.2 Static versus Dynamic Strategies

Consider an HNOW: Whereas programming a large application made up of several loosely coupled tasks can be performed rather easily (because these tasks can be dispatched dynamically on the available processors), implementing a tightly coupled algorithm, such as a dense linear algebra kernel, requires carefully tuned scheduling and mapping strategies.

Distributing the computations (together with the associated data) can be performed either dynamically or statically or a mixture of both. On one hand, we may think that dynamic strategies are likely to perform better because the machine loads will be self-regulated, hence self-balanced, if processors pick up new tasks just as they terminate their current computation. However, data dependences, in addition to communication costs and control overhead, may well lead to slowing the whole process down to the pace of the slowest processors [9]. On the other hand, static strategies will suppress (or at least minimize) data redistributions and control overhead during execution. Furthermore, in the context of a numerical library, static allocations seem to be necessary for a simple and efficient memory allocation. This paper is devoted to the design of an extension of the ScaLAPACK library and will consider only static strategies. We agree, however, that targeting larger platforms such as distributed collections of heterogeneous clusters, e.g., available from the metacomputing grid [20], may well enforce the use of dynamic schemes.

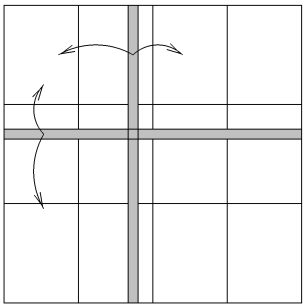


Fig. 1. The MM algorithm on a 3×4 homogeneous 2D-grid.

2.3 Matrix Multiplication on Homogeneous Grids

In this section, we survey the algorithm used in ScaLAPACK for matrix multiplication. Assume that we target a 2D homogeneous: The $p \times q$ processors are identical. In that case, ScaLAPACK uses a block version of the outer product algorithm[1] described in [1], [21], [30], which can be summarized as follows:

- Take a macroscopic view and concentrate on allocating (and operating on) matrix blocks to processors: Each element in A, B, and C is a square $r \times r$ block and the unit of computation is the updating of one block, i.e., a matrix multiplication of size r. In other words, we shrink the actual matrix size N by a factor r and we perform the multiplication of two $n \times n$ matrices whose elements are square $r \times r$ blocks, where $n = N/r$.
- At each step, a column of blocks (the pivot column) is communicated (broadcast) horizontally and a row of blocks (the pivot row) is communicated (broadcast) vertically.
- The A, B, and C matrices are identically partitioned into $p \times q$ rectangles. There is a one-to-one mapping between these rectangles and the processors. Each processor is responsible for updating its C rectangle: More precisely, it updates each block in its rectangle with one block from the pivot row and one block form the pivot column, as illustrated in Fig. 1. For square $p \times p$ homogeneous 2D-grids, and when the number of blocks in each dimension n is a multiple of p (the actual matrix size is thus $N = n.r$), it turns out that all rectangles are identical squares of $\frac{n}{p} \times \frac{n}{p}$ blocks.

This paper is devoted to an extension of this algorithm to a heterogeneous set of computing resources.

1. ScaLAPACK uses a two-dimensional grid rather than a linear array for scalability reasons [8].

2.4 Related Work

There are many papers in the literature dealing with dynamic schedulers to distribute the computations (together with the associated data) onto heterogeneous platforms. Most schedulers use simple mapping strategies such as master-slave techniques or paradigms based upon the idea *use the past to predict the future,* i.e., use the currently observed speed of computation of each machine to decide for the next distribution of work: See the survey paper of Berman [6] and the more specialized references [2], [13] for further details. Several scheduling and mapping heuristics have been proposed to map task graphs onto HNOWs [34], [35], [32], [26]. Scheduling tools such as Prophet [36] or AppLeS [6] are available (see also the survey paper [33]).

The static mapping of numerical kernels has, however, received much less attention. To the best of our knowledge, there is a single paper by Kalinov and Lastovetsky [28] which has similar objectives as ours. They are interested in LU decomposition on heterogeneous 1D and 2D grids. They propose a "heterogeneous block cyclic distribution" to map matrix blocks onto the different-speed processors. They use the *mpC* programming tool [3] to program the heterogeneous 1D or 2D grids, which they consider as fixed (they do not discuss how to configure the grid). In fact, their "heterogeneous block cyclic distribution" does not lead to a "true" 2D-grid because each processor has more than four direct neighbors to communicate with. We come back to their work in Section 4.1.2.

For the sake of completeness, we quote the following papers, which are not directly related to our work but which share some of our objectives:

- The decomposition of general data-parallel programs for execution onto heterogeneous clusters has been studied by Crandall and uinn [15], [14]: The idea is to split a data domain into different-size blocks for different-speed processors. Similarly, various array decomposition algorithms are proposed by Kaddoura et al. [27].
- The NP-complete optimization problem studied in Section 4.3.2 is related to some geometric problems such as partitioning a rectangle with interior points [31], [24] or array partitioning [25], [29]. Several NP-complete geometric optimization problems are listed in the NP Compendium [16], [4].

3 Unidimensional Grids

In this section, we target unidimensional grids, i.e., HNOWs configured as linear arrays. We investigate data allocation schemes to evenly balance the workload for ScaLAPACK kernels on such platforms.

We start with the simple problem of distributing independent chunks of computations to linear arrays of heterogeneous processors (Section 3.1). We use this result to tackle the implementation of linear solvers, for which we propose an optimal data distribution in Section 3.2.

3.1 Distributing Independent Chunks

Consider the following simple problem: Given M independent chunks of computations, each of equal size (i.e., each requiring the same amount of work), how can we assign these chunks to p physical processors $P_1, P_2, \ldots, P_p$ of respective cycle-times $t_1, t_2, \ldots, t_p$ so that the workload is best balanced? Here, the execution time is understood as the number of time units needed to perform one chunk of computation, i.e., each processor P_i executes each computation chunk within t_i time units. Then, how do we distribute chunks to processors? The intuition is that the load of P_i should be inversely proportional to t_i. Since the loads (i.e., number of chunks) on each processor must be integers, we use the following algorithm to solve the problem, where c_i denotes the number of chunks allocated to processor P_i. Thus, the overall execution obtained with an allocation $C = (c_1, c_2, \ldots, c_p)$ is given by $\max_i c_i t_i$.

lgorithm 3.1 Optimal distribution for M independent chunks over p processors of cycle-times $t_1, \ldots, t_p$:

Initialization: Approximate the c_i so that
$c_i \times t_i \approx Constant$ and $c_1 + c_2 + \ldots + c_p \leq M$.

Let $c_i = \left\lfloor \frac{\frac{1}{t_i}}{\sum_{i=1}^{p} \frac{1}{t_i}} \times M \right\rfloor$ for $1 \leq i \leq p$.

Iteratively increment some c_i until
$c_1 + c_2 + \ldots + c_p = M$

For $m = c_1 + c_2 + \ldots + c_p$ to M
find $k \in \{1, \ldots, p\}$ such that
$t_k \times (c_k + 1) = \min\{t_i \times (c_i + 1))\}$
$c_k = c_k + 1$

Proposition 1. *Algorithm 3.1 gives the optimal allocation.*

Proof. Consider an optimal allocation denoted by $o_1, \ldots, o_p$. Let j be such that $\forall i \in \{1, \ldots, p\}$, $o_j t_j \geq o_i t_i$. To prove the correctness of the algorithm, we prove the invariant

$$(I) : \forall i \in \{1, \ldots, p\}, c_i t_i \leq o_j t_j.$$

After the initialization,

$$c_i \leq \frac{\frac{1}{t_i}}{\sum_{k=1}^{p} \frac{1}{t_k}} \times M.$$

We have

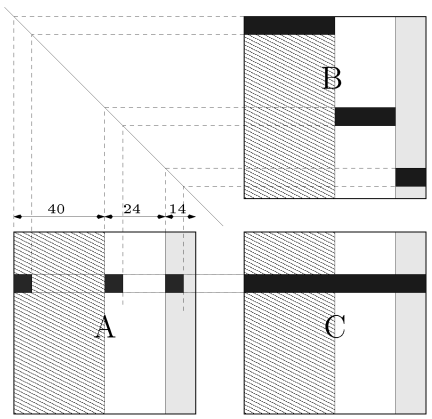
$$M = \sum_{k=1}^{p} o_k \leq o_j t_j \times \sum_{k=1}^{p} \frac{1}{t_k}.$$

Hence,

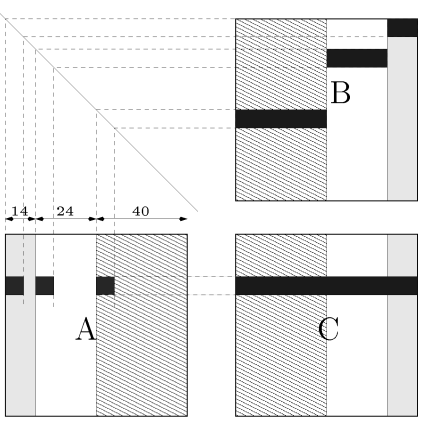
$$c_i t_i \leq \frac{M}{\sum_{k=1}^{p} \frac{1}{t_k}} \leq o_j t_j$$

and invariant (I) holds.

We use an induction to prove that invariant (I) holds after each incrementation. Suppose that, at a given step, some c_k will be incremented. Before that step, $\sum_{i=1}^{p} c_i < M$, hence, there exists $k' \in \{1, \ldots, p\}$ such that $c_{k'} < o_k$. We have $t_{k'}(c_{k'} + 1) \leq t_{k'} o_{k'} \leq t_j o_j$ and the choice of k implies that $t_k(c_k + 1) \leq t_{k'}(c_{k'} + 1)$. Invariant (I) does hold after the incrementation.



1^{st} **step**:
• p_1: $C[*, 1:40] += A[*, 1] \times B[1, 1:40]$;
• p_2: $C[*, 41:64] += A[*, 41] \times B[41, 41:64]$
• p_3: $C[*, 65:78] += A[*, 65] \times B[65, 65:78]$.



39^{th} **step**:
• p_1: $C[*, 1:40] += A[*, 39] \times B[39, 1:40]$
...

Fig. 2. Different steps of matrix multiplication on a platform made of three heterogeneous processors of respective cycle-times $t_1 = 3$, $t_2 = 5$, and $t_3 = 8$. All indices in the figure are block numbers.

Finally, the time needed to compute the M chunks with the allocation $c_1, \ldots, c_p$ is $\max\{t_i \times c_i\} \leq o_j t_j$ and our allocation is optimal. □

3.1.1 omplexity and se

Since, after the initialization step, $c_1 + c_2 + \ldots + c_p \geq M - p$, there are at most p steps of incrementation, so that the complexity[2] of Algorithm 3.1 is $O(p^2)$. This algorithm can only be applied to simple load balancing problems such as matrix product on a processor ring (see Section 3.2). Indeed, such a program can be decomposed into successive communication-free steps. The communication between steps is reduced to a simple shift across the ring of processes. Each step consists of a bunch of independent chunks that can be distributed using Algorithm 3.1. Consider a toy example with three processors of respective cycle-times $t_1 = 3$, $t_2 = 5$, and $t_3 = 8$. We aim to compute the product $C = A \times B$, where A and B are of size $2,496 \times 2,496$. The matrices can be decomposed into 78×78 square blocks of size 32×32 (32 is a typical block size for cache-based workstations [8]. Hence, $M = 78$ blocks of columns have to be distributed among the processors and $M = 78$ blocks of columns will be computed at each step. Table 1 applies Algorithm 3.1 to this load balancing problem. A few different steps for matrix multiplication are represented in Fig. 2. Our simple allocation is quite sufficient for matrix multiplication because each step is optimally load-balanced.

3.2 Linear Solvers

Whereas the previous solution is well-suited to matrix multiplication, it is not adapted for LU and R decompositions, as explained below. Roughly speaking, the LU decomposition algorithm works as follows on a homogeneous linear array [11]: As pointed out in Section 2.1, the preferred distribution is a `CYCLIC(b)` distribution of columns. At each step, the processor that owns the pivot block factors it and broadcasts it to all the processors, which update their remaining column blocks. At the next step, the next block of b columns becomes the pivot panel, and the computation progresses.

Because the largest fraction of the work takes place in the update, we would like to load-balance the work so that the update is best balanced. Consider the first step. After the factorization of the first block, all updates are independent chunks: Here, a chunk consists of the update of a single block of b columns. If the matrix size is $n = M \times b$, there remains $M - 1$ chunks to be updated. We can use Algorithm 3.1 to distribute these independent chunks. But, the size of the matrix shrinks as the computation goes on. At the second step, the number of blocks to update is only $M - 2$. If we want to distribute these chunks independently of the first step, redistribution of data will have to take place between the two steps and this will incur a lot of communications. Rather, in our library-oriented perspective, we search for a static allocation of column blocks to processors that will remain the same throughout the computations as the decomposition progresses. We aim at balancing the updates of all steps with the same

TABLE 1
Steps of Algorithm 3.1 for Three Processors with $t_1 = 3$, $t_2 = 5$, and $t_3 = 8$ and $M = 78$

Steps	c_1	c_2	c_3	$\max_i(c_i t_i)$
Init, m=76	39	23	14	117
m=77	40	23	14	120
m=M=78	40	24	14	120

2. Using a naive implementation. The complexity can be reduced down to $O(p \log(p))$ using ad hoc data structures.

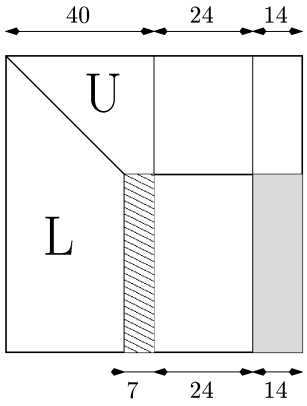


Fig. 3. Thirty-third step of LU decomposition (indices are block numbers): with the former distribution, the computation becomes less balanced. Here, after factoring block 33, processor 1 has seven updates and works for $7 \times 3 = 21$ units of time, while processor 2 works $24 \times 5 = 120$ units of time.

allocation. As illustrated in Fig. 3, we need a distribution that is somewhat repetitive (because the matrix shrinks), but not fully cyclic (because processors have different speeds).

Looking closer at the successive updates, we see that only column blocks of index $i+1$ to M are updated at step i. Hence, our objective is to find a distribution such that, for each $i \in \{2, \ldots, M\}$, the amount of blocks in $\{i, \ldots, M\}$ owned by a given processor is approximately inversely proportional to its cycle-time (proportional to its speed). To derive such a distribution, we use a dynamic programming algorithm [9], which is best explained using the former toy example.

3.3 A Dynamic Programming Algorithm

Consider an example with three processors of (relative) cycle-times $t_1 = 3$, $t_2 = 5$, and $t_3 = 8$. In Table 2, we report the allocations found by the algorithm up to $M = 10$. The entry "Selected processor" denotes the rank of the processor chosen to build the next allocation. At each step, "Selected processor" is computed so that the cost of the allocation is minimized. The cost of the allocation is computed as follows: Let us denote by c_i the number of chunks allocated to processor P_i, then the execution time for an allocation $\mathcal{C} = (c_1, c_2, \ldots, c_p)$ is $\max_{1 \le i \le p} c_i t_i$ (the maximum is taken over all processor execution times). So, the average cost to execute one chunk is

$$\widehat{\mathcal{C}} = \frac{\max_{1 \le i \le p} c_i t_i}{\sum_{i=1}^{p} c_i}.$$

For instance, at step 4, i.e., to allocate the fourth chunk, we start from the solution for three chunks, i.e., $(c_1, c_2, c_3) = (2, 1, 0)$. Which processor P_i should receive the fourth chunk, i.e., which c_i should be incremented? There are three possibilities, $(c_1 + 1, c_2, c_3) = (3, 1, 0)$, $(c_1, c_2 + 1, c_3) = (2, 2, 0)$, and $(c_1, c_2, c_3 + 1) = (2, 1, 1)$, of respective average costs $\frac{9}{4}$ (P_1 is the slowest), $\frac{10}{4}$ (P_2 is the slowest), and $\frac{8}{4}$ (P_3 is the slowest). Hence, we select $i = 3$ and we retain the solution $(c_1, c_2, c_3) = (2, 1, 1)$.

TABLE 2
Running the Dynamic Programming Algorithm with Three Processors: $t_1 = 3$, $t_2 = 5$, and $t_3 = 8$

Number of chunks	c_1	c_2	c_3	Cost	Selected processor
0	0	0	0		1
1	1	0	0	3	2
2	1	1	0	2.5	1
3	2	1	0	2	3
4	2	1	1	2	1
5	3	1	1	1.8	2
6	3	2	1	1.67	1
7	4	2	1	1.71	1
8	5	2	1	1.87	2
9	5	3	1	1.67	3
10	5	3	2	1.6	

Of course, if we are to allocate 10 chunks, we can use Algorithm 3.1 and find that five chunks should be given to processor P_1, three to P_2, and two to P_3. But, the dynamic programming algorithm returns the optimal solution for allocating any number of chunks, from one chunk up to M chunks:

Proposition 2. *The dynamic programming algorithm returns the optimal allocation for any subset of chunks $[1, s]$, where $s \le M$.*

See [9] for the proof. The complexity of the dynamic programming algorithm is $O(pM)$,[3] where p is the number of processors and M is the upper bound on the number of chunks. Note that the cost of the allocations is not a decreasing function of M.

3.4 Application to LU Decomposition

For LU decomposition, we allocate slices of B blocks of width r to processors, as illustrated in Fig. 4. B is a parameter to be discussed below. For a matrix of size $n = M \times b$, we can simply let $B = M$, i.e., define a single slice.

Within each slice, we use the dynamic programming algorithm in a "reverse" order. In other words, the kth chunk (for $1 \le k \le B$) is allocated to processor $\sigma(B - k + 1)$. Consider the toy example in Table 1 with three processors of relative speed $t_1 = 3$, $t_2 = 5$, and $t_3 = 8$. The dynamic programming algorithm allocates chunks to processors, as shown in Table 2. Hence, the obtained pattern is $(P_3 P_2 P_1 P_1 P_2 P_1 P_3 P_1 P_2 P_1)$ (see Fig. 5 for the detailed allocation within a slice). As illustrated in Fig. 4, at a given step, there are several slices of at most B chunks and the number of chunks in the first slice decreases as the computation progresses (the leftmost chunk in a slice is computed first and then there only remains $B - 1$ chunks in the slice, and so on). In the example, the reversed allocation best balances the updates in the first slice at each step: At the first step, when there are the initial 10 chunks and nine updates (the first chunk is not updated), but also at the second step, when only eight updates remain, and so on. The updating

3. As for Algorithm 3.1, the complexity can easily be improved in $O(\log(p)M)$.

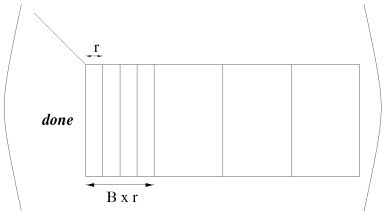


Fig. 4. Allocating slices of B chunks.

of the other slices remains well-balanced by construction since their size does not change and we keep the best allocation for $B = 10$. See Fig. 5 for the detailed allocation within a slice, together with the cost of the updates.

3. .1 Sca P on a eterogeneous inear rray

We are ready to propose an extension of the ScaLAPACK library on a heterogeneous cluster configured as a unidimensional array. The ScaLAPACK library is devoted to dense linear solvers such as LU or R factorizations. It turns out that all these solvers share the same computation unit, namely the processing of a block of b columns at a given step. They all exhibit the same control graph: The computation processes by steps; at each step, the pivot block is processed and then it is broadcast to update the remaining blocks.

The proposed data allocation is periodic: At the beginning of the computation, we distribute slices of the matrix to processors in a cyclic fashion. Each slice is composed of B chunks (blocks of b columns) and is allocated according to the previous discussion. The value of B is defined by the user and can be chosen as M if $n = M \times b$, i.e., we can define a single slice for the whole matrix. But, we can also choose a value independent of the matrix size: We may look for a fixed value, chosen from the relative processor speeds, to ensure good load-balancing.

A major advantage of a fully static distribution with a fixed parameter B is that we can use the current ScaLAPACK release with little programming effort. In the homogeneous case with p processors, we use a `CYCLIC(b)` distribution for the matrix data and we define p processes. In the heterogeneous case, we still use a `CYCLIC(b)` distribution for the data, but we define B processes which we allocate to the p physical processors according to our load-balancing strategy. The experiments reported in Section 5 fully demonstrate that this approach is quite satisfactory in practice.

4 Two-Dimensional Grids

In this section, we first summarize existing algorithms for matrix multiplication and dense linear solvers on 2D (homogeneous) grids and we discuss their extension on 2D heterogeneous grids. Section 4.3 contains our most involved results: We state the optimization problem to be solved and we prove its NP-completeness, we discuss how to find optimal solutions (with exponential cost), and we introduce an efficient heuristic.

4.1 Linear Algebra Kernels on 2D Grids

In this section, we briefly recall the algorithms implemented in the ScaLAPACK library [8] on 2D homogeneous grids, which exhibit better scalability properties than unidimensional grids because the communication operations are more uniformly distributed so that, when the physical network allows it, higher parallel efficiency can be achieved, e.g., there will be almost no difference between 1D and 2D on simple Ethernet. Then, we discuss how to modify these algorithms to cope with 2D heterogeneous grids.

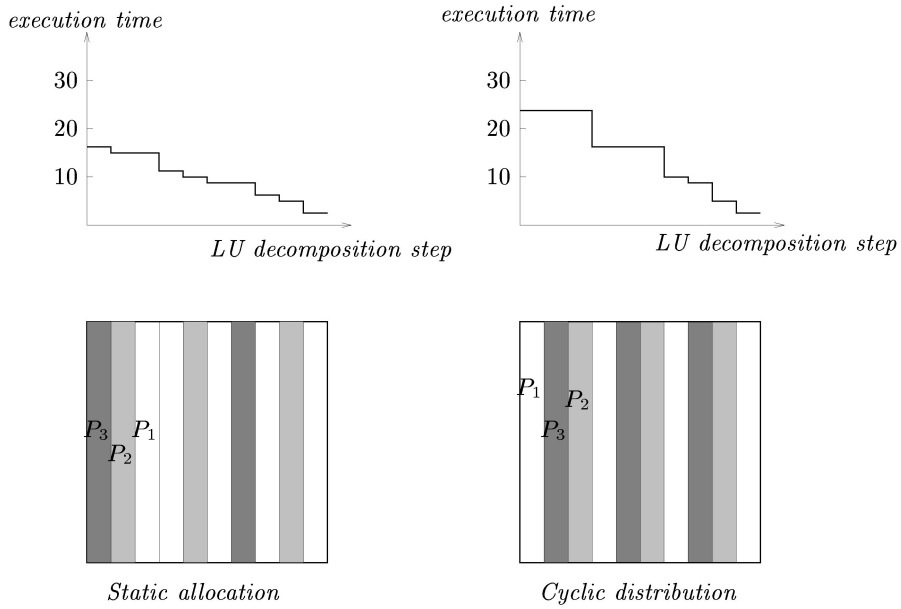


Fig. 5. Comparison of two different distributions for the LU-decomposition algorithm on a heterogeneous platform made of three processors of relative cycle-time 3, 5, and 8. The first distribution is the one given by our algorithm, the second one is the cyclic distribution. The total number of chunks is $B = 10$.

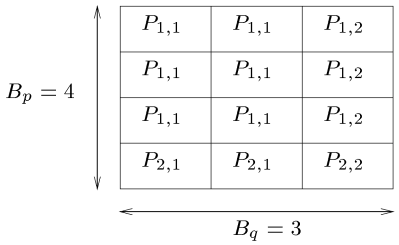


Fig. 6. A block panel with $B_p = 4$ and $B_q = 3$ for a 2×2 processors grid of respective cycle-time $t_{1,1} = 1$, $t_{1,2} = 2$, $t_{2,1} = 3$, and $t_{2,2} = 6$. The processor $P_{1,1}$ is twice as fast as the processor $P_{1,2}$, hence, it is assigned twice as many blocks within each panel.

.1.1 Matrix Multiplication

omogeneous rids. For the sake of simplicity, we focus here on the multiplication $C = AB$ of two square $n \times n$ matrices A and B. In that case, ScaLAPACK uses the outer product algorithm described in [1], [21], [30]. Consider a 2D processor grid of size $p \times q$. Assume first that $n = p = q$. In that case, the three matrices share the same layout over the 2D grid: Processor $P_{i,j}$ stores $a_{i,j}$, $b_{i,j}$, and $c_{i,j}$. Then, at each step k:

- Each processor $P_{i,k}$ (for all $i \in \{1, .., p\}$) horizontally broadcasts $a_{i,k}$ to processors $P_{i,*}$.
- Each processor $P_{k,j}$ (for all $j \in \{1, .., q\}$) vertically broadcasts $b_{k,j}$ to processors $P_{*,j}$,

so that each processor $P_{i,j}$ can independently compute $c_{i,j} = c_{i,j} + a_{i,k} \times b_{k,j}$.

A block version of this algorithm is used in the current version of the ScaLAPACK library because it is scalable, efficient, and it does not need any initial permutation (unlike Cannon's algorithm [30]). Moreover, on a homogeneous grid, broadcasts are performed as independent ring broadcasts (along the rows and the columns), hence, they can be pipelined. As already pointed out, ScaLAPACK uses a blocked version of this basic algorithm. Each matrix coefficient in the description above is replaced by a $b \times b$ square block. A level of virtualization is added: usually, the number of blocks $\lceil \frac{n}{r} \times \rceil \lceil \frac{n}{r} \rceil$ is much greater than the number of processors $p \times q$. Thus, blocks are scattered in a cyclic fashion along both grid dimensions so that each processor is responsible for updating several blocks at each step of the algorithm. In other words, ScaLAPACK uses a `CYCLIC(b)` allocation in both grid dimensions.

.1.2 eterogeneous Grids

Suppose now we have a $p \times q$ grid of heterogeneous processors. Instead of distributing the $b \times b$ matrix blocks cyclically along each grid dimension, we distribute *block panels* cyclically along each grid dimension. A block panel is a $B_p \times B_q$ rectangle of consecutive $b \times b$ blocks. See Fig. 6 for an example with $B_p = 4$ and $B_q = 3$: This panel will be distributed cyclically along both dimensions of the 2D grid. The previous cyclic dimension for homogeneous grids obviously corresponds to the case $B_p = B_q = 1$. Now, the distribution of individual blocks is no longer purely cyclic, but remains periodic. We illustrate in Fig. 7 how block panels are distributed on the 2D-grid.

$P_{1,1}$	$P_{1,1}$	$P_{1,2}$	$P_{1,1}$	$P_{1,1}$	$P_{1,2}$	$P_{1,1}$
$P_{1,1}$	$P_{1,1}$	$P_{1,2}$	$P_{1,1}$	$P_{1,1}$	$P_{1,2}$	$P_{1,1}$
$P_{1,1}$	$P_{1,1}$	$P_{1,2}$	$P_{1,1}$	$P_{1,1}$	$P_{1,2}$	$P_{1,1}$
$P_{2,1}$	$P_{2,1}$	$P_{2,2}$	$P_{2,1}$	$P_{2,1}$	$P_{2,2}$	$P_{2,1}$
$P_{1,1}$	$P_{1,1}$	$P_{1,2}$	$P_{1,1}$	$P_{1,1}$	$P_{1,2}$	$P_{1,1}$
$P_{1,1}$	$P_{1,1}$	$P_{1,2}$	$P_{1,1}$	$P_{1,1}$	$P_{1,2}$	$P_{1,1}$
$P_{1,1}$	$P_{1,1}$	$P_{1,2}$	$P_{1,1}$	$P_{1,1}$	$P_{1,2}$	$P_{1,1}$
$P_{2,1}$	$P_{2,1}$	$P_{2,2}$	$P_{2,1}$	$P_{2,1}$	$P_{2,2}$	$P_{2,1}$
$P_{1,1}$	$P_{1,1}$	$P_{1,2}$	$P_{1,1}$	$P_{1,1}$	$P_{1,2}$	$P_{1,1}$
$P_{1,1}$	$P_{1,1}$	$P_{1,2}$	$P_{1,1}$	$P_{1,1}$	$P_{1,2}$	$P_{1,1}$

Fig. 7. Allocating 4×3 panels on a 2×2 grid of processors of respective cycle-time $t_{1,1} = 1$, $t_{1,2} = 2$, $t_{2,1} = 3$, and $t_{2,2} = 6$. There are a total of 10×10 matrix blocks.

How many $b \times b$ blocks should be assigned to each processor within a panel? Intuitively, as in the case of unidimensional grids, the workload of each processor (i.e., the number of block per panel it is assigned to) should be inversely proportional to its cycle-time. In the example of Fig. 6, the allocation of the $B_p \times B_q = 4 \times 3 = 12$ blocks of the panel perfectly balances the load among the four processors.

There is an important condition to enforce when assigning blocks to processors within a block panel. We want each processor in the grid to communicate only with its four direct neighbors. This implies that each processor in a grid row is assigned the same number of matrix rows. Similarly, each processor in a grid column must be assigned the same number of matrix columns. If these conditions do not hold, additional communications will be needed, as illustrated in Fig. 8.

Translated in terms of $b \times b$ matrix blocks, the above conditions mean that each processor P_{ij}, $1 \leq j \leq q$, in the ith grid row must receive the same number r_i of blocks. Similarly, P_{ij}, $1 \leq i \leq p$, must receive c_j blocks. This condition does hold in the example of Fig. 7, where $(c_1, c_2) = (2, 1)$ and $(r_1, r_2) = (3, 1)$, hence, each processor only communicates with its direct neighbors.

Unfortunately, and in contrast with the unidimensional case, the additional constraints induced by the communication pattern may well prevent the achievement of perfect load balance amongst processors. Coming back to Fig. 6, we did achieve a perfect load balance, owing to the fact that the processor cycle-times could be arranged in the rank-1 matrix

$$\begin{pmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 3 & 6 \end{pmatrix}.$$

For instance, change the cycle-time of $P_{2,2}$ into $t_{22} = 5$. If we keep the same allocation as in Fig. 6, P_{22} remains idle every sixth time-step. Note that there is no solution to perfectly balance the work. Indeed, let r_1, r_2, c_1, and c_2 be the number of blocks assigned to each row and column grid. Processor P_{ij} computes $r_i \times c_j$ blocks in time $r_i \times c_j \times t_{ij}$.

$P_{1,1}$	$P_{1,2}$
$P_{1,1}$	$P_{2,2}$
$P_{1,1}$	$P_{1,2}$
$P_{2,1}$	$P_{1,2}$
$P_{1,1}$	$P_{1,2}$
$P_{1,1}$	$P_{1,2}$
$P_{1,1}$	$P_{1,2}$
$P_{2,1}$	$P_{1,2}$

Fig. 8. The distribution of Kalinov and Lastovetsky. Two consecutive columns are represented here. Processor have two west neighbors instead of one.

To have a perfect load balance, we have to fulfill the following equations:

$$r_1 \times t_{11} \times c_1 = r_1 \times t_{12} \times c_2 = r_2 \times t_{21} \times c_1 = r_2 \times t_{22} \times c_2$$
$$\text{that is } r_1 c_1 = 2r_1 c_2 = 3r_2 c_1 = 5r_2 c_2.$$

We derive $c_1 = 2c_2$, then $r_1 = 3r_2 = \frac{5}{2} r_2$, hence, a contradiction. Note that we have not taken into account the additional condition $(r_1 + r_2) \times (c_1 + c_2) = 12$, stating that there are 12 blocks within a block panel: It is impossible to perfectly load-balance the work, whatever the size of the panel.

If we relax the constraints on the communication pattern, we can achieve a perfect load-balance as follows: First, we balance the load in each processor column independently (using the unidimensional scheme); next, we balance the load between columns (using the unidimensional scheme again, weighting each column by the inverse of the harmonic mean of the processors cycle-times within the column, see below). This is the "heterogeneous block cyclic distribution" of Kalinov and Lastovetsky [28], which leads to the solution of Fig. 8. Because processor $P_{2,2}$ has two west neighbors instead of one, at each step of the algorithm, it is involved in two horizontal broadcasts instead of one.

We use the example to explain, with further detail, how the heterogeneous block cyclic distribution of Kalinov and Lastovetsky [28] works. First, they balance the load in each processor column independently, using the unidimensional scheme. In the example, there are two processors in the first grid column with cycle-times $t_{11} = 1$ and $t_{21} = 3$, so P_{11} should receive three times more matrix rows than P_{21}. Similarly, for the second grid column, P_{12} (cycle-time $t_{12} = 2$) should receive five out of every seven matrix rows, while P_{22} (cycle-time $t_{22} = 5$) should receive the remaining two rows. Next, how to distribute matrix columns? The first grid column operates as a single processor of cycle-time $2\frac{1}{1+\frac{1}{3}} = \frac{3}{2}$. The second grid column operates as a single processor of cycle-time $2\frac{1}{\frac{1}{2}+\frac{1}{5}} = \frac{20}{7}$. So, out of every 61 matrix columns. we assign 40 to the first processor column and 21 to the second processor column.

Because we do not want to rebuild ScaLAPACK from scratch, we do not want the number of horizontal and vertical communications to depend upon the data distribution. For large grids, the number of horizontal neighbors of a given processor cannot be bounded a priori if we use Kalinov and Lastovetsky's approach. We enforce the grid communication pattern (each processor only communicates with its four direct neighbors) to minimize communication overhead. The price to pay is that we have to solve a difficult optimization problem to load-balance the work as efficiently as possible and that our algorithm may not lead to perfect load balancing because of the topology constraint. Solving this optimization problem is the objective of Section 4.3. In [5], more general data distribution strategies are proposed which lead to perfect load balancing while minimizing communication overhead. Those schemes nevertheless require writing completely new routines for linear algebra kernels and cannot rely on top of ScaLAPACK.

4.2 The LU and R Decompositions

We first recall the ScaLAPACK algorithm for the LU or R decompositions on a homogeneous 2D-grid. We discuss next how to implement them on a heterogeneous 2D-grid.

.2.1 omogeneous Grids

In this section, we briefly review the direct parallelization of the right-looking variant of the LU decomposition. We assume that the matrix A is distributed onto a two-dimensional grid of (virtual) homogeneous processors. We use a `CYCLIC(b)` decomposition in both dimensions. The right-looking variant is naturally suited to parallelization and can be briefly described as follows: Consider a matrix A of order $n = M \times b$ and assume that the LU factorization of the $k \times b$ first columns has been proceeded with $0 \leq k \leq M - 1$. During the next step, the algorithm factors the next panel of b columns, pivoting if necessary. Next, the pivots are applied to the remainder of the matrix. The lower trapezoid factor just computed is broadcast to the other process columns of the grid using an increasing-ring topology so that the upper trapezoid factor can be updated via a triangular solve. This factor is then broadcast to the other process rows using a minimum spanning tree topology so that the remainder of the matrix can be updated by a rank-b update. This process continues recursively with the updated matrix. In other words, at each step, the current panel of columns is factored into L and the trailing submatrix $\bar{A}$ is updated. The key computation is this latter rank-b update, $\bar{A} \leftarrow \bar{A} - LU$, which can be implemented as follows:

1. The column processor that owns L broadcasts it horizontally (so there is a broadcast in each processor row).
2. The row processor that owns U broadcasts it vertically (so there is a broadcast in each processor column).
3. Each processor locally computes its portion of the update.

$B_q = 6$

$B_p = 8$

$P_{1,1}$	$P_{1,2}$	$P_{1,1}$	$P_{1,1}$	$P_{1,2}$	$P_{1,1}$
$P_{2,1}$	$P_{2,2}$	$P_{2,1}$	$P_{2,1}$	$P_{2,2}$	$P_{2,1}$
$P_{1,1}$	$P_{1,2}$	$P_{1,1}$	$P_{1,1}$	$P_{1,2}$	$P_{1,1}$
$P_{1,1}$	$P_{1,2}$	$P_{1,1}$	$P_{1,1}$	$P_{1,2}$	$P_{1,1}$
$P_{1,1}$	$P_{1,2}$	$P_{1,1}$	$P_{1,1}$	$P_{1,2}$	$P_{1,1}$
$P_{2,1}$	$P_{2,2}$	$P_{2,1}$	$P_{2,1}$	$P_{2,2}$	$P_{2,1}$
$P_{1,1}$	$P_{1,2}$	$P_{1,1}$	$P_{1,1}$	$P_{1,2}$	$P_{1,1}$
$P_{1,1}$	$P_{1,2}$	$P_{1,1}$	$P_{1,1}$	$P_{1,2}$	$P_{1,1}$

Fig. 9. Allocation of the blocks within a block panel with $B_p = 8$ and $B_q = 6$ for a grid of processors of respective cycle-time $t_{1,1} = 1$, $t_{1,2} = 2$, $t_{2,1} = 3$, and $t_{2,2} = 5$.

The communication volume is thus reduced to the broadcast of the two row and column panels and matrix $\bar{A}$ is updated in place (this is known as an outer-product parallelization). Load balance is very good. The simplicity of this parallelization, as well as its expected good performance, explains why the right-looking variants have been chosen in ScaLAPACK [10]. See [18], [10], [7] for a detailed performance analysis of the right-looking variants, which demonstrates their good scalability property. The parallelization of the R decomposition is analogous [12], [11].

.2.2 eterogeneous Grids

For the implementation of the LU and R decomposition algorithms on a heterogeneous 2D grid, we modify the ScaLAPACK `CYCLIC(b)` distribution very similarly as for the matrix multiplication problem. The intuitive reason is the following: As pointed out before, the core of the LU and R decompositions is a rank-b update, hence, the load-balancing techniques for the outer-product matrix algorithm naturally apply. However, periodic distributions must be used to take into account the shrinking of the matrix during the elimination.

We still use block panels made up of several $b \times b$ matrix blocks. The block panels are distributed cyclically along both dimensions of the grid. The only modification is that the order of the blocks within a block panel becomes important. Consider the previous example with four processors laid along a 2×2 grid as follows:

$$T = \begin{pmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{pmatrix} = \begin{pmatrix} 1 & 2 \\ 3 & 5 \end{pmatrix}.$$

Say we use a panel with $B_p = 8$ and $B_q = 6$, i.e., a panel composed of 48 blocks. Using the methods described below (see Section 4.3), we assign the blocks as follows, corresponding to the approximation of T by a rank-1 matrix $\left(\begin{smallmatrix} 1 & 2 \\ 3 & 6 \end{smallmatrix}\right)$:

- Within each panel column, the first processor row receives six blocks and the second processor rows receives two blocks ($(r_1, r_2) = (6, 2)$).
- Out of the six panel columns, the first grid column receives four and the second grid column receives two of them ($(c_1, c_2) = (4, 2)$).

This allocation is represented in Fig. 9. We need to explain how we have allocated the six panel columns. For the matrix multiplication problem, the ordering of the blocks within the panel was not important because all of the processors execute the same number of (independent) computations at each step of the algorithm. For the LU and R decomposition algorithms, the ordering of columns and rows is quite important. In the example, the first processor column operates like six processors of cycle-time 1 and two processors of cycle-time 3, which is equivalent to a single processor $P_{*,1}$ of cycle-time $\frac{3}{20}$; the second processor column operates like six processors of cycle-time 2 and two processors of cycle-time 5, which is equivalent to a single processor $P_{*,2}$ of cycle-time $\frac{5}{17}$. The unidimensional algorithm allocates the six panel columns as $(P_{*,1}P_{*,2}P_{*,1}P_{*,1}P_{*,2}P_{*,1})$. Identically, the vertical allocation pattern is $(P_{1,*}P_{2,*}P_{1,*}P_{1,*}P_{1,*}P_{2,*}P_{1,*}P_{1,*})$ and we retrieve the allocation of Fig. 9.

To conclude this section, we have a difficult load-balancing problem to solve. First, we do not know which is the best layout of the processors, i.e., how to arrange them to build an efficient 2D grid. In some cases (rank-1 matrices), we are able to load-balance the work perfectly, but, in most cases, this is not possible. Next, once the grid is built, we have to determine the number of blocks that are assigned to each processor within a block panel. Again, this must be done so as to load-balance the work because processors have different speeds. Finally, the panels are cyclically distributed along both grid dimensions. The rest of the paper is devoted to a solution to this difficult load-balancing problem.

4.3 The 2D Heterogeneous Grid Allocation Problem

.3.1 Problem Statement and Formulation

Consider n processors $P_1, P_2, \ldots, P_n$ of respective cycle-times $t_1, t_2, \ldots, t_n$. The problem is to arrange these processors along a two-dimensional grid of size $p \times q \leq n$ in order to compute the product $C = AB$ of two $N \times N$ matrices as fast as possible. We need some notations to formally state this objective.

Consider a given arrangement of $p \times q \leq n$ processors along a two-dimensional grid of size $p \times q$. Let us renumber the processors as P_{ij}, with cycle-time t_{ij} (with $1 \leq i \leq p$ and $1 \leq j \leq q$). Assume that processor P_{ij} is assigned a block of r_i rows and c_j columns of data elements, meaning that it is responsible for computing $r_i \times c_j$ elements of the C matrix: see Fig. 10 for an example.

There are two (equivalent) ways to compute the efficiency of the grid:

- Processor P_{ij} is scheduled to evaluate rectangular data block $r_i \times c_j$ of the matrix C, which it will process within $r_i \times c_j \times t_{ij}$ units of time. The total execution time t_{exe} is taken over all processors:

$$t_{exe} = \max_{i,j}\{r_i \times t_{ij} \times c_j\}.$$

  t_{exe} must be normalized to the average time t_{ave} needed to process a single data element: Since there are a total of N^2 elements to compute, we enforce that $\sum_{i=1}^{p} r_i = N$ and that $\sum_{j=1}^{q} c_j = N$. We get

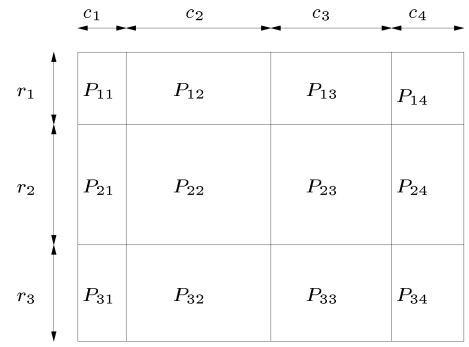


Fig. 10. Allocating computations to processors on a 3×4 grid.

$$t_{ave} = \frac{\max_{i,j}\{r_i \times t_{ij} \times c_j\}}{\left(\sum_{i=1}^{p} r_i\right) \times \left(\sum_{j=1}^{q} c_j\right)}.$$

We are looking for the minimum of this quantity over all possible integer values r_i and c_j. We can simplify the expression for t_{ave} by searching for (nonnegative) rational values r_i and c_j which sum up to 1 (instead of N):

$$\textbf{Objective } Obj_1 : \min_{(\sum_i r_i=1; \sum_j c_j=1)} \max_{i,j}\{r_i \times t_{ij} \times c_j\}.$$

Given the rational values r_i and c_j returned by the solution of the optimization problem Obj_1, we scale them by the factor N to get the final solution. We may have to round up some values, but we do so while preserving the relation $\sum_{i=1}^{p} r_i = \sum_{j=1}^{q} c_j = N$. Stating the problem as Obj_1 renders its solution generic, i.e., independent of the parameter N.

- Another way to tackle the problem is the following: What is the largest number of data elements that can be computed within one time unit? Assume again that each processor P_{ij} of the $p \times q$ grid is assigned a block of r_i rows and c_j columns of data elements. We need to have $r_i \times t_{ij} \times c_j \leq 1$ to ensure that P_{ij} can process its block within one cycle. Since the total number of data elements being processed is $(\sum_{i=1}^{p} r_i) \times (\sum_{j=1}^{q} c_j)$, we get the (equivalent) optimization problem:

$$\textbf{Objective } Obj_2 : \max_{r_i \times t_{ij} \times c_j \leq 1} \left\{ \left(\sum_i r_i\right) \times \left(\sum_j c_j\right) \right\}.$$

Again, the rational values r_i and c_j returned by the solution of the optimization problem Obj_2 can be scaled and rounded to get the final solution.

Although there are $p+q$ variables r_i and c_j, there are only $p+q-1$ degrees of freedom: If we multiply all r_is by the same factor λ and divide all c_j by λ, nothing changes in Obj_2. In other words, we can impose $r_1 = 1$, for instance, without loss of generality.

We can further manipulate Obj_2 as follows:

$$\begin{aligned}
&\max_{r_i \times t_{ij} \times c_j \leq 1} \left\{ \left(\sum_{i=1}^{p} r_i\right) \times \left(\sum_{j=1}^{q} c_j\right) \right\} \\
&= \max_{r_i} \left\{ \max_{c_j \text{ with } r_i \times t_{ij} \times c_j \leq 1} \left\{ \left(\sum_{i=1}^{p} r_i\right) \times \left(\sum_{j=1}^{q} c_j\right) \right\} \right\} \\
&= \max_{r_i} \left\{ \left(\sum_{i=1}^{p} r_i\right) \times \max_{c_j \text{with } r_i \times t_{ij} \times c_j \leq 1} \left\{ \left(\sum_{j=1}^{q} c_j\right) \right\} \right\} \\
&= \max_{r_i} \left\{ \left(\sum_{i=1}^{p} r_i\right) \times \max_{\forall i, c_j \leq \frac{1}{r_i \times t_{ij}}} \left\{ \left(\sum_{j=1}^{q} c_j\right) \right\} \right\} \\
&= \max_{r_i} \left\{ \left(\sum_{i=1}^{p} r_i\right) \times \left(\sum_{j=1}^{q} \min_i \left\{ \frac{1}{r_i \times t_{ij}} \right\}\right) \right\} \\
&= \max_{r_i} \left\{ \left(\sum_{i=1}^{p} r_i\right) \times \left(\sum_{j=1}^{q} \frac{1}{\max_i\{r_i \times t_{ij}\}}\right) \right\}.
\end{aligned}$$

We obtain an expression with only p variables (and $p-1$ degrees of freedom). This last expression does not look very friendly, though. Solving this optimization problem, optimally or through a heuristic, is the main objective of the following sections.

The 2D Load-Balancing Problem. In the next section, we give a solution to the 2D load-balancing problem which can be stated as follows: Given $n = p \times q$ processors, how do we arrange them along a 2D grid of size $p \times q$ so as to optimally load-balance the work of the processors for the matrix multiplication problem? Note that solving this problem will, in fact, lead to the solution of many linear algebra problems, including dense linear system solvers.

The problem is even more difficult to tackle than the optimization problem stated above because we do not assume the processors arrangement as given. We search among all possible arrangements (layouts) of the $p \times q$ processors as a $p \times q$ grid and, for each arrangement, we must solve the optimization problem Obj_1 or Obj_2.

.3.2 P- ompleteness

In this section, we formally state the previous optimization problem and prove its NP-completeness. In the presentation, processor speeds come in more handy than cycle-times: If processor P_i has (relative) cycle-time t_i, we use its speed $s_i = 1/t_i$ rather than t_i. We start with the definition of the 2D load-balancing problem, where the mapping f represents the processor arrangement along the grid and the r_i and c_j are the variables used in Obj_1:

Definition 1. *MA -GRID(s): Given p^2 real positive numbers $s_1, \ldots, s_{p^2}$, find*

$$r_1, \ldots, r_p, c_1, \ldots, c_p,$$

and a one-to-one mapping f from $[1,p] \times [1,p]$ to $[1,p^2]$ so that

$$\forall (i,j) \in [1,p] \times [1,p], \quad r_i c_j \leq s_{f(i,j)}$$

and $(\sum_{i=1}^{p} r_i)(\sum_{j=1}^{p} c_j)$ is maximal.

The decision problem associated to the optimization problem MAX-GRID is the following:

Definition 2. *MA -GRID(s, K): Given p^2 real positive numbers $s_1, \ldots, s_{p^2}$ and a real positive number K, does there exist*

$$r_1, \ldots, r_p, c_1, \ldots, c_p,$$

and a one-to-one mapping f from $[1,p] \times [1,p]$ to $[1,p^2]$ so that

$$\forall (i,j) \in [1,p] \times [1,p], \quad r_i c_j \leq s_{f(i,j)}$$

and

$$\left(\sum_{i=1}^{p} r_i\right)\left(\sum_{j=1}^{p} c_j\right) \geq K.$$

Theorem 1. *MA -GRID(s, K) is NP-complete.*

Proof. The proof is lengthy and technical. We use several lemmas. First, we select the following NP-complete problem for the reduction:

Lemma 1.

$$2P\text{-}eq \leq_P MAX\text{-}GRID,$$

where 2P-eq is defined as follows:

Definition 3. *2-Partition-Equal (2P-eq). Given a set of p integers, $\mathcal{A} = \{a_1, \ldots, a_p\}$, is there a partition of $\{1, \ldots, p\}$ into two subsets $\mathcal{A}_1$ and $\mathcal{A}_2$ such that*

$$\sum_{i \in \mathcal{A}_1} a_i = \sum_{i \in \mathcal{A}_2} a_i \text{ and } card(\mathcal{A}_1) = card(\mathcal{A}_2) \ ?$$

Since *2P-eq* is known to be NP-Complete [22], Lemma 1 will complete the proof of Theorem 1.

(a) Reduction $2P\text{-}eq \leq_P$ MAX-(s, K). Here, we consider an arbitrary instance of the 2-Partition-Equal problem, i.e., a set $\mathcal{A} = \{a_1, \ldots, a_{2n}\}$ of $2n$ integers. We have to polynomially transform this instance into an instance of the MAX-GRID problem which has a solution iff the original instance of 2-Partition-Equal has one solution.

Define $\{b_1, \ldots, b_{2n}\}$ as $\forall i, b_i = (a_i + 2n \max_k a_k)$. Thus, $2n \max a_i \leq b_i \leq (2n+1) \max a_i$. We build the instance of the MAX-GRID problem (denoted by MAX-GRID$(b_1, \ldots, b_{2n}, K)$), where

$$\begin{cases} K &= (4n \max a_i)^2 + \sum_{i=1}^{2n} b_i + \frac{(\sum_{i=1}^{2n} b_i)^2}{4(4n \max a_i)^2}, \\ s_1 &= (4n \max a_i)^2, \\ s_{i+1} &= b_i, \ \forall i, \ 1 \leq i \leq 2n, \\ s_i &= 1, \ \forall i, \ 2n+2 \leq i \leq (n+1)^2. \end{cases}$$

In what follows, we show that a solution is necessarily as depicted in Fig. 11, where the restriction of f to $([2, n+1], 1) \bigcup (1, [2, n+1])$ defines a one-to-one mapping with $[2, 2n+1]$ and

$$\begin{cases} & r_1 c_1 &= (4n \max a_i)^2, \\ \forall i, \ 2 \leq i \leq n+1, & r_i &= \frac{b_{\sigma(i,1)}}{c_1}, \\ \forall j, \ 2 \leq j \leq n+1, & c_j &= \frac{b_{\sigma(1,j)}}{r_1}, \end{cases}$$

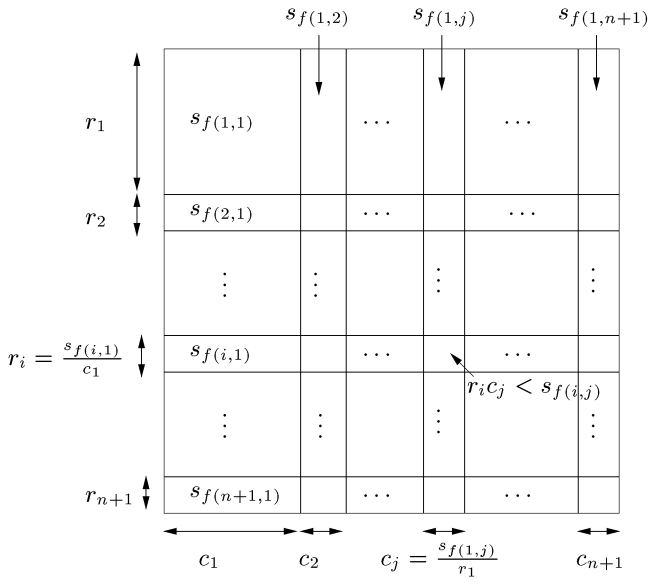


Fig. 11. Solution of MAX-GRID$(b_1, b_2, \ldots, b_{2n}, K)$.

Thus, we can check that

$$\left(\sum_{i=1}^{p} r_i\right)\left(\sum_{j=1}^{p} c_j\right) \geq K \Longleftrightarrow \sum_{2}^{n+1} b_{f(i,1)} = \sum_{2}^{n+1} b_{f(1,j)}.$$

The intuitive reason is the following: Since s_1 is much larger than other areas, the best choice is r_1 and c_1 so that $r_1 c_1 = s_1$. Similarly, because $b_i \gg 1$, it would be better to put the b_is on the same row or the same column as s_1. Otherwise, one of the b_is would be in the middle of the grid and the area assigned to the corresponding processor would be much smaller than the one it is able to process. Finally, the area of the grid will be maximal when the grid is balanced, i.e., if the b_is satisfy *2P-eq*. All these points will be made clear in the proof below.

Lemma 2. *If $(\sum_{i=1}^{p} r_i)(\sum_{j=1}^{p} c_j) \geq K$, then the area assigned to the processor of speed s_1 is at least $11n^2 \max a_i{}^2$.*

Proof. Up to a permutation, suppose that $f^{-1}(1) = (1,1)$, i.e., that the area assigned to the processor of speed s_1 is $r_1 c_1$. Since $\sum_{i \geq 2} s_i \leq 5n^2 \max a_i{}^2$ and because

$$\left(\sum_{i=1}^{p} r_i\right)\left(\sum_{j=1}^{p} c_j\right) \leq r_1 c_1 + \sum_{i \geq 2} s_i,$$

we have

$$r_1 c_1 \geq 11n^2 \max a_i{}^2.$$

□

Lemma 3. *Let f be any one-to-one mapping from $[1, n+1] \times [1, n+1]$ to $[1, (n+1)^2]$ s.t. $f(1,1) = 1$. Moreover, let us suppose that $r_1 c_1 \geq 11n^2 \max a_i{}^2$ (Lemma 2). Then, $(\sum_{i=1}^{p} r_i)(\sum_{j=1}^{p} c_j)$ is maximal if and only if*

$$\begin{cases} \forall i \geq 2, & r_i c_1 = s_{\sigma(i,1)}, \\ \forall j \geq 2, & c_j r_1 = s_{\sigma(1,j)}. \end{cases}$$

Proof. By definition,

$$\begin{cases} \forall i \geq 2, & r_i c_1 \leq s_{\sigma(i,1)} \\ \forall j \geq 2, & c_j r_1 \leq s_{\sigma(1,j)}, \end{cases}$$

and $(\sum_{i=1}^{p} r_i)(\sum_{j=1}^{p} c_j)$ is maximal when each r_i and c_j is maximal.

Moreover, if

$$\begin{cases} \forall i \geq 2, & r_i c_1 = s_{\sigma(i,1)} \\ \forall j \geq 2, & c_j r_1 = s_{\sigma(1,j)}, \end{cases}$$

then all other conditions $r_i c_j \leq s_{f(i,j)}$ are automatically fulfilled. Indeed,

$$\forall (i,j) \neq (1,1), \quad 1 \leq s_{f(i,j)} \leq (2n+1) \max a_i$$

and, thus,

$$\begin{aligned} r_i c_j &= \frac{s_{f(i,1)} s_{f(1,j)}}{x_1 y_1}, \\ &\leq \frac{(2n+1)^2 \max {a_i}^2}{11 n^2 \max {a_i}^2} \\ &\leq 1 \\ &\leq s_{f(i,j)}. \end{aligned}$$

□

Therefore, f being given, the maximal value for $(\sum_{i=1}^{p} r_i)(\sum_{j=1}^{p} c_j)$ is

$$S(f) = \left(r_1 + \frac{\sum_2^{n+1} s_{f(i,1)}}{c_1} \right) \left(c_1 + \frac{\sum_2^{n+1} s_{f(1,j)}}{r_1} \right).$$

Let us set

$$p = r_1 c_1, \; S_1 = \sum_2^{n+1} s_{f(i,1)} \text{ and } S_2 = \sum_2^{n+1} s_{f(1,j)}.$$

Thus,

$$S(f) = p + (S_1 + S_2) + \frac{S_1 S_2}{p}.$$

Lemma 4.

$S(f) \geq K \Longleftrightarrow (p = (4n \max a_i)^2)$ and $\left(S_1 = S_2 = \dfrac{\sum b_i}{2} \right)$

and, therefore, $S(f) \geq K$ if and only if there exists a solution to the original instance of the 2P-eq problem.

Proof. By construction, $S_1 + S_2 \leq \sum b_i$ and, therefore, $S_1 S_2 \leq \frac{(\sum b_i)^2}{4}$. Moreover, $S_1 S_2 = \frac{(\sum b_i)^2}{4}$ if and only if $S_1 = S_2 = \sum b_i$. Thus,

$$S(f) \leq p + \sum b_i + \frac{(\sum b_i)^2}{4p}.$$

Let us consider the function g defined by

$$g(p) = p + \frac{(\sum b_i)^2}{4p}.$$

This function is a decreasing rather than increasing function of p and reaches the minimum value for $p = \frac{\sum b_i}{2}$. Since

$$\frac{\sum b_i}{2} \gg 11 n^2 \max {a_i}^2 \leq p \leq (4n \max a_i)^2,$$

g is maximal for $p = (4n \max a_i)^2$. In other words,

$$\begin{cases} S(\sigma) \leq (4n \max a_i)^2 + \sum b_i + \frac{(\sum b_i)^2}{4(4n \max a_i)^2} = K \\ S(\sigma) = K \text{ if and only if } (p = (4n \max a_i)^2) \\ \text{and } (S_1 = S_2 = \frac{\sum b_i}{2}). \end{cases}$$

□

Thus, MAX-GRID$(b_1, \ldots, b_{2n}, K)$ has a solution iff $2P\text{-}eq(b_1, \ldots, b_{2n})$ has a solution and, therefore, iff the original instance of $2P\text{-}eq(a_1, \ldots, a_{2n})$ has a solution.

(b) Conciseness of the Transformation. The last element of the proof is to prove that our instance of the MAX-GRID problem has a size polynomial in the size of the original instance of the *2P-eq* problem.

Lemma 5. *Define $MAX = \max_k a_k$ as above and let $c(a)$ and $c(b)$ denote, respectively, the encoding of the data a and b. Then,*

$$Length(c(b)) = O(Length(c(a))^2).$$

Proof.

$$\begin{aligned} &\text{Length}(c(a)) = \sum_k \log(a_k) \geq \log(\text{MAX}) + (n-1) \log(\min_k a_k) \\ &\geq (n-1) \log 2 + \log \text{MAX}. \\ &\text{Length}(c(b)) = \sum_k \log(b_k) = \sum_k \log\Big(2n \; \text{MAX}\Big(1 + \frac{a_k}{n\text{MAX}}\Big)\Big) \\ &\leq 1 + n(\log n + \log 2) + n \log \text{MAX}. \end{aligned}$$

Therefore,

$$\text{Length}(c(b)) = O(\text{Length}(c(a))^2).$$

□

This achieves the proof of the NP-completeness of MAX-GRID. □

.3.3 Searching for the Optimal Solution

For small grid sizes, we may want to search for the optimal solution even though the previous result shows that an exponential cost is unavoidable (unless P = NP). We start with a useful result to reduce the number of arrangements to be searched. Next, we derive an algorithm to solve the optimization problem Obj_1 or Obj_2 for a fixed (given) arrangement. Despite the reduction, we still have an exponential number of arrangements to search for. Even worse, for a fixed arrangement, our algorithm exhibits an

exponential cost. Therefore, we shall introduce a heuristic in the next section, in order to give a fast but suboptimal solution to the 2D load-balancing problem.

Reduction to Nondecreasing rrangements. In this section, we show that we do not have to consider all the possible arrangements; instead, we reduce the search to "nondecreasing arrangements". A nondecreasing arrangement on a $p \times q$ grid is defined as follows: In every grid row, the cycle-times are increasing: $t_{ij} \leq t_{i,j+1}$ for all $1 \leq j \leq q-1$. Similarly, in every grid column, the cycle-times are increasing: For all $1 \leq i \leq p-1$, $t_{ij} \leq t_{i+1,j}$.

Proposition 3. *There exists a nonincreasing arrangement which is optimal.*

Proof. The proof works as follows:

1. Let the pq cycle-times be denoted as $t_1, t_2, \ldots, t_{pq}$.
2. Consider an optimal arrangement on the grid of size $p \times q$. Note that there is no reason that the optimal arrangement must be a nondecreasing arrangement.
3. Show that some well-chosen "correct" transpositions can be applied to the arrangement while preserving the optimality of the solution. The correct transpositions will make the arrangement "closer" to a nondecreasing arrangement.
4. Show that the number of steps to reach a nondecreasing arrangement is finite.

We need a few definitions:

Definition 4.

- ***(Arrangement)*** *An arrangement is a one-to-one mapping*

$$\sigma : \begin{pmatrix} [1,pq] & \mapsto & [1,p] \times [1,q] \\ k & \rightarrow & \sigma(k) = (i,j) \end{pmatrix}$$

  which assigns a position to each processor in the grid.
- ***(Nondecreasing arrangement)*** *An arrangement σ is nondecreasing if $t_{\sigma^{-1}(i,j)} \leq t_{\sigma^{-1}(i',j')}$ for all $(1,1) \leq (i,j) \leq (i',j') \leq (p,q)$.*
- ***(Correct transposition)*** *Let σ be an arrangement. If $\sigma(k) < \sigma(l)$ and $t_k > t_l$, the transposition $\tau(k,l)$ which transposes the values of $\sigma(k)$ and $\sigma(l)$ is said to be correct.*

Given any arrangement σ, there exists a suite of correct transpositions that modifies σ into a nondecreasing arrangement. To prove this, we use a *weight function* W that quantifies the distance to the "nondecreasingness." We will show that a correct transposition will decrease the weight of the arrangement it is applied to. We choose

$$W(\sigma) = \sum_{i,j} t_{\sigma^{-1}(i,j)} \times (p+q-i-j).$$

Let us check that, for each correct transposition τ, $W(\tau(\sigma)) < W(\sigma)$. Let σ be an arrangement such that $(i,j) \leq (i',j')$ and $t_{\sigma^{-1}(i,j)} > t_{\sigma^{-1}(i',j')}$. Let $k = \sigma^{-1}(i,j)$ and $l = \sigma^{-1}(i',j')$: By hypothesis, the transposition $\tau = \tau(k,l)$ is correct. Let $\sigma' = \tau \circ \sigma$. We have

$$\begin{aligned} W(\sigma') &= W(\sigma) + (t_{\sigma^{-1}(i',j')} - t_{\sigma^{-1}(i,j)})(p+q-i-j) \\ &\quad + (t_{\sigma^{-1}(i,j)} - t_{\sigma^{-1}(i',j')})(p+q-i'-j') \\ &= W(\sigma) - ((i'-i)+(j'-j))(t_{\sigma^{-1}(i,j)} - t_{\sigma^{-1}(i',j')}) \\ &< W(\sigma). \end{aligned}$$

Consider an optimal arrangement σ and let $r_1, \ldots, r_p$ and $c_1, \ldots, c_q$ be the solution to the optimization problem Obj_1. Since an equivalent solution is obtained by transposing two columns or two rows of the arrangement, we can assume that $r_1 \geq r_2 \geq \ldots r_p$ and $c_1 \geq c_2 \geq \ldots c_q$. If σ is nondecreasing, we are done. Otherwise, there exists $(1,1) \leq (\alpha,\beta) < (p,q)$ such that either $t_{\sigma^{-1}(\alpha+1,\beta)} < t_{\sigma^{-1}(\alpha,\beta)}$ or $t_{\sigma^{-1}(\alpha,\beta+1)} < t_{\sigma^{-1}(\alpha,\beta)}$. The proof is the same in both cases, hence, assume that $t_{\sigma^{-1}(\alpha+1,\beta)} < t_{\sigma^{-1}(\alpha,\beta)}$. Let $k = \sigma^{-1}(\alpha,\beta)$ and $l = \sigma^{-1}(\alpha+1,\beta)$: By hypothesis, the transposition $\tau = \tau(k,l)$ is correct. Let $\sigma' = \tau \circ \sigma$. We want to show that σ' is as good as σ (in the sense of Obj_2), so we need to show that, for all i and j, $r_i c_j t_{\sigma'^{-1}(i,j)} \leq 1$.

Indeed, we have $r_\alpha \geq r_{\alpha+1}$ and $t_{\sigma^{-1}(\alpha+1,\beta)} < t_{\sigma^{-1}(\alpha,\beta)}$, hence,

$$\begin{cases} r_\alpha c_\beta t_{(\tau\circ\sigma)^{-1}(\alpha,\beta)} = r_\alpha c_\beta t_{\sigma^{-1}(\alpha+1,\beta)} \leq r_\alpha c_\beta t_{\sigma^{-1}(\alpha,\beta)} \leq 1 \\ r_{\alpha+1} c_\beta t_{(\tau\circ\sigma)^{-1}(\alpha+1,\beta)} = r_{\alpha+1} c_\beta t_{\sigma^{-1}(\alpha,\beta)} \leq r_\alpha c_\beta t_{\sigma^{-1}(\alpha,\beta)} \leq 1. \end{cases}$$

Therefore, σ' is optimal, too, and $W(\sigma') < W(\sigma)$. If σ' is not nondecreasing, we repeat the process, which converges in a finite number of steps, because there is a finite number of weight values. □

Spanning Trees for a iven rrangement. In this section, we show how to solve the optimization problem Obj_1 or Obj_2 for a given arrangement. For small size problems, all the possible nondecreasing arrangements can be generated, hence, we have an exponential but feasible solution to the 2D load balancing problem. Let σ be a given arrangement on a $p \times q$ grid and let $(r_1, \ldots, r_p, c_1, \ldots, c_q)$ be the solution to the optimization problem Obj_1.

Consider the optimization problem Obj_1. We have to maximize the quadratic expression $(\sum_{1 \leq i \leq p} r_i)(\sum_{1 \leq j \leq q} c_j)$ under $p \times q$ inequalities $r_i t_{ij} c_j \leq 1$. We have $p+q-1$ degrees of freedom. The objective of this section is to show that, for at least $p+q-1$, inequalities are in fact equalities. We use a graph-oriented approach to this purpose.

We consider the following complete bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, composed on one side of p vertices labeled with r_i and the other side by q vertices labeled with c_j. The weight of the edge (r_i, c_j) is t_{ij}. Given a spanning tree $\mathcal{T} = (\mathcal{V}, \mathcal{E}')$ of the graph $\mathcal{G}$, if we start from $r_1 = 1$, we can (uniquely) determine all the values of the r_i and c_j by following the edges of $\mathcal{T}$, enforcing the equalities

$$\forall (r_i, c_j) \in \mathcal{E}', \quad r_i t_{i,j} c_j = 1.$$

The spanning tree $\mathcal{T}$ is said to be *acceptable* if and only if all the remaining inequalities are satisfied: $\forall (r_i, c_j) \in \mathcal{E}, \quad r_i t_{i,j} c_j \leq 1$. The value of an acceptable spanning tree is $(\sum r_i)(\sum c_j)$. We claim that the solution of Obj_1 is obtained with the acceptable spanning tree of maximal value.

This leads to the following algorithm:

- We generate recursively all the spanning trees of $\mathcal{G}$ of root r_1.
- Iteratively, we add vertices to the tree and, by enforcing the condition $r_i t_{ij} c_j = 1$, we evaluate the corresponding labels.
- Let us define *an acceptable subtree* as a subgraph of an acceptable tree. Then, during the recursive process, it might be possible to detect that a subtree is not acceptable so that all the trees are not built.

Finally, we select the acceptable tree that maximizes $\sum_{1\leq i\leq p} r_i \sum_{1\leq j\leq q} c_j$.

Correctness of the lgorithm. To justify the previous algorithm, consider an optimal solution to Obj_1 and draw the bipartite graph $\mathcal{U} = (\mathcal{V}, \mathcal{E}')$ corresponding to the equalities: $\mathcal{U}$ has $p+q$ vertices labeled with r_i and c_j. There is an edge between vertices r_i and c_j ($(r_i, c_j) \in \mathcal{E}'$) if and only if $r_i c_j t_{i,j} = 1$. Let us show that this graph is connected, so suppose it is not.

Suppose (without any loss of generality) that $\mathcal{V}' = \{r_1, \ldots, r_{p'}, c_1, \ldots, c_{q'}\}$ is a connected component of $\mathcal{U}$ and that $p' < p$.

First, suppose that $q' = q$. It means that, for all $1 \leq j \leq q$, $r_{p'+1} c_j t_{p'+1,j} < 1$. Then, $r_{p'+1}$ can be increased by a factor of α, with $\alpha = \min_{1\leq j\leq q} \frac{1}{r_{p'+1} c_j t_{p'+1,j}}$, and the so-built solution is strictly better (since the product $\sum_{1\leq i\leq p} r_i \sum_{1\leq j\leq q} c_j$ is increased), which contradicts the optimality of the initial solution.

Consequently, $q' < q$ and $p' < p$. It means that, for all

$$(p'+1, 1) \leq (i, j) \leq (p, q')$$

and for all

$$(1, q'+1) \leq (i, j) \leq (p', q), \quad r_i c_j t_{i,j} < 1.$$

Let

$$\begin{cases} \alpha_r &= \min_{(i>p' \text{ and } j\leq q')} \frac{1}{r_i c_j t_{i,j}} \\ \alpha_c &= \min_{(i\leq p' \text{ and } j>q')} \frac{1}{r_i c_j t_{i,j}}. \end{cases}$$

We also introduce the notations

$$\begin{cases} R_a &= \sum_{i\leq p'} r_i \\ R_b &= \sum_{i>p'} r_i \\ C_a &= \sum_{j\leq q'} c_j \\ C_b &= \sum_{j>q'} c_j. \end{cases}$$

Now, we have two possibilities to increase the connectivity of the graph: Either increase the elements corresponding to R_b by a factor of α_r (but decrease the elements corresponding to C_b so as to maintain the acceptable solution) or increase the elements corresponding to C_b by a factor of α_c (but decrease the element corresponding to R_b by a factor $\frac{1}{\alpha_c}$). As we will see, at least one of these solutions does increase the product $\sum_{1\leq i\leq p} r_i \sum_{1\leq j\leq q} c_j = (R_a + R_b)(C_a + C_b)$, which contradicts the optimality hypothesis of the initial solution.

Indeed, consider the function f defined by $f(\lambda) = (\lambda R_b + R_a)(\frac{C_b}{\lambda} + C_a)$ Note that $f'(1) = R_b C_a - R_a C_b$. Moreover, f is a continuous function that is first strictly decreasing to a minimum and then strictly increasing. Therefore:

- If $f'(1) \geq 0$, then, for all $\lambda \geq 1$, $f(\lambda) > f(1)$. In particular, $f(\alpha_r) > f(1)$.
- If $f'(1) \leq 0$, then, for all $\lambda \leq 1$, $f(\lambda) > f(1)$. In particular, $f(\frac{1}{\alpha_c}) > f(1)$.

One of the previous two solutions will indeed increase the connectivity of $\mathcal{U}$ while preserving the objective function $\sum_{1\leq i\leq p} r_i \sum_{1\leq j\leq q} c_j$. We conclude that there does exist an acceptable spanning tree whose value is the optimal solution.

In summary, given an arrangement, we are able to compute the solution to the optimization problem. The cost is exponential because there is an exponential number of spanning trees to check for acceptability. Still, our method is constructive and can be used for problems of limited size.

Case of a 2×2 rid. For small size grids, we can find analytical solutions to the optimization problem. As an example, we explicitly show the solution for a 2×2 grid. In that case, we want to maximize (see the definition of Obj_2) the quantity $(r_1 + r_2)(\frac{1}{\max(r_1 t_{11}, r_2 t_{21})} + \frac{1}{\max(r_1 t_{12}, r_2 t_{22})})$. We normalize our problem by letting $r_1 = 1$ and $r_2 = r$. We have to maximize

$$(1+r)\left(\frac{1}{max(t_{11}, r t_{21})} + \frac{1}{max(t_{12}, r t_{22})}\right).$$

There are three cases to study:

- *First case* $0 \leq r \leq min(\frac{t_{11}}{t_{21}}, \frac{t_{12}}{t_{22}})$: In this interval, the value of the expression varies as an increasing function of r. So, the maximum on this interval is obtained for $r = min(\frac{t_{11}}{t_{21}}, \frac{t_{12}}{t_{22}})$.
- *Second case* $max(\frac{t_{11}}{t_{21}}, \frac{t_{12}}{t_{22}}) \leq r \leq +\infty$: In this interval, the expression varies as a decreasing function of r. So, the maximum is obtained for $r = max(\frac{t_{11}}{t_{21}}, \frac{t_{12}}{t_{22}})$.
- *Third case* $min(\frac{t_{11}}{t_{21}}, \frac{t_{12}}{t_{22}}) \leq r \leq max(\frac{t_{11}}{t_{21}}, \frac{t_{12}}{t_{22}})$: By symmetry, we can suppose that $\frac{t_{11}}{t_{21}} \leq \frac{t_{12}}{t_{22}}$. So, our expression is now $(1+r)(\frac{1}{r t_{21}} + \frac{1}{t_{12}})$. This function is first decreasing and then increasing. Hence, the maximum is obtained on the bounds of the interval, i.e., for $r = \frac{t_{11}}{t_{21}}$ or $r = \frac{t_{12}}{t_{22}}$.

In conclusion, there are two possible values for the maximum, namely $r = \frac{t_{11}}{t_{21}}$ or $r = \frac{t_{12}}{t_{22}}$. For the objective function, we obtain the value

$$\max\left(\left(1 + \frac{t_{11}}{t_{21}}\right)\left(\frac{1}{t_{11}} + \frac{1}{\max(t_{12}, \frac{t_{11} t_{22}}{t_{21}})}\right),\right.$$
$$\left.\left(1 + \frac{t_{12}}{t_{22}}\right)\left(\frac{1}{\max(t_{11}, \frac{t_{12} t_{21}}{t_{22}})} + \frac{1}{t_{12}}\right)\right).$$

Rank-1 Matrices. If the matrix $(t_{ij})_{1\leq i\leq p, 1\leq j\leq q}$ is a rank-1 matrix, then the optimal arrangement for the 2D load-balancing problem is easy to determine. Assume without

loss of generality that $t_{11} = 1$. We let $r_1 = c_1 = 1$, $r_i = \frac{1}{t_{i1}}$ for $2 \leq i \leq p$ and $c_j = \frac{1}{t_{1j}}$ for $2 \leq i \leq q$. All the $p \times q$ inequalities $r_i t_{ij} c_j$ are equalities, which means that all processors are fully utilized:

$$r_i t_{ij} c_j = \frac{1}{t_{i1}} t_{ij} \times \frac{1}{t_{1j}} = 1,$$

because the 2×2 determinant

$$\begin{vmatrix} t_{11} & t_{1j} \\ t_{i1} & t_{ij} \end{vmatrix}$$

is zero (with $t_{11} = 1$). No idle time occurs with such a solution; the load-balancing is perfect.

Unfortunately, given $p \times q$ integers, it is very difficult to know whether they can be arranged into a rank-1 matrix of size $p \times q$. If such an arrangement does not exist, we can intuitively say that the optimal arrangement is the "closest one" to a rank-1 matrix.

.3. euristic Solution

In this section, we study a heuristic solution to find an arrangement of the processors and a solution to the corresponding optimization problem that leads to good load-balancing. As pointed out above, if the processors cycle-times can be arranged into a rank-1 matrix, it is easy to compute the r_is and the c_js so that no idle time occurs. In general, this is not possible. Nevertheless, basic linear algebra techniques enable us to find both a reasonable arrangement (close to a rank-one matrix) and reasonable values for the r_is and the c_js (so that the idle time is low).

The heuristic works as follows: First, it determines a reasonable (nondecreasing) arrangement matrix for processor cycle-times. Then, it computes approximate values for the r_is and the c_js corresponding to this arrangement matrix. Finally, an iterative refinement of the arrangement matrix is proposed which computes a new arrangement matrix which better fits with the values of the r_is and the c_js computed during the second step.

nitial rrangement of the Processors. We are given the processors cycle-times as input to the heuristic. Our aim is to find an arrangement so that the resulting matrix is close to a rank-one matrix. We arrange the processors cycle-times in the matrix T as follows:

$$\begin{cases} \forall i, \ \forall 1 \leq j < q, & t_{i,j} \leq t_{i,j+1} \\ \forall 1 \leq i < p, & t_{i,q} \leq t_{i+1,1}. \end{cases}$$

For instance, if we consider the case of nine processors with cycle-times $(1, 2, \ldots, 9)$, the arrangement matrix T that we obtain is

$$T = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}.$$

This arrangement is not optimal, but it is nondecreasing. We will refine it using SVD techniques [23] in order to obtain a better arrangement.

Computing the r_is and the c_js. As noticed above, the computation of the r_is and the c_js is obvious when T is a rank-1 matrix. The SVD decomposition provides the best approximation (in the sense of the 2-norm) of a given matrix by a rank-one matrix. It is therefore natural to use it to find the best approximation of the arrangement matrix defined above and then to compute the r_is and the c_js corresponding to this approximation.

Let us denote by $U \Sigma V^t = S$ the singular value decomposition of S, where $S = (\frac{1}{t_{i,j}})_{i,j}$. The best approximation of S by a rank-1 matrix is given by sab^t, where a and b are, respectively, the left and right singular vectors associated with s, the largest singular value of S. Thus, if we set

$$\begin{cases} \forall i, & r_i = s a_i \\ \forall j, & c_j = b_j, \end{cases}$$

we can expect that

$$\forall i, j, \qquad r_i t_{i,j} c_j \simeq 1.$$

We consider T^{inv} rather than T since the approximation by the rank-1 matrix is usually better on the largest components. This way, we better approximate the components of T corresponding to processors with low time cycle.

In order to ensure that the inequalities $r_i t_{i,j} c_j \leq 1$ are fulfilled, we divide each c_j by the largest component of the th column of the matrix $r_i t_{i,j} c_j$. Then, in order to avoid idle time, we divide each r_i by the largest component of the ith row of the matrix $r_i t_{i,j} c_j$. Thus, we obtain two vectors r and c satisfying $\forall i, j, \ r_i t_{i,j} c_j \leq 1$ and such that

$$\begin{cases} \forall i, \exists j & r_i t_{i,j} c_j = 1 \\ \forall j, \exists i & r_i t_{i,j} c_j = 1. \end{cases}$$

Consider again the matrix

$$T = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}.$$

We obtain

$$r = \begin{pmatrix} 1.1661 \\ 0.3675 \\ 0.2100 \end{pmatrix}, c = \begin{pmatrix} 0.6803 \\ 0.4288 \\ 0.2859 \end{pmatrix},$$

and

$$T_{exe} = (r_i t_{i,j} c_j)_{i,j} = \begin{pmatrix} 0.7933 & 1 & 1 \\ 1 & 0.7879 & 0.6303 \\ 1 & 0.7203 & 0.5402 \end{pmatrix}.$$

The value of the objective function $(\sum r_i)(\sum c_j)$ is 2.4322.

Note that this allocation can easily be improved by using the process described in "*Correctness of the algorithm*" of Section 4.3.3.

terative Refinement. In this section, we propose an iterative refinement in order to obtain a better arrangement of the processors (and a better solution to the corresponding optimization problem). In order to determine the new arrangement matrix, we compute the matrix $T^{opt} = (\frac{1}{r_i \, c_j})_{i,j}$. T^{opt} is a rank-1 matrix whose components are the processor cycle-times that are optimal with respect to the vectors r and c computed above. In the case of our example, we obtain

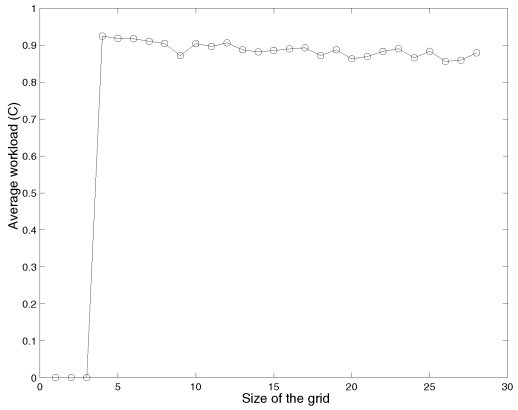


Fig. 12. Evaluation of the allocation. Comparison with the absolute lower bound.

$$T^{opt} = \begin{pmatrix} 1.2606 & 2.0000 & 3.0000 \\ 4.0000 & 6.3464 & 9.5195 \\ 7.0000 & 11.1061 & 16.6592 \end{pmatrix}.$$

This suggests the choice of another arrangement matrix T for the processors' cycle-times that better fits to the matrix T^{opt}. More precisely, we derive the new matrix T from the following conditions:

$$\forall i,j,k,l, \quad t_{i,j} \leq t_{k,l} \Longleftrightarrow t^{opt}_{i,j} \leq t^{opt}_{k,l}.$$

Then, we compute the r_is and the c_js as shown in the previous paragraph and we restart the process. We consider that the process has converged when no modification occurs in the matrix T. In our example, after the second step, the arrangement matrix T becomes

$$T = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 7 \\ 6 & 8 & 9 \end{pmatrix}$$

and the value of the objective function $(\sum r_i)(\sum c_j)$ is 2.5065 (instead of 2.4322). Convergence is obtained after three steps. The value of the objective function $(\sum r_i)(\sum c_j)$ is then 2.5889 and the corresponding arrangement matrix is

$$T = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 6 & 8 \\ 5 & 7 & 9 \end{pmatrix}.$$

Simulation Results. In this section, we present the results of our algorithm for processors with random cycle-times in $[0,1]$. We consider the arrangement of n^2 processors into an $n \times n$ grid. Fig. 12 displays the evolution with n of the objective function

$$\widehat{C}(r_1, r_2, \ldots, r_p, c_1, \ldots, c_q) = \frac{(\sum_{i=1}^{p} r_i)(\sum_{j=1}^{q} c_j)}{\sum_{i,j} \frac{1}{t_{ij}}}.$$

Indeed, during one unit of time:

- The amount of work done using this allocation $\widehat{C}(r_1, r_2, \ldots, r_p, c_1, \ldots, c_q)$ is $(\sum_{i=1}^{p} r_i)(\sum_{j=1}^{q} c_j)$.
- The maximal amount of work that each processor P_{ij} could do (not necessarily reachable) is $\frac{1}{t_{ij}}$.

Fig. 13 displays the evolution with n of the ratio

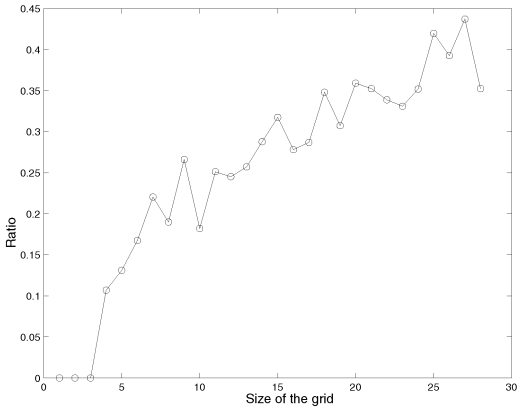


Fig. 13. Evolution of the ratio τ.

$$\tau = \frac{((\sum r_i)(\sum c_j))_{\text{after convergence}}}{((\sum r_i)(\sum c_j))_{\text{after the first step}}} - 1.$$

Finally, Fig. 14 displays the evolution with n of the average number of steps necessary to reach convergence.

Concluding Remarks on the euristic Solution. The heuristic solution gives satisfying results. Nevertheless, the algorithm does not converge to an optimal solution with respect to the optimization problem. Moreover, it seems that the number of steps of the iterative process grows with n and, therefore, involves more than $O(n^3)$ flops to arrange n^2 processors into an $n \times n$ grid. Nevertheless, one usually obtains satisfying results after only a few steps.

5 MPI Experiments

5.1 Estimating Processor Speed

There are too many parameters to accurately predict the actual speed of a machine for a given program, even assuming that the machine load will remain the same throughout the computation. Cycle-times must be understood as *normalized cycle-times* [13], i.e., application-dependent elemental computation times, which are to be computed via small-scale experiments (repeated several times, with an averaging of the results).

We ranked the computers to be used with respect to performance by measuring the time necessary to multiply two matrices of order 500 in double precision arithmetic.

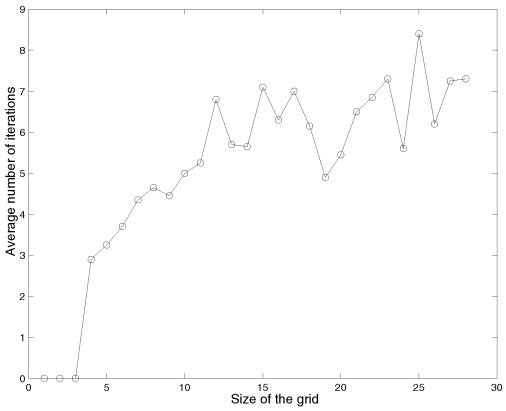


Fig. 14. Number of iterations.

TABLE 3
Specifications of the Nine Heterogeneous Processors

Name	Mhz	500x500 Matrix Multiply (Mflops)
texas	507	362
alabama	507	357
mississipi	498	357
onyx	431	305
cotnari	351	250
cuervo	200	134
muddy-waters	402	287
pink-floyd	402	284
petit-gris	200	128

This sorting criterion is certainly adequate for parallel dense linear algebra such as the LU and R decompositions studied in this paper. On the Intel Pentium processor-based computers, we used the ATLAS [37] software, which automatically generates an efficient BLAS implementation for this architecture in particular. For example, on a 500 Mhz Pentium III computer, ATLAS 3.0 achieves a performance of 362 Mflops for multiplying two matrices of order 500. The higher level of cache was flushed between every timing measure and we used the operating system wallclock timer. On the Sun Sparc workstations, we used the BLAS library supplied by the vendor (sunperf) and we similarly measured their performance. Table 3 lists the computers we used in our experiments, their peak performance, and the performance they achieved with respect to our ranking test.

5.2 Numerical Results

In this section, we compare the classical block cyclic distribution of ScaLAPACK with the heterogeneous 2D distribution that we propose. The tests have been performed both on matrix multiplication and R decomposition, using the nine processors listed in Table 3. The speed of the slowest processor is 128, while the speed of the fastest processor is 362. The ratio $\frac{362}{128} \approx 2.8$ shows that the processor set is reasonably heterogeneous: In fact, it mixes machines that are two years older than others.

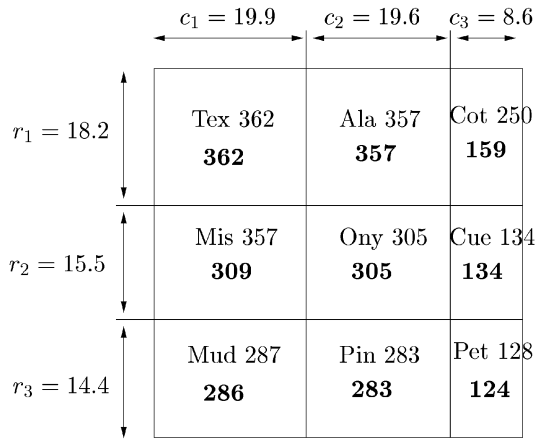


Fig. 15. Heterogeneous distribution. Bold entries represent the actual work allocated to the processors.

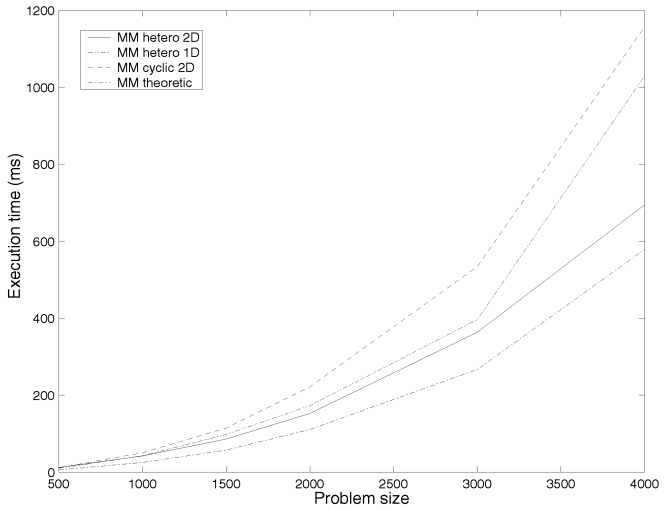


Fig. 16. Comparison of different allocations for matrix multiplication using nine processors.

For the largest experiment with nine processors, the heuristic of Section 4.3.4 leads to the distribution depicted in Fig. 15. This distribution theoretically enables us to update $(\sum_i r_i) \times (\sum_j c_j) = 2318.44$ elements at each time step. With the classical homogeneous distribution, the whole process is slowed down to the pace of the slowest processor and, therefore, we can only expect to update $9 \times 128 = 1,152$ elements at each time step. The theoretical improvement is therefore $\frac{2318.44}{1152} = 2.01$.

We display in Figs. 16 and 17 the time of matrix multiplication and R decomposition for different problem sizes. There are three experimental plots in each figure, corresponding to our unidimensional heterogeneous distribution, our two-dimensional heterogeneous distribution, and the ScaLAPACK two-dimensional block-cyclic distribution. We also report the theoretical execution time expected for the two-dimensional heterogeneous distribution (if communication costs could be neglected). For each kernel, the heterogeneous strategy on a 2D grid is better

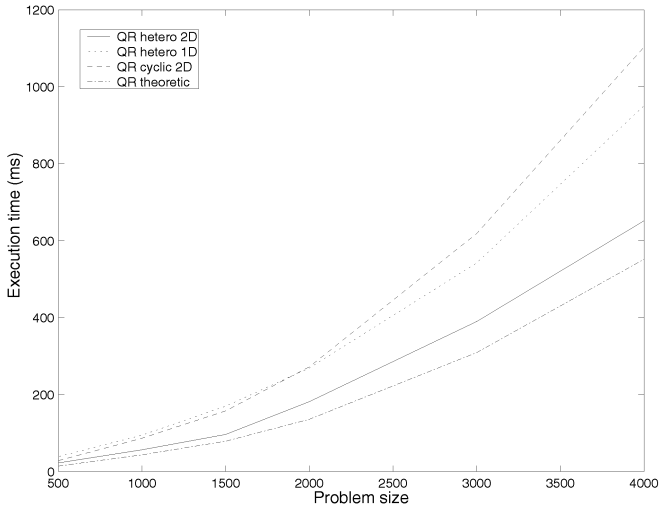


Fig. 17. Comparison of different allocations for R decomposition using nine processors.

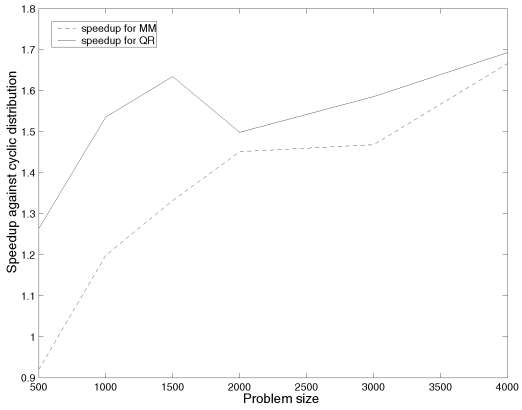


Fig. 18. Evolution of the speedup against cyclic distribution with the matrix size for matrix multiplication and R decomposition using nine processors. The theoretical upper bound is 2.01.

than the heterogeneous strategy for a unidimensional grid, which in turn is better than with the block-cyclic distribution on a 2D grid.

Of course, the results displayed in Figs. 16 and 17 show that this theoretical speedup is not reached. This is due to the communications involved in the algorithms. All the computations have been performed with a slow network (100-base Ethernet) and the cost of communications is important, especially when involved matrices are small. Nevertheless, we can observe in Fig. 18 that the largest the involved matrices, the closer the actual speedup to the theoretical improvement.

6 Conclusion

In this paper, we have discussed data allocation strategies to implement matrix products and dense linear system solvers on heterogeneous computing platforms. Such platforms are likely to play an important role in the future. We have shown both theoretically and experimentally (through MPI experiments) that our data allocation algorithms were quite satisfactory. They form the basis for a successful extension of the ScaLAPACK library to heterogeneous platforms.

We point out that extending the standard ScaLAPACK block-cyclic distribution to heterogeneous 2D grids has turned out to be surprisingly difficult. In most cases, a perfect balancing of the load between all processors cannot be achieved and deciding how to arrange the processors along the 2D grid is a challenging NP-complete problem. But, we have formally stated the optimization problem to be solved and we have presented both an exact solution (with exponential cost) and a heuristic solution.

We believe that the original motivation for this work, namely the extension of ScaLAPACK, will prove very important in the (much larger) context of metacomputing: Dense linear algebra algorithms are the prototype of tightly coupled kernels that need to be implemented efficiently on collections of distributed and heterogeneous platforms: We view them as a perfect testbed before experimenting on more challenging computational problems on the grid. Future work will indeed be related to using *collections* of heterogeneous clusters rather than a single one. Implementing linear algebra kernels on several collections of workstations or parallel servers, scattered all around the world and connected through fast but nondedicated links, would give rise to a "Computational Grid ScaLAPACK." Our results constitute a first step toward achieving this ambitious goal.

Acknowledgments

The authors thank the reviewers whose comments and suggestions have greatly improved the presentation of the paper. Antoine Petitet was a member of the Innovative Computing Laboratory, Computer Science Department, University of Tennessee at Knoxville while this work was conducted.

References

[1] R. Agarwal, F. Gustavson, and M. Zubair, "A High Performance Matrix Multiplication Algorithm on a Distributed-Memory Parallel Computer, Using Overlapped Communication," *IBM J. Research and Development*, vol. 38, no. 6, pp. 673-681, 1994.

[2] S. Anastasiadis and K.C. Sevcik, "Parallel Application Scheduling on Networks of Workstations," *J. Parallel and Distributed Computing*, vol. 43, pp. 109-124, 1997.

[3] D. Arapov, A. Kalinov, A. Lastovetsky, and I. Ledovskih, "A Parallel Language and Its Programming System for Heterogeneous Networks," *Concurrency: Practice and Experience*, vol. 12, no. 13, pp. 1317-1343, 2000.

[4] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, *Complexity and Approximation.* Berlin: Springer, 1999.

[5] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert, "Partitioning a Square into Rectangles: Np-Completeness and Approximation Algorithms," Technical Report RR-2000-10, LIP, ENS Lyon, Feb. 2000.

[6] F. Berman, "High-Performance Schedulers," *The Grid: Blueprint for a New Computing Infrastructure*, I. Foster and C. Kesselman, eds., pp. 279-309, Morgan-Kaufmann, 1999.

[7] L. Blackford, J. Choi, A. Cleary, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley, "Scalapack: A Portable Linear Algebra Library for Distributed-Memory Computers—Design Issues and Performance," *Proc. Supercomputing '96*, 1996.

[8] L.S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley, *ScaLAPACK Users' Guide.* SIAM, 1997.

[9] P. Boulet, J. Dongarra, Y. Robert, and F. Vivien, "Static Tiling for Heterogeneous Computing Platforms," *Parallel Computing*, vol. 25, pp. 547-568, 1999.

[10] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley, "ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers—Design Issues and Performance," *Computer Physics Comm.*, vol. 97, pp. 1-15, 1996 (also LAPACK Working Note 95).

[11] J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker, and R.C. Whaley, "The Design and Implementation of the ScaLAPACK LU, R, and Cholesky Factorization Routines," *Scientific Programming*, vol. 5, pp. 173-184, 1996.

[12] E. Chu and A. George, " R Factorization of a Dense Matrix on a Hypercube Multiprocessor," *SIAM J. Scientific and Statistical Computing*, vol. 11, pp. 990-1028, 1990.

[13] M. Cierniak, M.J. Zaki, and W. Li, "Scheduling Algorithms for Heterogeneous Network of Workstations," *The Computer J.*, vol. 40, no. 6, pp. 356-372, 1997.

[14] P.E. Crandall, "The Limited Applicability of Block Decomposition in Cluster Computing," *Proc. Fourth IEEE Int'l Symp. High Performance Distributed Computing*, pp. 102-109, 1995.

[15] P.E. Crandall and M.J. uinn, "Block Data Decomposition for Data-Parallel Programming on a Heterogeneous Workstation Network," *Proc. Second Int'l Symp. High Performance Distributed Computing*, pp. 42-49, 1993.

[16] P. Crescenzi and V. Kann, "A Compendium of NP Optimization Problems," http://www.nada.kth.se/~viggo/wwwcompendium/wwwcompendium.html, 2001.
[17] D.E. Culler and J.P. Singh, *Parallel Computer Architecture: A Hardware Software Approach.* San Francisco: Morgan Kaufmann, 1999.
[18] J. Dongarra, R. van de Geijn, and D. Walker, "Scalability Issues in the Design of a Library for Dense Linear Algebra," *J. Parallel and Distributed Computing,* vol. 22, no. 3, pp. 523-537, 1994.
[19] J.J. Dongarra and D.W. Walker, "Software Libraries for Linear Algebra Computations on High Performance Computers," *SIAM Review,* vol. 37, no. 2, pp. 151-180, 1995.
[20] *The Grid: Blueprint for a New Computing Infrastructure,* I. Foster and C. Kesselman, eds. Morgan-Kaufmann, 1999.
[21] G. Fox, S. Otto, and A. Hey, "Matrix Algorithms on a Hypercube I: Matrix Multiplication," *Parallel Computing,* vol. 3, pp. 17-31, 1987.
[22] M.R. Garey and D.S. Johnson, *Computers and Intractability, a Guide to the Theory of NP-Completeness.* W.H. Freeman, 1991.
[23] G.H. Golub and C.F. Van Loan, *Matrix Computations,* second ed. Johns Hopkins, 1989.
[24] T.F. Gonzalez and S. Zheng, "Approximation Algorithm for Partitioning a Rectangle with Interior Points," *Algorithmica,* vol. 5, pp. 11-42, 1990.
[25] M. Grigni and F. Manne, "On the Complexity of the Generalized Block Distribution," *Proc. Parallel Algorithms for Irregularly Structured Problems, Third Int'l Workshop IRREGULAR '96,* pp. 319-326, 1996.
[26] M. Iverson and F. Özgüner, "Dynamic, Competitive Scheduling of Multiple Dags in a Distributed Heterogeneous Environment," *Proc. Seventh Heterogeneous Computing Workshop,* 1998.
[27] M. Kaddoura, S. Ranka, and A. Wang, "Array Decompositions for Nonuniform Computational Environments," *J. Parallel and Distributed Computing,* vol. 36, no. 2, pp. 91-105, 1996.
[28] A. Kalinov and A. Lastovetsky, "Heterogeneous Distribution of Computations while Solving Linear Algebra Problems on Networks of Heterogeneous Computers," *Proc. HPCN Europe 1999,* P. Sloot, M. Bubak, A. Hoekstra, and B. Hertzberger, eds., pp. 191-200, 1999.
[29] S. Khanna, S. Muthukrishnan, and M. Paterson, "On Approximating Rectangle Tiling and Packing," *Proc. Ninth Ann. ACM-SIAM Symp. Discrete Algorithms,* pp. 384-393, 1998.
[30] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing.* Benjamin/Cummings, 1994.
[31] A. Lingas, R.Y. Pinter, R.L. Rivest, and A. Shamir, "Minimum Edge Length Partitioning of Rectilinear Polygons," *Proc. 20th Ann. Allerton Conf. Comm., Control, and Computing,* 1982.
[32] M. Maheswaran and H.J. Siegel, "A Dynamic Matching and Scheduling Algorithm for Heterogeneous Computing Systems," *Proc. Seventh Heterogeneous Computing Workshop,* 1998.
[33] H.J. Siegel, H.G. Dietz, and J.K. Antonio, "Software Support for Heterogeneous Computing," *ACM Computing Surveys,* vol. 28, no. 1, pp. 237-239, 1996.
[34] G.C. Sih and E.A. Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures," *IEEE Trans. Parallel and Distributed Systems,* vol. 4, no. 2, pp. 175-187, Feb. 1993.
[35] M. Tan, H.J. Siegel, J.K. Antonio, and Y.A. Li, "Minimizing the Application Execution Time through Scheduling of Subtasks and Communication Traffic in a Heterogeneous Computing System," *IEEE Trans. Parallel and Distributed Systems,* vol. 8, no. 8, pp. 857-871, Aug. 1997.
[36] J.B. Weissman and X. Zhao, "Scheduling Parallel Applications in Distributed Networks," *Cluster Computing,* vol. 1, no. 1, pp. 109-118, 1998.
[37] R.C. Whaley and J. Dongarra, "Automatically Tuned Linear Algebra Software," *Proc. Supercomputing '98,* 1998.

Olivier Beaumont obtained the PhD thesis from the Université de Rennes in January 1999. He is currently an associate professor in the Computer Science Laboratory LIP at ENS Lyon. His main research interests are parallel algorithms on distributed memory architectures.

incent Boudet is currently a PhD student in the Computer Science Laboratory LIP at ENS Lyon. He is mainly interested in algorithm design and in compilation-parallelization techniques for distributed memory architectures.

Antoine Petitet received the Engineer of Computer Science degree from ENSEEIHT Toulouse, France, in 1990. He was awarded the PhD degree in computer science from the University of Tennessee, Knoxville, in 1996. He is currently a benchmark engineer at Sun France. His research interests primarily focus on parallel computing, numerical linear algebra and the design of parallel numerical software libraries for distributed memory concurrent computers.

Fabrice Rastello obtained the PhD degree from the Ecole Normale Supérieure de Lyon in September 2000. He is currently working as an engineer for ST Microelectronics in Grenoble. His main research interests are parallel algorithms for distributed memory architectures and automatic compilation/parallelization techniques.

Yves Robert obtained the PhD degree from the Institut National Polytechnique de Grenoble in January 1986. He is currently a full professor in the Computer Science Laboratory LIP at ENS Lyon. He is the author of three books, 60 papers published in international journals, and 70 papers published in international conferences. His main research interests are parallel algorithms for distributed memory architectures and automatic compilation/parallelization techniques. He is a member of the ACM and the IEEE.

Appendix E

Ordonnancement sur plates-formes hétérogènes

Les trois papiers (tous publiés dans IEEE TPDS) sont plus proches de la problématique générale de cette habilitation.

Le premier (”The Master-Slave Paradigm with Heterogeneous Processors”) présente certains des résultats évoqués au Chapitre 2 et démontre la difficulté de quelques problèmes élémentaires de distribution de tâches quand on cherche à minimiser le temps de complétion.

Le second (”Scheduling Strategies for Master-Slave Tasking on Heterogeneous Processor Platforms”) présente les premiers travaux que nous avons conduits sur la distribution de tâches indépendantes (sur des plates-formes en arbre, essentiellement).

Le troisième (”Scheduling Divisible Loads on Star and Tree Networks: Results and Open Problems”) est un article de synthèse sur les tâches divisibles. Son principal intérêt est de démontrer (dans le Paragraphe 3) quelques résultats théoriques souvent énoncés et utilisés dans la littérature, mais dont nous n’avons jamais trouvé la preuve auparavant!

The Master-Slave Paradigm with Heterogeneous Processors

Olivier Beaumont, Arnaud Legrand, and Yves Robert, *Member*, *IEEE*

Abstract—In this paper, we revisit the master-slave tasking paradigm in the context of heterogeneous processors. We assume that communications are handled by a bus and, therefore, at most one communication can take place at a given time step. We present a polynomial algorithm that gives the optimal solution when a single communication is needed before the execution of the tasks on the slave processors. When communications are required both before and after the processing of the tasks, we show that the problem is strongly NP-Complete. In this case, we present a guaranteed approximation algorithm. Finally, we present asymptotically optimal algorithms when communications are required before the processing of each task, or both before and after the processing of each task.

Index Terms—Heterogeneous processors, master-slave tasking, communication, matching, complexity.

1 Introduction

Master-slave tasking is a simple yet widely used technique to execute independent tasks under the centralized supervision of a control processor. In the standard implementation of master-slave, the tasks are executed by identical processors (the slaves). We revisit the master-slave paradigm in the framework of heterogeneous computing resources: Slave processors have different computation speeds. We present several scenarios to model the communication pattern between the master and the slaves. In all cases, such communications will take place in exclusive mode on a dedicated hardware resource (such as a serial bus), i.e., *at most one communication will take place between the master and one slave at any time step*.

To give a single motivation, this framework applies to any Monte Carlo simulation where large numbers of identical and independent simulations are run for different values of the random number generator seed. Monte Carlo simulations are widely used in various areas such as cellular microphysiology [20], reactor simulations [21], or studies on conformations of proteins [19].

The rest of the paper is organized as follows: In Section 2, we state four different variants of the master-slave problem: 1) with communications only before dispatching the tasks, 2) with communications both before and after processing the tasks, 3) with communication before each task processing, and 4) with communication both before and after each task processing. We give, in Section 3, a polynomial time algorithm that solves the first problem. The second problem is intrinsically more difficult and we prove in Section 4 that it is strongly NP-Complete; we also prove a guaranteed approximation algorithm for the second problem in Section 4, and we report some simulation results. We present asymptotically optimal algorithms when communications are required before each task (third problem) in Section 5, and when communications are required both before and after each task (fourth problem) in Section 6. We briefly survey related work in Section 7. Finally, we give some remarks and conclusions in Section 8.

- *A. Legrand and Y. Robert are with LIP, UMR CNRS-ENS Lyon-INRIA 5668, Ecole Normale Superieure de Lyon, 69364 Lyon Cedex 07, France. E-mail: {Arnaud.Legrand, Yves.Robert}@ens-lyon.fr.*
- *O. Beaumont is with LaBRI, UMR CNRS 5800, Domaine Universitaire, 33405 Talence Cedex, France. E-mail: Olivier.Beaumont@labri.fr.*

Manuscript received 1 Apr. 2001; revised 22 July 2002; accepted 31 Mar. 2003.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 113908.

2 Problem Statement

The target architectural platform is represented in Fig. 1. The master M and the p slaves $P_1, P_2, \ldots, P_p$ communicate through a shared medium, typically a bus, that can be accessed only in exclusive mode. In this context, at a given time-step, at most one slave processor can communicate with the master, either to receive data or to send results back.

We assume that there is a pool of independent tasks to be processed by the p slaves. All tasks are of same-size, that is, they represent the same amount of processing. Tasks are considered to be atomic (execution cannot be preempted once initiated). Processors are heterogeneous; more precisely, slave P_i requires t_i units of time to process a single task. We say that t_i is the "cycle-time" of processor P_i. Each P_i will execute c_i tasks (where c_i is to be determined) from the pool. Regardless of the hypotheses concerning communication costs, there are two (related) optimization problems:

MinTime(C). Given a total number of tasks C, determine the best allocation of tasks to slaves, i.e., the allocation $\mathcal{C} = \{c_1, c_2, \ldots, c_p\}$ s.t. $\sum_{i=1}^{p} c_i = C$ and which minimizes the total execution time.

MaxTasks(T). Given a time bound T, determine the best allocation of tasks to slaves, i.e., the allocation $\mathcal{C} = \{c_1, c_2, \ldots, c_p\}$ s.t. all processors complete their execution within T units of time and $\sum_{i=1}^{p} c_i = C$ is maximized.

In the paper, we concentrate on solving the second problem MaxTasks(T). Given the solution to this problem, we find a solution to MinTime(C) by using binary search on T, calling MaxTasks(T) several times, and returning the

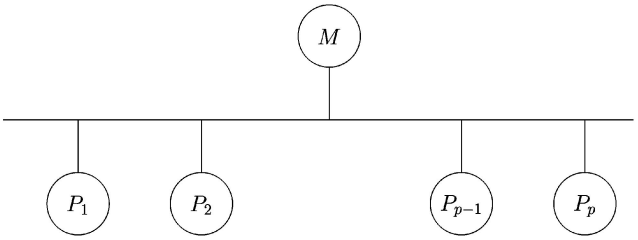


Fig. 1. The target master-slave architecture.

smallest value of T for which the answer is at least C. More precisely, to solve MinTime(C), we first determine an upper bound T, for instance, by assigning all C tasks to the fastest processor, and by rounding up the execution time obtained in this way to the closest power of 2. Next, we call MaxTasks($T/2$): If the answer is greater than or equal to C, we call MaxTasks with the value $T/4$; otherwise, we call MaxTasks with the value $3T/4$, and we proceed recursively.

We now state some specific hypotheses for the communication costs. For each modeling of these communication costs, we analytically formulate the MaxTasks(T) problem.

2.1 Without Any Communication Cost

Assume first that there is no communication cost at all. The solution of problem MaxTasks(T) is straightforward. Because each slave processor can work independently during T time-steps, the only constraint for P_i writes $c_i t_i \leq T$, or $c_i \leq \frac{T}{t_i}$. Because the c_i must be integers, letting $c_i = \left\lfloor \frac{T}{t_i} \right\rfloor$ for all i, $1 \leq i \leq p$, clearly defines a feasible and optimal solution.

In this simple case, note that problem MinTime(C) can also be solved directly, using the greedy algorithm described in [3] to distribute independent tasks to heterogeneous processors.

2.2 With an Initial Scattering of Data

The formulation of this problem is taken from Andonie et al. [1], who study the implementation of distributed back-propagation neural networks on heterogeneous networks of workstations, using the PVM library [11]. The training of the neural network is divided into computational phases. At each phase, the training pattern is distributed among the slaves, which are different-speed processors. Before executing any task, each slave must receive some data file from the master processor. Because the communication medium is exclusive, it is not relevant to distinguish whether the data file is the same for all slaves (then, the master executes a broadcast operation), or whether it is different (then, the master executes a scatter operation): We only assume that each slave must receive the same amount of data, and that each reception costs t_{com} units of time. In the model of Andonie et al. [1], there is no communication cost paid to send the results back to the master. In general, when the slaves compute "yes/no" results, the cost of returning the results may well be neglected in front of the cost of the initial scatter and/or of the computations. Note that we deal with another model, including communication costs both before and after the tasks, in Section 2.3.

Due to the constraint on the communication medium, the p messages will be sent one after the other. Obviously, it cannot hurt to send the messages as soon as possible, i.e., at time steps 0, t_{com}, $2t_{\text{com}}$, $\ldots$, $(p-1)t_{\text{com}}$. The problem is then to determine the ordering of the p messages, i.e., the permutation σ of $\{1, 2, \ldots, p\}$ such that slave P_i receives the message at time $\sigma(i)t_{\text{com}}$. We are ready to state the optimization problem analytically:

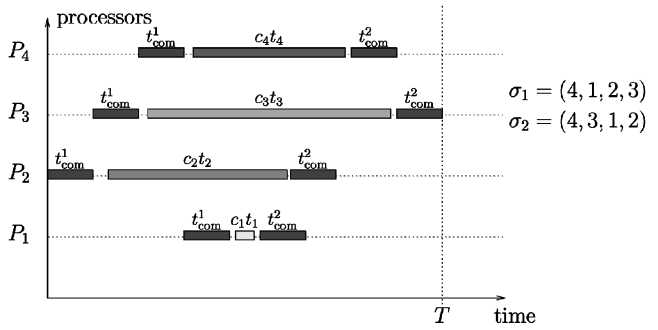


Fig. 2. Delaying messages sent back to the host.

MaxTasks1(T). Given a time bound T, determine the best allocation order of tasks to slaves, i.e., a permutation σ and an allocation $\mathcal{C} = \{c_1, c_2, \ldots, c_p\}$ s.t. all processors complete their execution within T units of time and the total number of tasks is maximized:

$$\max \left(\sum_{i=1}^{p} c_i \mid \sigma \text{ permutation and } \forall i \in [1,p] : \sigma(i)t_{\text{com}} + c_i t_i \leq T \right).$$

To solve MinTime(C) in this context, we would compute the execution time on the fastest processor as $t_{\text{com}} + C \min_{1 \leq i \leq p} t_i$, and round this value up to the next power of 2: This leads to the value of T used in the dichotomic search.

2.3 With Initial and Final Communications

As pointed out above, it is natural to assume that, after the processing of their tasks, slave processors will send some data back to the master. Because this message may well have a different size than the message initially sent by the master, we model this situation by using two communication costs, t^1_{com} for the messages sent by the master to the slaves, and t^2_{com} for the messages sent by the slaves to the master.

As shown above, we look for a permutation σ_1 which determines the ordering of the initial messages from the host: The host sends data to slave P_i at time $\sigma_1(i)t^1_{\text{com}}$. But, we also look for a second permutation σ_2 which determines the ordering of the final messages sent back to the host: Given a time bound T, slave P_i sends data back to the host at time $T - \sigma_2(i)t^2_{\text{com}}$. Here, σ_2 corresponds to the reverse order of the communications back to the host (see Fig. 2). This formulation is without any loss of generality: Some slave processor P_i might send its message earlier than this bound, but we can always shift the communication pattern as stated, i.e., delay some messages, as illustrated in Fig. 2. We are ready to state the optimization problem analytically:

MaxTasks2(T). Given a time bound T, determine the best allocation of tasks to slaves, i.e., two permutations σ_1 and σ_2, and an allocation $\mathcal{C} = \{c_1, c_2, \ldots, c_p\}$ s.t. all processors complete their execution within T units of time and the total number of tasks is maximized:

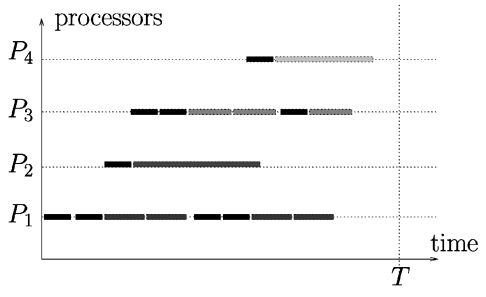


Fig. 3. Grouping some messages sent by the host to a given processor.

$$\max\left(\sum_{i=1}^{p} c_i \mid \sigma_1, \sigma_2 \text{ permutations and } \forall i \in [1,p] : \sigma_1(i)t^1_{\text{com}} + c_i t_i + \sigma_2(i)t^2_{\text{com}} \leq T\right).$$

2.4 With Communications Before Each Task Processing

We also consider the case when communications are required between the master and the slave before the processing of each task. In this third model, we consider that the cost of this communication is t_{com}. This model is quite natural: Some specific input data may well have to be propagated from the master to the slave before computation can start.

We look for three functions $f_{\text{startcomm}}$, $f_{\text{startcomp}}$, and f_{proc}: $f_{\text{startcomm}}(i)$ represents the time-step at which the communication required by task i will begin; $f_{\text{startcomp}}(i)$ represents the time-step at which the processing of task i will begin on processor $f_{\text{proc}}(i)$. The functions $f_{\text{startcomm}}$, $f_{\text{startcomp}}$, and f_{proc} must fulfill the following conditions:

- $\forall i \geq 1,\ \ f_{\text{startcomm}}(i+1) - f_{\text{startcomm}}(i) \geq t_{\text{com}}$, which states that communications take place in exclusive mode.
- $\forall i \geq i,\ \ f_{\text{startcomp}}(i) \geq f_{\text{startcomm}}(i) + t_{\text{com}}$, which states that the processing of task i cannot start before the end of the communication required by task i.
- $\forall 1 \leq i < j$, if $f_{\text{proc}}(i) = f_{\text{proc}}(j) = k$, then $f_{\text{startcomp}}(j) \geq f_{\text{startcomp}}(i) + t_k$, which states that tasks are processed sequentially on each processor k.
- $\forall 1 \leq i < j,\ \ \text{if } f_{\text{proc}}(i) = f_{\text{proc}}(j) = k,$, then

$$[f_{\text{startcomm}}(j), f_{\text{startcomm}}(j) + t_{\text{com}}] \cap [f_{\text{startcomp}}(i), f_{\text{startcomp}}(i) + t_k] = \emptyset,$$

  which states that communications and computations cannot be overlapped on processor k.

This formulation is quite general. Note that each processor can perform several communications before processing the corresponding tasks, as illustrated in Fig. 3. We are ready to state the optimization problem analytically:

MaxTasks3(T). Given a time bound T, determine the best allocation of tasks to slaves, i.e., three functions $f_{\text{startcomm}}$, $f_{\text{startcomp}}$, and f_{proc}, satisfying all the conditions stated above, s.t. all processors complete their execution within T units of time and the total number of tasks is maximized:

$$\max\big(N \mid \forall i \leq N,\ f_{\text{startcomp}}(i) + t_{f_{\text{proc}}(i)} \leq T\big).$$

2.5 With Communications Both Before and After Each Task Processing

It is natural to assume that, after the processing of each task, slave processors will send some data back to the master. As seen previously, we model this situation by using two different communication costs, t^1_{com} for the messages sent by the master to the slaves, and t^2_{com} for the messages sent by the slaves to the master.

We look for four functions: $f^1_{\text{startcomm}}$, $f^2_{\text{startcomm}}$, $f_{\text{startcomp}}$, and f_{proc}: $f^1_{\text{startcomm}}(i)$ represents the time-step at which the communication from the host required before task i will begin (just as $f_{\text{startcomm}}(i)$ in the previous section); similarly, $f^2_{\text{startcomm}}(i)$ represents the time-step at which the communication back to the host after task i will begin. Finally, $f_{\text{startcomp}}$ and $f_{\text{proc}}(i)$ are defined as before: $f_{\text{startcomp}}(i)$ represents the time-step at which the processing of task i will begin on processor $f_{\text{proc}}(i)$. The functions $f^1_{\text{startcomm}}$, $f^2_{\text{startcomm}}$, $f_{\text{startcomp}}$, and f_{proc} have to fulfill the following conditions:

- $\forall i \geq 1,\ \forall j \geq 1,\ \ \forall (k,l) \in \{1,2\},\ \text{if } k \neq l \text{ or } i \neq j$ then

$$[f^k_{\text{startcomm}}(i), f^k_{\text{startcomm}}(i) + t^k_{\text{com}}] \cap [f^l_{\text{startcomm}}(j), f^l_{\text{startcomm}}(j) + t^j_{\text{com}}] = \emptyset,$$

  which states that communications take place in exclusive mode.
- $\forall i \geq 1,\ \ f_{\text{startcomp}}(i) \geq f_{\text{startcomm}}(i) + t^1_{\text{com}}$, which states that the processing of task i cannot start before the end of the communication from the host required by task i.
- $\forall i \geq 1, f_{\text{startcomp}}(i) + t_{f_{\text{proc}}(i)} \leq f^2_{\text{startcomm}}(i)$, which states that the communication back to the host required after task i cannot start before the end of the processing of task i.
- $\forall 1 \leq i < j,\ \text{if } f_{\text{proc}}(i) = f_{\text{proc}}(j) = k,\ f_{\text{startcomp}}(j) \geq f_{\text{startcomp}}(i) + t_k$, which states that tasks are processed sequentially on each processor k.
- $\forall 1 \leq i < j$, if $f_{\text{proc}}(i) = f_{\text{proc}}(j) = k$, then

$$[f^1_{\text{startcomm}}(j), f^1_{\text{startcomm}}(j) + t_{\text{com}}] \cap [f_{\text{startcomp}}(i), f_{\text{startcomp}}(i) + t_k] = \emptyset$$

  and

$$[f^2_{\text{startcomm}}(i), f^2_{\text{startcomm}}(i) + t_{\text{com}}] \cap [f_{\text{startcomp}}(j), f_{\text{startcomp}}(j) + t_k] = \emptyset,$$

  which states that communications and computations cannot be overlapped on processor k.

Again, this formulation is quite general. Note that each processor can perform several communications from the host before processing the corresponding tasks, as well as delaying several communications back to the host: this is

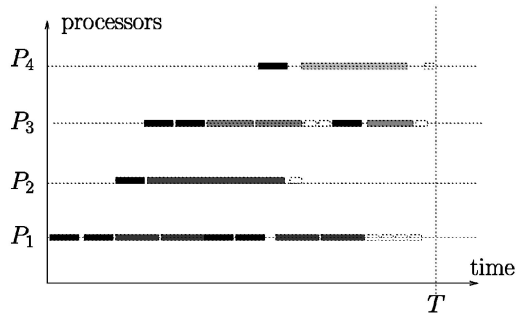


Fig. 4. Grouping some messages sent by and to the host.

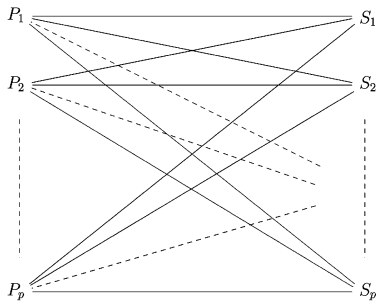


Fig. 5. Bipartite graph for MaxTasks1(T).

illustrated in Fig. 4. We are ready to state the optimization problem analytically:

MaxTasks4(T). Given a time bound T, determine the best allocation of tasks to slaves, i.e., four functions $f^1_{\text{startcomm}}$, $f^2_{\text{startcomm}}$, $f_{\text{startcomp}}$, and f_{proc} satisfying all the conditions stated above, s.t. all processors complete their execution within T units of time and the total number of tasks is maximized:

$$\max\bigl(N \mid \forall i \leq N,\ f^2_{\text{startcomm}}(i) + t^2_{\text{com}} \leq T\bigr).$$

3 Solution with an Initial Scattering of Data

3.1 Restricted Search

To (partially) solve the MaxTasks1(T) problem of Section 2.2, Andonie et al. [1] restrict the search to allocations where the fastest processors start computing first. They use a dynamic programming algorithm to solve the optimization problem MinTime(C). With our setting for problem MaxTasks1(T), this amounts to sorting the cycle-times as $t_1 \leq t_2 \leq \ldots \leq t_p$, and to letting $\sigma(i) = i$ for $1 \leq i \leq p$. The intuition is that since fastest processors execute tasks more rapidly than the others, they should work longer.

However, the intuition is misleading in some cases. Assume, for instance, two slave processors ($p = 2$) with $t_1 = 5$ and $t_2 = 9$, and let $t_{\text{com}} = 1$. For the time bound $T = 28$, it is better to start the slow processor P_2 first: P_2 can then execute three tasks: $t_{\text{com}} + 3t_2 = 28 \leq T$; the fast processor, although started at time-step $2t_{\text{com}} = 2$, can execute five tasks: $2t_{\text{com}} + 5t_1 = 27 \leq T$. If we start the fast processor first, it cannot execute more than five tasks, while the second processor can execute only two.

3.2 Matching Techniques

The optimal solution to the MaxTasks1(T) problem can be found using a weighted-matching algorithm. The idea is to draw a complete bipartite graph with $2p$ vertices, as shown in Fig. 5. Vertices on the left represent processors, while vertices on the right represent possible values for the permutation σ. The edge from vertex P_i to vertex S_j is weighted with the maximum number of tasks that P_i can execute if $\sigma(i) = j$, namely, $\left\lfloor \frac{T - jt_{\text{com}}}{t_i} \right\rfloor$. It is easy to see that there is a one-to-one correspondence between matchings in this complete bipartite graph and permutations. Because the total weight of a given matching is the total number of tasks that can be executed for the corresponding choice of the permutation, our problem reduces to finding the maximum weighted matching in the bipartite graph. Efficient (polynomial) algorithms exist to solve this problem, see [12], [22]. More precisely, the complexity of finding a maximum weighted matching in a bipartite graph with $2p$ vertices is of order $O(p^{\frac{5}{2}})$ [15]. We formally state our result:

Proposition 1. *An optimal solution to the MaxTasks1(T) problem with p processors can be found in $O(p^{\frac{5}{2}})$ time using a maximum weighted matching algorithm.*

Note that the overhead induced by the search for an optimal task allocation with the matching algorithm does depend on p, the number of processors, but not on $\sum_{i=1}^{p} c_i$, the total number of tasks to be processed. Therefore, this overhead becomes negligible when the number of tasks becomes large.

To conclude this section, we work out the example given in Section 3.1: With two processors, $t_1 = 5$, $t_2 = 9$, $t_{\text{com}} = 1$, and $T = 28$, we obtain the weighted graph of Fig. 6. For instance, the weight of the edge (P_1, S_2) is $\left\lfloor \frac{28 - 2t_{\text{com}}}{t_1} \right\rfloor = 5$. The maximum weighted matching is composed of the two edges, (P_1, S_2) and (P_2, S_1), and has total weight 8, which is indeed the maximum number of tasks that can be executed within 28 time-steps.

4 Solution with Initial and Final Communications

The solution to the MaxTasks2(T) problem with initial and final messages turns out to be surprisingly difficult. In fact, we prove that this problem is strongly NP-Complete in Section 4.1. Nevertheless, we present an efficient guaranteed approximation using matching techniques, as explained in Section 4.2.

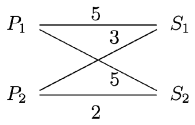


Fig. 6. Working out the example of Section 3.1.

4.1 Strong NP-Completeness of MaxTasks2(*T*)

In this section, we prove the strong NP-Completeness of MaxTasks2(T). The decision problem associated to MaxTasks2(T) is the following:

MaxTasks2-Dec(T, K). Given a time bound T and a bound K, determine an allocation order of tasks to slave, i.e., two permutations σ_1 and σ_2, and an allocation $\mathcal{C} = \{c_1, c_2, \ldots, c_p\}$ s.t. all processors complete their execution within T units of time and the total number of tasks is at least K:

$$\text{Find } \sigma_1, \sigma_2 \text{ permutations}, c_1, c_2, \ldots, c_p, \sum c_i = K,$$
$$\forall i \in [1,p] : \sigma_1(i)t^1_{\text{com}} + c_i t_i + \sigma_2(i)t^2_{\text{com}} \leq T.$$

In order to prove the strong NP-Completeness of MaxTasks2-Dec(T, K), we use a reduction to RN-3DM, a special instance of Numerical 3-Dimensional Matching that has been proved to be strongly NP-Complete by Yu [23]. A general instance of RN-3DM is as follows:

RN-3DM: Let $U = (u_1, u_2, \ldots, u_p)$ and e such that $\sum_{i=1}^{p} u_i = pe + p(p+1)$. Are there two permutations λ_1 and λ_2, such that

$$\forall 1 \leq i \leq p, \quad \lambda_1(i) + \lambda_2(i) + u_i = e?$$

We can now prove the following theorem:

Theorem 1. *MaxTasks2-Dec(T, K) is strongly NP-Complete.*

Proof. We consider the following instance of MaxTasks2-Dec(T, K). Let $t^1_{\text{com}} = t^2_{\text{com}} = 1$, $m = \max_{1 \leq i \leq p} u_i$, $t_i = u_i + e + m$ for $1 \leq i \leq p$, $T = 2e + m$, and $K = p$. Clearly, the size of this instance of MaxTasks2-Dec(T, K) is polynomial (and even linear) in the size of the original instance of RN-3DM. We prove that the above instance of MaxTasks2-Dec(T, K) has a solution if and only if the general instance of RN-3DM has a solution.

Let us first suppose that the general instance of RN-3DM has a solution. Then, there exist two permutations, λ_1 and λ_2, such that

$$\forall 1 \leq i \leq p, \quad \lambda_1(i) + \lambda_2(i) + u_i = e.$$

We let $\sigma_1 = \lambda_1$ and $\sigma_2 = \lambda_2$. Then,

$$\begin{aligned} \forall 1 \leq i \leq p, \quad \lambda_1(i) + \lambda_2(i) + u_i = e \quad &\Longrightarrow \quad \sigma_1(i) + u_i + e + \\ &\quad m + \sigma_2(i) = 2e + m \\ &\Longrightarrow \quad \sigma_1(i) + t_i + \sigma_2(i) \\ &\quad = T. \end{aligned}$$

Therefore, each slave can perform exactly one task in time T, and we have built a solution to our instance of MaxTasks2-Dec(T, K). Reciprocally, suppose that we have built a solution to our instance of MaxTasks2-Dec(T, K), i.e., two permutations σ_1 and σ_2, and an allocation $\mathcal{C} = \{c_1, c_2, \ldots, c_p\}$ s.t. $\sum_{i=1}^{p} c_i = p$ and

$$\forall i \in [1,p] : \sigma_1(i) + c_i t_i + \sigma_2(i) \leq T.$$

Then,

$$\forall 1 \leq i \leq p, \quad \sigma_1(i) + \sigma_2(i) + c_i(u_i + e + m) \leq 2e + \max u_i.$$

First, let us remark that necessarily, $\forall i,\ c_i < 2$. Indeed, if $\exists i,\ c_i \geq 2$, then

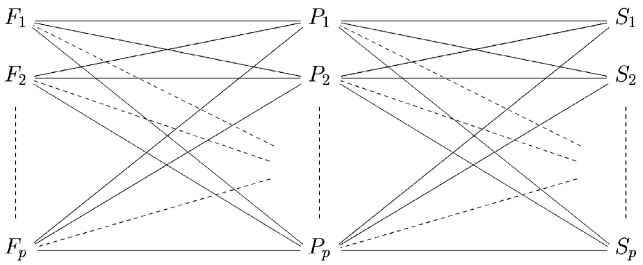


Fig. 7. Bipartite graph with initial and final communications.

$$\begin{aligned} c_i(u_i + e + \max u_i) \quad &\geq \quad 2u_i + 2e + 2\max u_i \\ &> \quad 2e + max\ u_i = T, \end{aligned}$$

which is clearly impossible since the tasks assigned to processor i would not be processed within T units of time. Thus, since $\sum_{i=1}^{p} c_i = p$ and $\forall i,\ c_i < 2$, then $\forall i,\ c_i = 1$. Therefore, if we set $\lambda_1 = \sigma_1$ and $\lambda_2 = \sigma_2$, then

$$\forall 1 \leq i \leq p, \ \ \sigma_1(i) + u_i + e + m + \sigma_2(i) \leq 2e + m \Longrightarrow$$
$$\lambda_1(i) + \lambda_2(i) + u_i \leq e.$$

By summing all the inequalities above, we obtain $p(p+1) + \sum_{i=1}^{p} u_i \leq pe$ and, since $p(p+1) + \sum_{i=1}^{p} u_i = pe$, all the inequalities above are in fact tight. Hence, we have built a solution to the general instance of RN-3DM.

This achieves the proof of the strong NP-Completeness of MaxTasks2-Dec(T, K). □

4.2 Matching Techniques

To take both permutations σ_1 and σ_2 into account, we build a bipartite graph $G = (V, E)$ with $3p$ vertices (i.e., $|V| = 3p$), as shown Fig. 7. The p leftmost vertices F_i correspond to the first permutation σ_1, the p center vertices P_i correspond to processors, and the p rightmost vertices S_i correspond to the second permutation σ_2. Rather than a matching, we extract from the graph a subset E' of $2p$ edges so that, in the graph $G = (V, E')$, each vertex F_i or S_i is exactly of degree 1, and each vertex P_i is exactly of degree 2. The complexity of extracting such a spanning subgraph from the graph with $3p$ vertices is of order $O(p^{\frac{5}{2}})$ again since we can solve independently the maximum weighted matching in both bipartite graphs with $2p$ vertices (on the left-hand size and on the right-hand size in Fig. 7) in time of order $O(p^{\frac{5}{2}})$.

The problem is that edge weights cannot be determined fully accurately; the inequality $\sigma_1(i)t^1_{\text{com}} + c_i t_i + \sigma_2(i)t^2_{\text{com}} \leq T$ translates into

$$c_i \leq \left\lfloor \frac{T - \sigma_1(i)t^1_{\text{com}} - \sigma_2(i)t^2_{\text{com}}}{t_i} \right\rfloor,$$

and we need to know *both* $\sigma_1(i)$ and $\sigma_2(i)$ to compute c_i. Instead, we use the approximation

$$\left\lfloor \frac{T/2 - \sigma_1(i)t^1_{\text{com}}}{t_i} \right\rfloor + \left\lfloor \frac{T/2 - \sigma_2(i)t^2_{\text{com}}}{t_i} \right\rfloor.$$

This approximation enables us to weight the edges as follows: The edge between F_j and P_i is weighted as

$$\left\lfloor \frac{T/2 - jt^1_{\text{com}}}{t_i} \right\rfloor,$$

while the edge between P_i and S_k is weighted as

$$\left\lfloor \frac{T/2 - kt^2_{\text{com}}}{t_i} \right\rfloor.$$

Theorem 2. *The previous approximation leads to tasks allocations that differs at most by p from the optimal solution.*

Proof. First, note that

$$\forall a, b : 0 \leq \lfloor a+b \rfloor - \lfloor a \rfloor - \lfloor b \rfloor \leq 1. \quad (1)$$

Therefore, for any allocation (σ_1, σ_2, c) built using the previous approximation, we have

$$\begin{aligned} \forall i,\ c_i &\leq \left\lfloor \frac{T/2 - \sigma_1(i)t^1_{\text{com}}}{t_i} \right\rfloor + \left\lfloor \frac{T/2 - \sigma_2(i)t^2_{\text{com}}}{t_i} \right\rfloor \\ &\leq \left\lfloor \frac{T - \sigma_1(i)t^1_{\text{com}} - \sigma_2(i)t^2_{\text{com}}}{t_i} \right\rfloor, \end{aligned}$$

and (σ_1, σ_2, c) is then a solution of the initial problem.

Let us note by

$$C_{app}(\sigma_1, \sigma_2) = \sum_{i=1}^{p} \left\lfloor \frac{T/2 - \sigma_1(i)t^1_{\text{com}}}{t_i} \right\rfloor + \left\lfloor \frac{T/2 - \sigma_2(i)t^2_{\text{com}}}{t_i} \right\rfloor,$$

the approximative cost of two permutations and

$$\sigma_{app} = (\sigma_1^{(app)}, \sigma_2^{(app)}) = \underset{\sigma_1, \sigma_2}{\operatorname{argmax}}\, C_{app}(\sigma_1, \sigma_2),$$

the permutations built by our algorithm.

Let

$$C_{opt}(\sigma_1, \sigma_2) = \sum_{i=1}^{p} \left\lfloor \frac{T - \sigma_1(i)t^1_{\text{com}} - \sigma_2(i)t^2_{\text{com}}}{t_i} \right\rfloor$$

denote the real cost of two permutations and let

$$\sigma_{opt}(\sigma_1^{(opt)}, \sigma_2^{(opt)}) = \underset{\sigma_1, \sigma_2}{\operatorname{argmax}}\, C_{opt}(\sigma_1, \sigma_2)$$

be some permutations that are optimal solutions to the original problem.

By definition,

$$C_{opt}(\sigma_{app}) \leq C_{opt}(\sigma_{opt}) \quad (2)$$

and

$$C_{app}(\sigma_{opt}) \leq C_{app}(\sigma_{app}). \quad (3)$$

Using (1), we find that

$$\forall \sigma : C_{opt}(\sigma) - p \leq C_{app}(\sigma) \leq C_{opt}(\sigma) \quad (4)$$

and, therefore,

$$\begin{aligned} C_{opt}(\sigma_{opt}) - p &\leq C_{app}(\sigma_{opt}) && \text{(using (4))} \\ &\leq C_{app}(\sigma_{app}) && \text{(using (3))} \\ &\leq C_{opt}(\sigma_{app}) && \text{(using (4))} \\ C_{opt}(\sigma_{opt}) - p &\leq C_{opt}(\sigma_{app}) \leq C_{opt}(\sigma_{opt}) && \text{(using (2)),} \end{aligned}$$

which means that our approximation leads to tasks allocations that differs at most by p from the optimal solution. □

4.3 Simulations

In this section, we compare solutions given by the approximation algorithm proposed in Section 4, with the optimal ones. For a given instance of the problem, we compute the difference between number of tasks processed using the optimal permutations (found with an exhaustive search), and the number of tasks processed using the approximation algorithm proposed in Section 4. Fig. 8 depicts the mean value of this error for several tests when the number of processors varies:

1. In Fig. 8a, cycle times are moderately heterogeneous (numbers at random between 15 and 25) and communication times can be large compared to cycle times (numbers at random between 10 and 90).
2. In Fig. 8b, cycle times are moderately heterogeneous (numbers at random between 15 and 25), but communication times are rather small compared to cycle times (between 1 and 9).
3. In Fig. 8c, cycle times are strongly heterogeneous (numbers at random between between 5 and 35) and communication times can be large compared to cycle times (two numbers at random between 10 and 90).
4. In Fig. 8d, cycle times are strongly heterogeneous (numbers at random between between 5 and 35) and communication times are rather small compared to cycle times (two numbers at random between 1 and 9).

In all cases, the matching gives very satisfying results since the difference between the optimal number of tasks that may be processed and the number of tasks processed using the matching algorithm is quite small. As we could expect, this difference grows with p, but does not strongly depend upon the degree of heterogeneity.

5 Solution with Communications Before Each Task

In this section, we present an asymptotically optimal algorithm for MaxTasks3(T): When T becomes large, the ratio of the number of tasks processed by this algorithm over the number of tasks executed by the optimal solution tends to one.

5.1 Theoretical Bounds

In order to prove the asymptotic optimality of our algorithm, we need to determine the optimal number of tasks that can be performed if the cost of a communication between the master and the slave is t_{com} and the cycle times of slaves processors are $t_1 \leq t_2 \leq \ldots \leq t_p$. Consider a valid communication and computation scheme, i.e., three functions $f_{\text{startcomm}}$, $f_{\text{startcomp}}$, and f_{proc}, satisfying the conditions given in Section 2.4. Let T be the time bound and let N denote the maximal number of tasks that can be processed within T units of time:

$$N = \max\{n,\ \forall i \leq n,\ f_{\text{startcomp}}(i) + T_{f_{\text{proc}}(i)} \leq T\}.$$

Moreover, let

$$\forall i,\ 1 \leq i \leq p,\ \ \alpha_i = \frac{\text{Card}\{j,\ \ f_{\text{proc}}(j) = i\}\ \ t_{\text{com}}}{T}$$

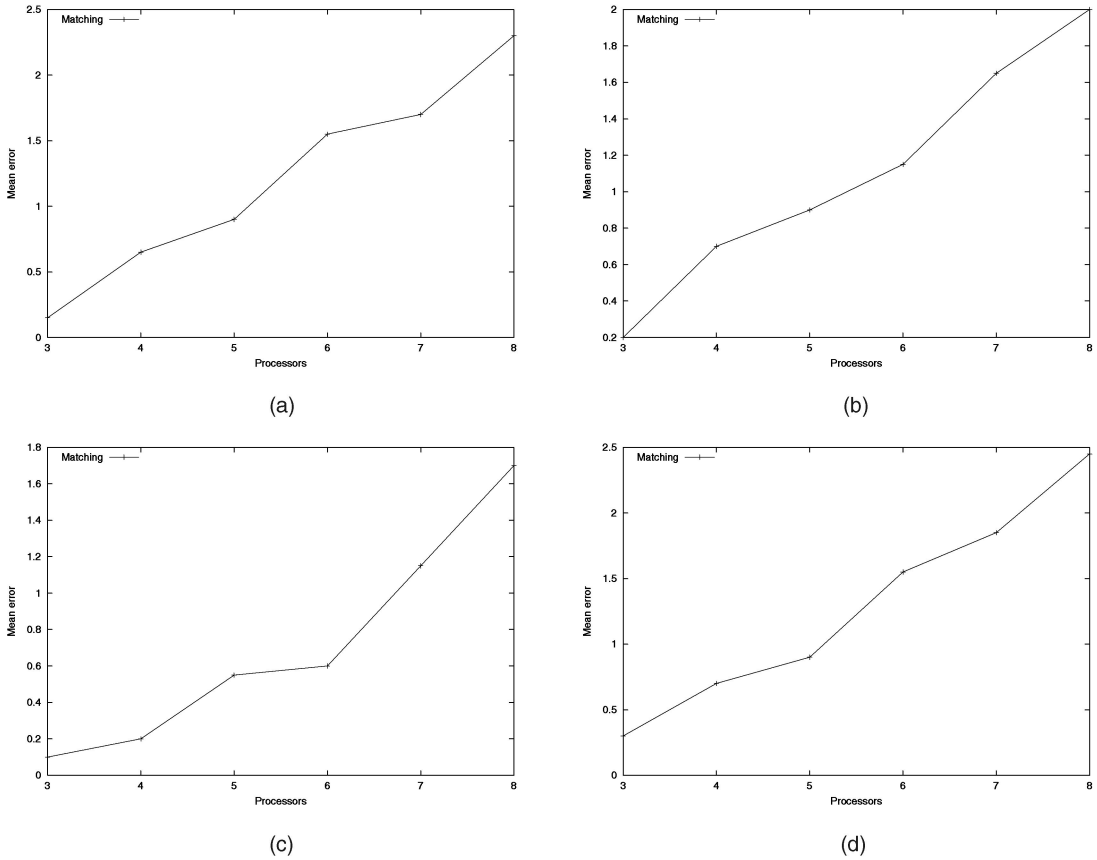


Fig. 8. A few simulations. (a) comp 20 ± 5; comm 50 ± 40, (b) comp 20 ± 5; comm 5 ± 4, (c) comp 20 ± 15; comm 50 ± 40, and (d) comp 20 ± 15; comm 5 ± 4.

be the ratio of the time spent by the master to perform communications with slave P_i over the time bound T.

Lemma 1. *With the above notations, we have*

$$N \leq \sum_i \frac{T}{t_{\text{com}} + t_i} \tag{5}$$

$$N \leq \frac{T}{t_{\text{com}}} \tag{6}$$

Proof.

1. Since $\forall 1 \leq i \leq p,\ \alpha_i = \frac{\text{Card}\{j,\ f_{\text{proc}}(j)=i\}\ t_{\text{com}}}{T}$, then $\frac{T\alpha_i}{t_{\text{com}}} = \text{Card}\{j,\ f_{\text{proc}}(j) = i\}$ and
$$\frac{T\sum_{i=1}^{p} \alpha_i}{t_{\text{com}}} = \sum_{i=1}^{p} \text{Card}\{j,\ f_{\text{proc}}(j) = i\} = N. \tag{7}$$
2. Let us determine the maximal number of tasks that can be performed by slave P_i, $\forall 1 \leq i \leq p$. Since the overall cost of a task on slave P_i is $t_{\text{com}} + t_i$, we have
$$\text{Card}\{j,\ f_{\text{proc}}(j) = i\} \leq \frac{T}{t_{\text{com}} + t_i},\ \text{i.e.,}\ \alpha_i \leq \frac{t_{\text{com}}}{t_{\text{com}} + t_i}. \tag{8}$$
3. Let us determine the number of communications that can be performed by the master. The cost of a communication is t_{com}, so that $Nt_{\text{com}} \leq T$ and, then, using (7), we deduce
$$\sum_{i=1}^{p} \alpha_i \leq 1. \tag{9}$$
4. Using (7) and (8), we have
$$N \leq \sum_i \frac{T}{t_{\text{com}} + t_i}.$$
5. Using (7) and (9), we deduce
$$N \leq \frac{T}{t_{\text{com}}}.$$

□

In order to determine the optimal number of tasks that can be performed during T time steps, we need to distinguish two different cases, according to the value of $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i}$. Indeed, it turns out that the communication network is not the limiting resource if $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} \leq 1$, but it becomes the limiting resource otherwise.

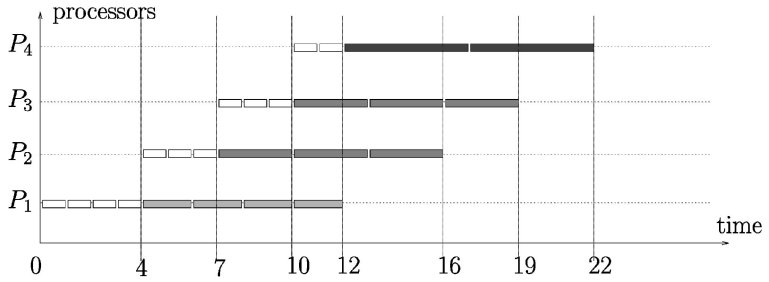


Fig. 9. Example when $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} \leq 1$.

5.2 Solution of MaxTasks3(*T*) if $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} \leq 1$

To solve MaxTasks3(T), we propose an algorithm that consists in determining a pattern for communications and computations, that will be reproduced periodically throughout the execution.

Let $tc_i = t_{\text{com}} + t_i$, for $1 \leq i \leq p$, denote the overall cost of the processing of a task on slave P_i since we cannot overlap communications and computations. Let T^{pattern} be the least common multiple of these p values tc_i: T^{pattern} determines the length of the pattern. Let $nb_i^{\text{pattern}} = \frac{T^{\text{pattern}}}{tc_i}$ be the number of tasks processed by processor P_i during the execution of the pattern.

Consider the following example: $t_{\text{com}} = 1$ and $p = 4$ slave processors with $t_1 = 2$, $t_2 = 3$, $t_3 = 3$, and $t_4 = 5$. We have $tc_1 = 3$, $tc_2 = 4$, $tc_3 = 4$, and $tc_4 = 6$. In this case, $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} = 1$, $T^{\text{pattern}} = 12$, and $nb_1^{\text{pattern}} = 4$, $nb_2^{\text{pattern}} = 3$, $nb_3^{\text{pattern}} = 3$, and $nb_4^{\text{pattern}} = 2$.

To formally build the pattern, we need some complicated notations. We advise the reader to follow the construction for the example, using Fig. 9. First, we define time-steps and processors within the pattern, using three new functions $f_{\text{startcomm}}^{\text{pattern}}$, $f_{\text{startcomp}}^{\text{pattern}}$, and $f_{\text{proc}}^{\text{pattern}}$, which we define as follows (initially, $nb_0^{\text{pattern}} = 0$):

- Determine which processor executes task number i:

$$\forall 1 \leq i \leq \sum_{k=1}^{p} nb_k^{\text{pattern}} : f_{\text{proc}}^{\text{pattern}}(i) = \min\left\{ j \mid i > \sum_{k=0}^{j-1} nb_k^{\text{pattern}} \right\}.$$

- Determine the beginning of the communication and of the computation for task number i:

$$f_{\text{startcomm}}^{\text{pattern}}(i) = 1 + \sum_{k=0}^{j-1} nb_k^{\text{pattern}}(t_{\text{com}} + t_k) + (i - 1 - \sum_{k=0}^{j-1} nb_k^{\text{pattern}}) t_{\text{com}},$$

$$f_{\text{startcomp}}^{\text{pattern}}(i) = 1 + \sum_{k=0}^{j} nb_k^{\text{pattern}} t_{\text{com}} + \sum_{k=0}^{j-1} nb_k^{\text{pattern}} t_k + (i - 1 - \sum_{k=0}^{j-1} nb_k^{\text{pattern}}) t_j.$$

The important fact about this pattern is that $\forall 2 \leq i \leq p$,

$$\left(f_{\text{startcomp}}^{\text{pattern}}\left(\sum_{k=0}^{i} nb_k^{\text{pattern}} \right) + t_i \right) - \left(f_{\text{startcomp}}^{\text{pattern}}\left(\sum_{k=0}^{i-1} nb_k^{\text{pattern}} \right) + t_{i-1} \right) = \left(f_{\text{startcomm}}^{\text{pattern}}\left(\sum_{k=0}^{i-1} nb_k^{\text{pattern}} + 1 \right) \right) - \left(f_{\text{startcomm}}^{\text{pattern}}\left(\sum_{k=0}^{i-2} nb_k^{\text{pattern}} + 1 \right) \right).$$

This condition states that the difference between the date of the end of the processing of the last task on slave P_i and the date of the end of the processing of the last task on slave P_{i-1} is equal to the difference between the beginning of the communication required by the first task processed by slave P_i and the beginning of the communication required by the first task processed by processor P_{i-1}. This is the key-condition that ensures the periodicity of the whole computation and communication scheme from one pattern to the next one: It is possible to execute the pattern defined above every T^{pattern} time steps, as illustrated in Fig. 9. During the execution of one pattern, we process $\sum_{i=1}^{p} nb_k^{\text{pattern}}$ tasks.

We are now able to define the functions $f_{\text{startcomm}}(n)$, $f_{\text{startcomp}}(n)$, and $f_{\text{proc}}(n)$ corresponding to our algorithm. Let k s.t. $k \sum_{i=1}^{p} nb_i^{\text{pattern}} < n \leq (k+1) \sum_{i=1}^{p} nb_i^{\text{pattern}}$ and let $i = n - k \sum_{i=1}^{p} nb_i^{\text{pattern}}$. Then,

$$f_{\text{startcomm}}(n) = kT^{\text{pattern}} + f_{\text{startcomm}}^{\text{pattern}}(i),$$
$$f_{\text{startcomp}}(n) = kT^{\text{pattern}} + f_{\text{startcomp}}^{\text{pattern}}(i),$$
$$f_{\text{proc}}(n) = f_{\text{proc}}^{\text{pattern}}(i).$$

One can easily check that the functions $f_{\text{startcomm}}$, $f_{\text{startcomp}}$, and f_{proc} satisfy all the conditions stated in Section 2.4.

5.3 Solution of MaxTasks3(*T*) if $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} > 1$

To solve MaxTasks3(T) when $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} > 1$, we slightly modify the algorithm proposed in the previous section. Indeed, in this case, the network is the limiting resource. The algorithm consists of determining a communication and computation pattern so that the communication network is always in use. Some slower processors will be kept idle at some periods, or even will never be used.

First of all, we sort the cycle-times of the slave processors and assume that

$$t_1 \leq t_2 \leq \ldots \leq t_p.$$

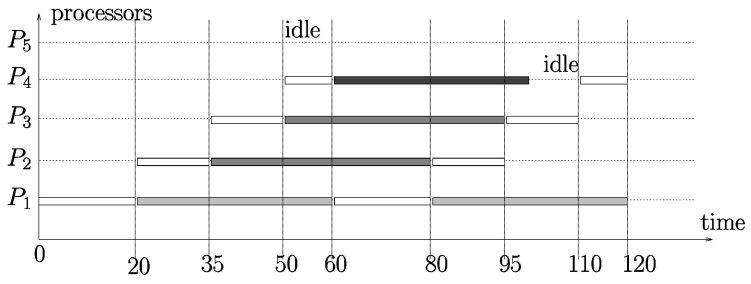


Fig. 10. Example when $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} > 1$.

Let tc_i be defined as previously and let

$$p_{max} = \max\left\{k \mid \sum_{i=1}^{k} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} \leq 1\right\}.$$

p_{max} is the index of the last processor whose computation power will be fully used in the pattern.

Let T^{pattern} be the least common multiple of t_{com} and of the tc_i, $1 \leq i \leq p_{max}$. Moreover, define nb_i^{pattern} as follows:

$$\forall\, 1 \leq i \leq p_{max}, \quad nb_i^{\text{pattern}} = \frac{T^{\text{pattern}}}{tc_i},$$

$$nb_{p_{max}+1}^{\text{pattern}} = \frac{T^{\text{pattern}} - t_{\text{com}} \sum_{i=1}^{p_{max}} nb_i^{\text{pattern}}}{t_{\text{com}}},$$

and let

$$nb_i^{\text{pattern}} = 0, \quad \forall\, i > p_{max}+1.$$

We see that processor number $p_{max}+1$ is not fully used, while other processors are not used at all.

With these notations, we define $f_{\text{startcomm}}^{\text{pattern}}$, $f_{\text{startcomp}}^{\text{pattern}}$, and $f_{\text{proc}}^{\text{pattern}}$ as in the previous section. Again, the only difference is that slaves P_i, $i > p_{max}+1$ are kept idle all the time, while slave $P_{p_{max}+1}$ is kept idle during the last $\left(T^{\text{pattern}} - nb_{p_{max}+1}^{\text{pattern}} tc_{p_{max}+1}\right)$ time steps.

The construction of the pattern is illustrated in Fig. 10, with $t_{\text{com}} = 1$, and $p = 5$ processors s.t. $t_1 = 2$, $t_2 = 3$, $t_3 = 3$, $t_4 = 4$, and $t_5 = 6$. In this case, $\sum_i \frac{t_{\text{com}}}{t_{\text{com}}+t_i} > 1$, $p_{max} = 3$, $T^{\text{pattern}} = 60$, and $nb_1^{\text{pattern}} = 20$, $nb_2^{\text{pattern}} = 15$, $nb_3^{\text{pattern}} = 15$, $nb_4^{\text{pattern}} = \frac{60-(20+15+15)*1}{1} = 10$, and $nb_5^{\text{pattern}} = 0$. Slave P_5 is not used at all, while slave P_4 stays idle the last $(60 - 10 * 5) = 10$ time-steps of the pattern.

We extend the definition of $f_{\text{startcomm}}^{\text{pattern}}$, $f_{\text{startcomp}}^{\text{pattern}}$, and $f_{\text{proc}}^{\text{pattern}}$ to $f_{\text{startcomm}}$, $f_{\text{startcomp}}$, and f_{proc} exactly as before. Again, one can easily check that the functions $f_{\text{startcomm}}$, $f_{\text{startcomp}}$, and f_{proc} satisfy all the conditions stated in Section 2.4.

5.4 Asymptotic Optimality

In this section, we show that the algorithm presented in Sections 5.2 and 5.3 is asymptotically optimal.

- Consider first the case $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} \leq 1$. Let N be the number of tasks that can be performed using our algorithm. We define k such that

$$kT^{\text{pattern}} \leq T - t_{\text{com}} \sum_{i=1}^{p-1} nb_i^{\text{pattern}} < (k+1)T^{\text{pattern}}.$$

In this expression, $t_{\text{com}} \sum_{i=1}^{p-1} nb_i^{\text{pattern}}$ represents the delay before slave P_p begins to receive its first message, as illustrated in Fig. 11, and k represents the number of patterns that can be entirely completed before time bound T. Then,

$$k \geq \frac{T}{T^{\text{pattern}}} - \frac{\sum_{i=1}^{p-1} nb_i^{\text{pattern}}}{T^{\text{pattern}}} - 1.$$

Therefore,

$$\begin{aligned} N &\geq k \sum_{i=1}^{p} nb_i^{\text{pattern}} \\ &\geq k \sum_{i=1}^{p} \frac{T^{\text{pattern}}}{t_{\text{com}}+t_i} \\ &\geq \left(\frac{T}{T^{\text{pattern}}} - \frac{\sum_{i=1}^{p-1} nb_i^{\text{pattern}}}{T^{\text{pattern}}} - 1\right) \sum_{i=1}^{p} \frac{T^{\text{pattern}}}{t_{\text{com}}+t_i} \\ &\geq \left(\sum_{i=1}^{p} \frac{T}{t_{\text{com}}+t_i}\right) - \left(\sum_{i=1}^{p-1} nb_i^{\text{pattern}} + T^{\text{pattern}}\right) \\ &\quad \left(\sum_{i=1}^{p} \frac{1}{t_{\text{com}}+t_i}\right). \end{aligned}$$

Lemma 1 ((5)) states that the optimal number of tasks N_{opt} that can be performed within T units of time satisfies

$$N_{opt} \leq \sum_{i=1}^{p} \frac{T}{t_{\text{com}}+t_i}.$$

Then, we have

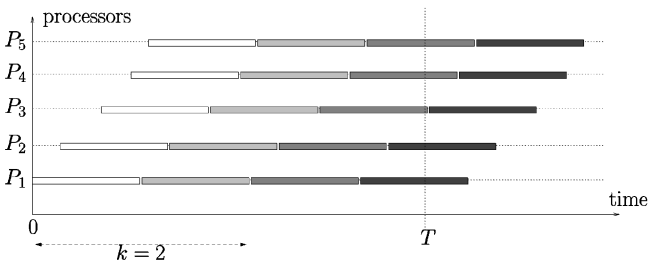


Fig. 11. Several consecutive patterns.

$$1 \geq \frac{N}{N_{opt}} \geq 1 + O\left(\frac{1}{T}\right),$$

which achieves the proof of the asymptotic optimality of our algorithm in the case $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} \leq 1$.

- Now, consider the case $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} > 1$. Let N be the number of tasks that can be performed using our algorithm. We define k such that

$$kT^{\text{pattern}} \leq T - t_{\text{com}} \sum_{i=1}^{p_{max}} nb_i^{\text{pattern}} < (k+1)T^{\text{pattern}}.$$

In this expression, $t_{\text{com}} \sum_{i=1}^{p_{max}} nb_i^{\text{pattern}}$ represents the delay before slave $P_{p_{max}+1}$ begins to receive its first message, and k represents the number of patterns that can be entirely completed before time bound T. Then,

$$k \geq \frac{T}{T^{\text{pattern}}} - \frac{t_{\text{com}} \sum_{i=1}^{p_{max}} nb_i^{\text{pattern}}}{T^{\text{pattern}}} - 1.$$

Therefore,

$$\begin{aligned} N &\geq k\left(\sum_{i=1}^{p_{max}} nb_i^{\text{pattern}} + nb_{p_{max}+1}\right) \\ &\geq k\frac{T^{\text{pattern}}}{t_{\text{com}}} \\ &\geq \left(\frac{T}{t_{\text{com}}}\right) - \left(\sum_{i=1}^{p_{max}} nb_i^{\text{pattern}} + \frac{T^{\text{pattern}}}{t_{\text{com}}}\right). \end{aligned}$$

Lemma 1 ((6)) states that the optimal number of tasks N_{opt} that can be performed within T units of time satisfies

$$N_{opt} \leq \frac{T}{t_{\text{com}}}.$$

Then, we have

$$1 \geq \frac{N}{N_{opt}} \geq 1 + O\left(\frac{1}{T}\right),$$

which achieves the proof of the asymptotic optimality of our algorithm in the case $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} > 1$.

We formally state this important result:

Theorem 3. *Let $N_{opt}(T)$ be the optimal number of tasks that can be executed within T time-steps. Let $N(T)$ be the number of tasks executed by the algorithm of Section 5.2 if $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} \leq 1$, and by the algorithm of Section 5.2 if $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} > 1$. Then,*

$$\lim_{T\to\infty} \frac{N(T)}{N_{opt}(T)} = 1.$$

5.5 Comparison with a Greedy Algorithm

In this section, we compare the results obtained with the algorithm presented in Sections 5.2 and 5.3 against the results obtained with a greedy algorithm, which works as follows: At each time step, if k slaves with respective cycle times $t_{i_1} \leq t_{i_2} \leq \ldots \leq t_{i_k}$ are waiting for a communication from the master, and if the communication network is free during the next t_{com} time steps, then a communication is performed between the master and the fastest slave P_{i_1}. In Fig. 12, we display the solution obtained with $t_{\text{com}} = 1$ and $p = 4$ slave processors with $t_1 = 2$, $t_2 = 3$, $t_3 = 3$, and $t_4 = 5$. These results are to be compared with those obtained by our algorithm (here, we have $\sum_{i=1}^{p} \frac{t_{\text{com}}}{t_{\text{com}}+t_i} = 1$), and displayed in Fig. 9.

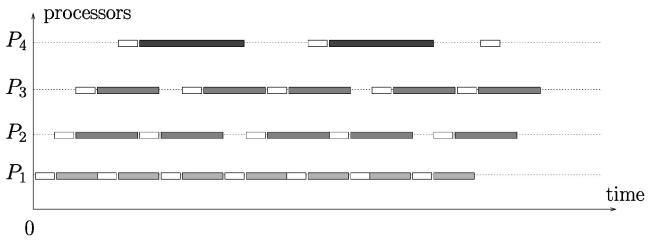


Fig. 12. Execution with the greedy algorithm.

The greedy algorithm also leads to a periodic allocation (the time period is 9); it is able to process eight tasks every nine time steps, but neither the computing resources nor the communication medium are saturated. Our algorithm is able to process 12 tasks every 12 time steps, thus leading to an improvement of order $\frac{9}{8}$.

6 Solution with Communications Both Before and After Each Task

In this section, we present an asymptotically optimal algorithm for MaxTasks4(T). The algorithm is very similar to the one presented in Section 5, so we only outline the sketch of the algorithm, and describe it through an example, without detailing the proofs.

As previously shown, we define a pattern for communications and computations that will be reproduced periodically. The pattern consists in two main phases:

- The first phase consists of both backward and forward communications between the master and the slaves.
- The second phase consists in task processing by the slaves.

Let t_{com}^1 be the communication cost for the messages from the master to the slaves, t_{com}^2 the communication cost for the messages from the slaves to the master, t_i, $1 \leq i \leq p$ the cycle times of the slaves, and T the time bound. Basically, the pattern of communications and computations is the same as those defined in Section 5, with $t_{\text{com}} = t_{\text{com}}^1 + t_{\text{com}}^2$.

The construction of the pattern is illustrated in Fig. 13, with $t_{\text{com}}^1 = 2$, $t_{\text{com}}^2 = 1$, and $p = 3$ processors: $t_1 = 8$, $t_2 = 8$, and $t_3 = 9$. In this case, $t_{\text{com}} = 3$, $\sum_i \frac{t_{\text{com}}}{t_{\text{com}}+t_i} \leq 1$, $T^{\text{pattern}} = 132$, and $nb_1^{\text{pattern}} = 12$, $nb_2^{\text{pattern}} = 12$, and $nb_3^{\text{pattern}} = 11$.

Of course, the first pattern is not executed entirely since no backward communication is required between the slaves and the master at the beginning of the execution. Similarly, the processing of tasks during the last pattern may be useless since corresponding backward messages from the slaves to the master may well not have been completed.

Nevertheless, this does not impact the asymptotic optimality of this algorithm:

Theorem 4. *Let $N_{opt}(T)$ be the optimal number of tasks that can be executed within T time-steps, and let $N(T)$ be the number of tasks executed by our algorithm. Then,*

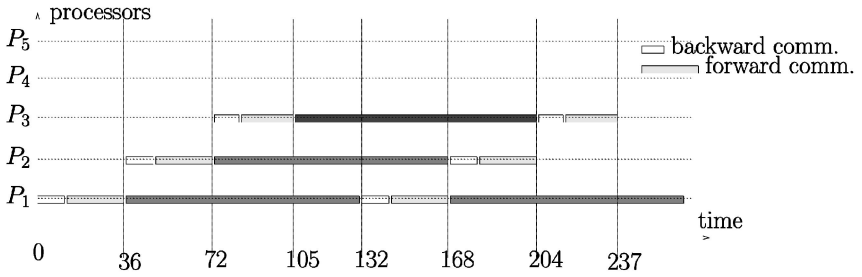


Fig. 13. Pattern when communications are required both before and after each task.

$$\lim_{T\to\infty} \frac{N(T)}{N_{opt}(T)} = 1.$$

7 Related Work

To the best of our knowledge, the most related work concerning both the application and the architecture model is presented in the paper of Andonie et al. [1], which we have already quoted in Section 3. A continuation of this work has been recently published by Chronopoulos and Jagannathan [9].

Several theoretical papers deal with complexity results for parallel machine problems with a server, establishing complexity (NP-completeness) results [13], [16], [7] and providing guaranteed approximations [17]. Before processing, each job must be loaded on a machine, which takes a certain setup time. All these setups have to be done by a single server which can handle at most one job at a time. Our first problem (with initial messages only) is a very special instance of this class of server problems.

Our second problem (with initial and final messages) is a special instance of the job-shop scheduling problem (see problem SS18 in [10]), where each job consists of only three tasks, the first and last of which having to be executed by the two machines dedicated to communications. Although this instance is very specific, we have shown that it remained NP-complete.

Generally speaking, note that our four problems differ from those cited above with a server and start-up times in that 1) all tasks are identical and independent and 2) communication times (the counterpart of the set-up times) are identical too. The difficulty lies solely in the heterogeneity of the computing resources.

Several papers by Robertazzi et al. [5], [2], [18], [8] have dealt with the distribution of a single task that can be arbitrarily partitioned in a linear fashion and then can be distributed to more than one processor to achieve better completion time. Optimal solutions have been derived when the target architecture is either a bus with heterogeneous processors (i.e., the architecture considered in this paper) or a tree. In this context, the main difference is that the cost of a communication depends on the amount of processing assigned to a processor, contrarily to the assumption made in Sections 3 and 4. Surprisingly, in this case, on a homogeneous bus, the overall completion time does not depend on the order of the communications between the master and the slaves, while it has been proven to be the main issue with our application model. The modeling of an application by the divisible load approach is also different from the one considered in Section 5. Indeed, in the context of divisible loads, one single communication is performed with each slave at the beginning, and this communication involves entirely the data needed for the entire processing. On the contrary, we allow several communications between the master and the slaves and, therefore, we can obtain a better overall completion time by enabling a fastest start, hence a better overlapping, for the whole set of processors.

In [14], Hedetniemi presents an interesting algorithm for polling in a tree network of processors that can be simplified for a bus of processors. The polling operation consists in two phases: First, the same message is sent to all the nodes by the master and, then, processors send back individual results to the master. This problem is close to the one considered in Section 4, but the polling operation does not require any processing by slave processors. Interestingly, we point out that this last operation is the clue of the complexity stated in Theorem 1; indeed, an optimal polynomial-time algorithm can be derived for polling in tree networks, whereas our problem is strongly NP-Complete.

8 Conclusion

In this paper, we have shown that deriving efficient algorithms for the master-slave paradigm, in the framework of heterogeneous computing resources and communication links used in exclusive mode, turns out to be surprisingly difficult. More precisely, we have designed an optimal polynomial algorithm in the case of an initial scattering of data and provided a guaranteed polynomial approximation algorithm in the case of initial and final communications. This last problem has been proven to be strongly NP-Complete, even on (intuitively) simple instances. Finally, we have presented asymptotically optimal algorithms for the case where each task processing must be preceded (and possibly followed) by a communication from (back to) the master.

The different variants of the master-slave problem that we have addressed in this paper seem quite representative of a large class of regular problems that exhibit a simple solution in the context of homogeneous processors, but turn out to raise several algorithmic difficulties in the context of heterogeneous resources [4], [6]. Data decomposition, scheduling heuristics, and load balancing were known to be hard problems in the context of classical parallel architectures. They become extremely difficult in the context of heterogeneous clusters, not to speak about metacomputing platforms.

REFERENCES

[1] R. Andonie, A.T. Chronopoulos, D. Grosu, and H. Galmeanu, "Distributed Backpropagation Neural Networks on a PVM Heterogeneous System," *Proc. Parallel and Distributed Computing and Systems Conf.*, pp. 555-560, 1998.

[2] S. Bataineh, T.Y. Hsiung, and T.G. Robertazzi, "Closed Form Solutions for Bus and Tree Networks of Processors Load Sharing a Divisible Job," *IEEE Trans. Computers*, vol. 43, no. 10, pp. 1184-1196, Oct. 1994.

[3] O. Beaumont, V. Boudet, A. Petitet, F. Rastello, and Y. Robert, "A Proposal for a Heterogeneous Cluster ScaLAPACK (Dense Linear Solvers)," *IEEE Trans. Computers*, vol. 50, no. 10, pp. 1052-1070, Oct. 2001.

[4] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert, "Matrix-Matrix Multiplication on Heterogeneous Platforms," *Proc. Int'l Conf. Parallel Processing*, 2000.

[5] V. Bharadwaj, D. Ghose, V. Mani, and T.G. Robertazzi, *Scheduling Divisible Loads in Parallel and Distributed Systems.* IEEE Press, 1996.

[6] P. Boulet, J. Dongarra, F. Rastello, Y. Robert, and F. Vivien, "Algorithmic Issues on Heterogeneous Computing Platforms," *Parallel Processing Letters*, vol. 9, no. 2, pp. 197-213, 1999.

[7] P. Brucker, C. Dhaenens-Flipo, S. Knust, S.A. Kravchenko, and F. Werner, "Complexity Results for Parallel Machine Problems with a Single Server," Technical Report Reihe P, No. 219, Fachbereich Mathematik Informatik, Univ. Osnabrück, 2000.

[8] S. Charcranoon, T.G. Robertazzi, and S. Luryi, "Optimizing Computing Costs Using Divisible Load Analysis," *IEEE Trans. Computers*, vol. 49, no. 9, pp. 987-991, Sept. 2000.

[9] A.T. Chronopoulos and S. Jagannathan, "A Distributed Discrete-Time Neural Network Architecture for Pattern Allocation and Control," *Proc. IPDPS Workshop Bioinspired Solutions to Parallel Processing Problems*, 2002.

[10] M.R. Garey and D.S. Johnson, *Computers and Intractability, a Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1991.

[11] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing.* MIT Press, 1996.

[12] M. Gondran and M. Minoux, *Graphs and Algorithms.* John Wiley and Sons, 1984.

[13] N. Hall, C.N. Potts, and C. Sriskandarajah, "Parallel Machine Scheduling with a Common Server," *Discrete Applied Math.*, vol. 102, pp. 223-243, 2000.

[14] S. Hedetniemi, Open Problems in Combinatorial Optimization, World Wide Web Document, http://www.cs.clemson.edu/~hedet/algorithms.html, year?

[15] J.E. Hopcroft and R.M. Karp, "An $n^{5/2}$ Algorithm for Maximum Matching in Bipartite Graphs," *SIAM J. Computing*, vol. 2, no. 4, pp. 225-231, 1973.

[16] S.A. Kravchenko and F. Werner, "Parallel Machine Scheduling Problems with a Single Server," *Math. Computational Modelling*, vol. 26, pp. 1-11, 1997.

[17] H. Lee and M. Guignard, "A Hybrid Bounding Procedure for the Workload Allocation Problem on Parallel Unrelated Machines with Setups," *J. Operational Research Soc.*, vol. 47, pp. 1247-1261, 1996.

[18] J. Sohn, T.G. Robertazzi, and S. Luryi, "Optimizing Computing Costs Using Divisible Load Analysis," *IEEE Trans. Parallel and Distributed Systems*, vol. 9, no. 3, pp. 225-234, Mar. 1998.

[19] K. Soman, R. Fraczkiewicz, C. Mumenthaler, B. von Freyberg, and T. Schaumann and W. Braun, FANTOM—(Fast Newton-Raphson Torsion Angle Minimizer), World Wide Web Document, http://www.scsb.utmb.edu/fantom/fm_home.html, 2003.

[20] J.R. Stiles, T.M. Bartol, M.M. Salpeter, and M.M. Salpeter, "Monte Carlo Simulation of Neuromuscular Transmitter Release Using MCell, a General Simulator of Cellular Physiological Processes," *Computational Neuroscience*, pp. 279-284, 1998.

[21] J. Watts and S. Taylor, "A Practical Approach to Dynamic Load Balancing," *IEEE Trans. Parallel and Distributed Systems*, vol. 9, pp. 235-248, 1998.

[22] D.B. West, *Introduction to Graph Theory.* Prentice Hall, 1996.

[23] W. Yu, "The Two-Machine Flow Shop Problem with Delays and the One-Machine Total Tardiness Problem," PhD thesis, Technishe Univ. Eidhoven, June 1996.

Olivier Beaumont received the PhD degree from the Université de Rennes in January 1999. He is currently an associate professor in the LaBRI laboratory in Bordeaux. His main research interests are parallel algorithms on distributed memory architectures.

Arnaud Legrand is currently a PhD student in the Computer Science Laboratory LIP at ENS Lyon. He is mainly interested in parallel algorithm design for heterogeneous platforms and in scheduling.

Yves Robert received the PhD degree from the Institut National Polytechnique de Grenoble in January 1986. He is currently a full professor in the Computer Science Laboratory LIP at ENS Lyon. He is the author of four books, 80 papers published in international journals, and 100 papers published in international conferences. His main research interests are scheduling techniques and parallel algorithms for clusters and grids. He is a member of the ACM and IEEE, and serves as an associate editor of *IEEE Transactions on Parallel and Distributed Systems*.

Scheduling Strategies for Master-Slave Tasking on Heterogeneous Processor Platforms

Cyril Banino, Olivier Beaumont, Larry Carter, *Fellow*, *IEEE*, Jeanne Ferrante, *Senior Member*, *IEEE*, Arnaud Legrand, and Yves Robert, *Member*, *IEEE*

Abstract—In this paper, we consider the problem of allocating a large number of independent, equal-sized tasks to a heterogeneous computing platform. We use a nonoriented graph to model the platform, where resources can have different speeds of computation and communication. Because the number of tasks is large, we focus on the question of determining the optimal steady state scheduling strategy for each processor (the fraction of time spent computing and the fraction of time spent communicating with each neighbor). In contrast to minimizing the total execution time, which is NP-hard in most formulations, we show that finding the optimal steady state can be solved using a linear programming approach and, thus, in polynomial time. Our result holds for a quite general framework, allowing for cycles and multiple paths in the interconnection graph, and allowing for several masters. We also consider the simpler case where the platform is a tree. While this case can also be solved via linear programming, we show how to derive a closed-form formula to compute the optimal steady state, which gives rise to a bandwidth-centric scheduling strategy. The advantage of this approach is that it can directly support autonomous task scheduling based only on information local to each node; no global information is needed. Finally, we provide a theoretical comparison of the computing power of tree-based versus arbitrary platforms.

Index Terms—Scheduling, heterogeneous, divisible load, steady-state, bandwidth-centric.

1 Introduction

In this paper, we deal with the master-slave paradigm on a heterogeneous platform, where resources have different speeds of computation and communication. More precisely, we tackle the problem of allocating a large number of independent, equal-sized tasks to a heterogeneous computing platform. We model a collection of heterogeneous resources and the communication links between them as the nodes and edges of an undirected *platform graph*. Each node is a computing resource (a processor, or a cluster, or whatever) capable of computing and/or communicating with its neighbors at (possibly) different rates.

We assume that one specific node, referred to as the *master*, initially holds (or generates the data for) a large collection of independent, identical tasks to be allocated on the grid. Think of a task as being associated with a file that contains all the data required for the execution of the task. The question for the master is to decide which tasks to execute itself, and how many tasks (i.e., task files) to forward to each of its neighbors. Due to heterogeneity, the neighbors may receive different amounts of work (maybe none for some of them). Each neighbor faces in turn the same dilemma: determine how many tasks to execute and how many to delegate to other processors.

- *C. Banino and O. Beaumont are with LaBRI, UMR CNRS 5800, Domaine Universitaire, 33405 Talence Cedex, France. E-mail: {Cyril.Banino, Olivier.Beaumont}@labri.fr.*
- *L. Carter and J. Ferrante are with the Computer Science Department, University of California, San Diego, 9500 Gilman Drive, La Jolla, CA 92093-0114. E-mail: {carter, ferrante}@cs.ucsd.edu.*
- *A. Legrand and Y. Robert are with LIP, UMR CNRS-ENS Lyon-INRIA 5668, École Normale Supérieure de Lyon, 69364 Lyon Cedex 07, France. E-mail: {Arnaud.Legrand, Yves.Robert}@ens-lyon.fr.*

Manuscript received 17 Feb. 2003; revised 7 July 2003; accepted 18 Aug. 2003.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0009-0203.

The master-slave scheduling problem is motivated by problems that are addressed by collaborative computing efforts such as SETI@home [16] and those distributed computing projects organized by companies such as Entropia [8]. Several papers have recently revisited the master-slave paradigm for processor clusters or grids, and we refer to Section 6 for comparison and discussion.

Because the number of tasks to be executed on the computing platform is expected to be very large (otherwise, why deploy the corresponding application on computational grids?), we target *steady state* optimization problems rather than standard *makespan* minimization problems. While minimizing the makespan, i.e., the total execution time, is a NP-hard problem in most practical situations [9], [18], [7], it turns out that the optimal steady state can be characterized very efficiently, with low-degree polynomial complexity. The optimal steady state is defined as follows (see Section 2.2): For each processor, determine the fraction of time spent computing and the fraction of time spent sending or receiving tasks along each communication link, so that the (averaged) overall number of tasks processed at each time-step is maximum.

The main contribution of this paper is the determination of the optimal steady state in a wide general framework. We derive three main results that we summarize below:

- The first result holds for arbitrary platform graphs whose underlying interconnection network may be very complex and, in particular, may include multiple paths and cycles (just as the Internet does). The master may well need to send tasks along multiple paths to properly feed a very fast but remote computing resource. In this context, we compute the optimal steady state using a linear programming formulation, which nicely encompasses the situation where there are several masters instead of a single

one. We also show (see Section 5.1) that steady state scheduling is asymptotically optimal, among all possible schedules (not only periodic ones).

- The second result holds for a tree-shaped platform graph, i.e., when the underlying interconnection network is an oriented tree rooted at the master. For such platforms, we derive a closed-form formula that characterizes the optimal steady state, which can then be computed via a simple bottom-up traversal of the tree. Interestingly, the optimal steady state is achieved through a *bandwidth-centric* scheduling strategy: If enough bandwidth is available to the node, then all children are kept busy; if bandwidth is limited, then tasks should be allocated only to children which have sufficiently fast communication times, in order of fastest communication time. The advantage of the bandwidth-centric approach is that it can directly support autonomous task scheduling based only on information local to each node; no global information is needed.
- The third result provides a comparison of the computing power of tree-shaped platforms versus arbitrary platform graphs. Given a network topology that may well include cycles and multiple paths, how does one extract the "best" spanning tree, i.e., the spanning tree rooted at the master which allows for the maximum number of tasks to be processed by all the computing resources? We show that the problem of extracting the optimal spanning tree is NP-complete. Even worse, we show that there exist heterogeneous networks for which the optimal spanning tree has a throughput which is arbitrarily bad compared with the throughput that can be achieved by the optimal (multiple-path) solution.

The rest of the paper is organized as follows: Section 2 is devoted to arbitrary platform graphs. We introduce the base model of communication and computation in Section 2.1. We formally state the equations governing the steady state in Section 2.2. In Section 2.3, we show how to determine the optimal steady state for an arbitrary platform graph, using a linear programming approach. Next, in Section 3, we deal with tree-shaped platform graphs: We provide a closed-form formula and a constructive method to compute the optimal steady state for such a platform. In Section 4, we discuss different hypotheses on the processing and communication capabilities of the computing resources: We show how to handle the case of processors operating with different characteristics and, in particular, different capabilities of overlapping communications with (independent) communications. In Section 5.1, we prove that steady state scheduling is always asymptotically optimal. Section 5.2 provides the negative complexity results stated above for the problem of extracting the optimal spanning tree (that of maximum steady state throughput) out of a given general platform. We briefly survey related work in Section 6. Finally, we give some remarks and conclusions in Section 7.

2 Optimal Steady State for General Platform Graphs

In this section, we formally state the optimization problem to be solved. We start with the architectural model in Section 2.1. Next, we explain all the equations governing the steady state operation on an arbitrary platform graph (Section 2.2). Then, in Section 2.3, we show how to cast the problem of determining the optimal steady state in terms of a linear programming problem to be solved in rational numbers (hence, of polynomial complexity). We deal with the extension to several masters in Section 2.3.3. We conclude this section by working out a little example in Section 2.4.

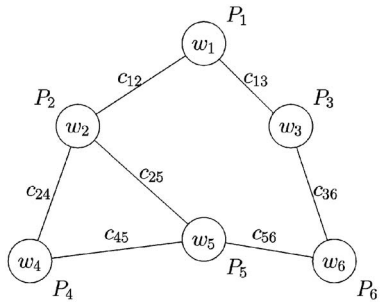


Fig. 1. A graph labeled with node (computation) and edge (communication) weights.

2.1 Architectural Model

The target architectural/application framework is represented by a node-weighted edge-weighted graph $G = (V, E, w, c)$, as illustrated in Fig. 1. Let $p = |V|$ be the number of nodes. Each node $P_i \in V$ represents a computing resource of weight w_i, meaning that node P_i requires w_i units of time to process one task (so the smaller w_i, the faster the processor node P_i). There is a *master* processor, i.e., a node P_m which plays a particular role. P_m initially holds the data for a large (say unlimited) collection of independent tasks to be executed. Think of each task as being associated with a file that contains all the data required for the execution of the task. Tasks are atomic; their computation or communication cannot be preempted.

Each edge $e_{ij} : P_i \to P_j$ is labeled by a value c_{ij} which represents the time needed to communicate one (data file associated with a) task between P_i and P_j, in either direction: We assume that the link between P_i and P_j is bidirectional and symmetric, i.e., that it takes the same amount of time to send (the data for) one task from P_i to P_j as in the reverse direction, from P_j to P_i. A variant would be to assume two unidirectional links, one in each direction, with possibly different label values, and we explain below how to modify the formulas to handle this variant. If there is no communication link between P_i and P_j, we let $c_{ij} = +\infty$ so that $c_{ij} < +\infty$ means that P_i and P_j are neighbors in the communication graph.

Note that we can include in c_{ij} the time needed for the receiving processor to return the result to the sending processor when it is finished. For the purpose of computing steady state behavior, it does not matter what fraction of the communication time is spent sending a problem and what fraction is spent receiving the results. To simplify the exposition, we will henceforth assume that all the time is spent sending the task data and no time is needed to communicate the results back. We assume that all w_i are positive rational numbers. We disallow $w_i = 0$ since it would permit node P_i to perform an infinite number of

tasks, but we allow $w_i = +\infty$; then, P_i has no computing power, but can still forward tasks to other processors. Similarly, we assume that all c_{ij} are positive rational numbers (or equal to $+\infty$ if there is no link between P_i and P_j).

There are several scenarios for the operation of the processors, which are surveyed in Section 4. In this section, we concentrate on the *full overlap, single-port model*, which we also call the *base model*. In this model, a processor node can simultaneously receive data from one of its neighbors, perform some (independent) computation, and send data to one of its neighbors. At any given time-step, there are at most two communications involving a given processor, one sent and the other received.

We state the communication model more precisely: If P_i sends (the data file associated with) a task to P_j at time-step t, then: 1) P_j cannot start executing or sending this task before time-step $t + c_{ij}$; 2) P_j cannot initiate another receive operation before time-step $t + c_{ij}$ (but, it can perform a send operation and independent computation); and 3) P_i cannot initiate another send operation before time-step $t + c_{ij}$ (but, it can perform a receive operation and independent computation).

2.2 Steady State Operation

Given the resources of a weighted graph G operating under the base model, we aim at determining the best steady state scheduling policy. After a start-up phase, we want the resources to operate in a periodic mode.

To formally define the steady state, we need some notation. Let $n(i)$ denote the index set of the neighbors of processor P_i. During one time unit:

- α_i is the fraction of time spent by P_i computing.
- s_{ij} is the fraction of time spent by P_i sending tasks to each neighbor processor P_j, for $j \in n(i)$, i.e., for each $e_{ij} \in E$.
- r_{ij} is the fraction of time spent by P_i receiving tasks from each neighbor processor P_j, for $j \in n(i)$, i.e., for each $e_{ij} \in E$.

We search for rational values for all these variables. The first set of constraints is that they all must belong to the interval $[0, 1]$, as they correspond to the fraction of activity during one time unit:

$$\forall i, 0 \leq \alpha_i \leq 1, \tag{1}$$

$$\forall i, \forall j \in n(i), 0 \leq s_{ij} \leq 1, \tag{2}$$

$$\forall i, \forall j \in n(i), 0 \leq r_{ij} \leq 1. \tag{3}$$

The second set of constraints is that the number s_{ij}/c_{ij} of tasks sent by P_i to P_j is equal to the number of tasks r_{ji}/c_{ij} received by P_j from P_i:

$$\forall e_{ij} \in E, s_{ij} = r_{ji}. \tag{4}$$

Remember that the communication graph is assumed to be symmetric: $e_{ij} \in E \Rightarrow e_{ji} \in E$ and, therefore, we also have $s_{ji} = r_{ij}$. It may well be the case that each link will only be used in one direction in the final solution, i.e., that either r_{ij} or s_{ij} will be zero, but we cannot guarantee this a priori.

There are specific constraints for the base model:

One-port model for outgoing communications. Because send operations to the neighbors of P_i are assumed to be sequential, we have the equation

$$\forall i, \sum_{j \in n(i)} s_{ij} \leq 1. \tag{5}$$

One-port model for incoming communications. Because receive operations from the neighbors of P_i are assumed to be sequential, we have the equation

$$\forall i, \sum_{j \in n(i)} r_{ij} \leq 1. \tag{6}$$

Full overlap. Because of the full overlap hypothesis, there is no further constraint on α_i: $0 \leq \alpha_i \leq 1$ and $\alpha_i = 1$ would mean that P_i is kept busy processing tasks all the time.

Limited bandwidth. This constraint is due to our hypothesis that the same link e_{ij} may be used in both directions simultaneously. We have to guarantee that the link bandwidth is not exceeded. The constraint translates into:

$$\forall e_{ij} \in E, s_{ij} + r_{ij} \leq 1. \tag{7}$$

We can slightly reformulate (7) by introducing b_{ij}, the link bandwidth, expressed in tasks per second (i.e., $b_{ij} = 1/c_{ij}$). In each time unit, there are $\frac{s_{ij}}{c_{ij}}$ tasks sent by P_i to P_j, and $\frac{r_{ij}}{c_{ij}}$ tasks received by P_i to P_j, so constraint (7) can be written

$$\forall e_{ij} \in E, \frac{s_{ij}}{c_{ij}} + \frac{r_{ij}}{c_{ij}} \leq b_{ij}.$$

Conservation laws. The last constraints deal with *conservation laws*: For every processor P_i which is not the master, the number of tasks received by P_i, i.e., $\sum_{j \in n(i)} \frac{r_{ij}}{c_{ij}}$, should be equal to the number of tasks that P_i consumes itself, i.e., $\frac{\alpha_i}{w_i}$, plus the number of tasks forwarded to its neighbors, i.e., $\sum_{j \in n(i)} \frac{s_{ij}}{c_{ij}}$. We derive the equation:

$$\forall i \neq m, \sum_{j \in n(i)} \frac{r_{ij}}{c_{ij}} = \frac{\alpha_i}{w_i} + \sum_{j \in n(i)} \frac{s_{ij}}{c_{ij}}. \tag{8}$$

It is important to understand that (8) really applies to the steady state operation. We can assume an initialization phase, during which tasks are forwarded to processors and no computation is performed. Then, during each time-period in steady state, each processor can simultaneously perform some computations, and send/receive some other tasks. This is why (8) is sufficient; we do not have to detail which operation is performed at which time-step because the tasks all commute, they are mutually independent.

Equation (8) does not hold for the master processor P_m because it holds an infinite number of tasks. Without loss of generality, we can enforce that $r_{mj} = 0$ for all $j \in n(m)$: The master does not need to receive any tasks from its neighbors. Note that it would be easy to handle unidirectional links: If $e_{ij} : P_i \to P_j$ is unidirectional (that is, if for some reason P_i can send tasks to P_j, but not vice versa), we let $r_{ij} = s_{ji} = 0$ and we suppress (7), which is automatically fulfilled. Similarly, it is straightforward to replace each bidirectional link e_{ij} by two

oriented arcs a_{ij} (from P_i to P_j) and a_{ji} (from P_j to P_i), respectively, weighted with c_{ij} and c_{ji}.

2.3 Solution

2.3.1 Optimal Throughput

The equations above constitute a linear programming problem, whose objective function is the number of tasks consumed within one unit of time, i.e., the throughput $n_{\text{task}}(G) = \sum_i \frac{\alpha_i}{w_i}$. Here is a summary:

MASTER SLAVE SCHEDULING PROBLEM MSSG(G)

$$\textbf{Maximize } n_{\text{task}}(G) = \sum_{i=1}^{p} \frac{\alpha_i}{w_i},$$

subject to

$$\begin{cases} \forall i, & 0 \le \alpha_i \le 1 \\ \forall i, \forall j \in n(i), & 0 \le s_{ij} \le 1 \\ \forall i, \forall j \in n(i), & 0 \le r_{ij} \le 1 \\ \forall e_{ij} \in E, & s_{ij} = r_{ji} \end{cases} \quad \begin{cases} \forall i, & \sum_{j \in n(i)} s_{ij} \le 1 \\ \forall i, & \sum_{j \in n(i)} r_{ij} \le 1 \\ \forall e_{ij} \in E, & s_{ij} + r_{ij} \le 1 \\ \forall i \ne m, & \sum_{j \in n(i)} \frac{r_{ij}}{c_{ij}} = \frac{\alpha_i}{w_i} + \sum_{j \in n(i)} \frac{s_{ij}}{c_{ij}} \\ \forall j \in n(m), & r_{mj} = 0 \end{cases}$$

Note that we can enforce $\alpha_m = 1$ because the master will keep on processing tasks all the time, but we do not have to because this condition will automatically be fulfilled by the solution. We can state the first main result of this paper.

Theorem 1. *The solution to the previous linear programming problem provides the optimal solution to* MSSG(G).

Because we have a linear programming problem in rational numbers, we obtain rational values for all variables in polynomial time (polynomial in $|V| + |E|$, the size of the heterogeneous platform). When we have the optimal solution, we take the least common multiple of the denominators and, thus, we derive an integer period T for the steady state operation.

Finally, we point out that we can restrict to solutions where each link is used only in one direction. Although there may exist optimal solutions of MSSG(G) for which it is not the case, we can always transform such solutions into solutions where each link is used only in one direction (without changing the throughput): if both r_{ij} and s_{ij} are nonzero, with, say, $r_{ij} \ge s_{ij}$, use $r'_{ij} = r_{ij} - s_{ij}$ and $s'_{ij} = 0$ to derive an equivalent solution.

2.3.2 Reconstructing the Schedule

There are several subtle points when reconstructing the actual periodic schedule. Once the linear programming problem is solved, we get the period T of the schedule, and the integer number of messages going through each link. First, because it arises from the linear program, $\log T$ is indeed a number polynomial in the problem size, but T itself is not. Hence, describing what happens at every time-step during the period would be exponential in the problem size, and we need a more "compact" description of the schedule. Second, we need to exhibit an orchestration of the message transfers where only independent communications, i.e., involving disjoint pairs of senders and receivers, can take place simultaneously.

Both problems are solved as follows: From our platform graph G, and using the result of the linear program, we build a bipartite graph: For each node P_i in G, we create two nodes P_i^{send} and P_i^{recv}. For each communication from P_i to P_j, we insert an edge between P_i^{send} and P_j^{recv}, which is weighted by the length of the communication. We are looking for a decomposition of this bipartite graph into a set of subgraphs where a node (sender or receiver) is occupied by at most one communication task. This means that at most one edge reaches each node in the subgraph. In other words, only communications corresponding to a matching in the bipartite graph can be performed simultaneously, and the desired decomposition of the graph is in fact an edge coloring. The weighted edge coloring algorithm of [15, vol. A, chapter 20] provides in time $\mathcal{O}(|E|^2)$ a polynomial number of matchings, which are used to perform the different communications, and which provides the desired polynomial-size description of the schedule. See [5] for further details. We point out that reconstructing the final schedule is much easier for tree-shaped platforms, as shown in Section 3.

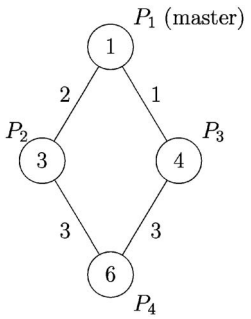


Fig. 2. An example with four processors.

Notice that it is necessary for each node to be able to hold in a buffer some number of tasks that it has received, but not yet computed or sent. Also, depending on the schedule, it may be necessary to preload some number of tasks into the node during the initialization period. The maximum number of such buffered tasks is at most T per node. The issue of limiting the number of buffered tasks to a more practical number is explored in [13].

2.3.3 With Several Masters

The extension for several masters is straightforward. Assume that there are k masters P_{m_1}, P_{m_2}, ..., P_{m_k}, each holding (the initial data for) a large collection of tasks. For each index m_q, $1 \le q \le k$:

1. Suppress (8) for $i = m_q$ (the conservation law does not apply to a master).
2. Add the constraints $r_{m_q,j} = 0$ for all $j \in m(q)$ (a master does not need to receive any task).

We then solve the new MSSG(G) problem.

2.4 Example

Consider the toy example of Fig. 2, with $p = 4$ processors (P_1 is the unique master). If we feed the values w_i and c_i into the linear program, and compute the solution using a tool such as the Maple simplex package, we obtain the optimal throughput $n_{\text{task}}(G) = \frac{7}{4}$. This means that the whole platform is equivalent to a single processor with processing

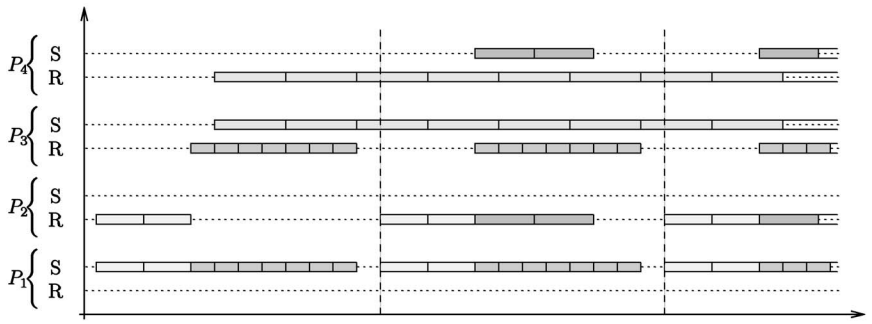


Fig. 3. Steady state for the example with four processors.

capability $w = \frac{1}{n_{\text{task}}(G)} = \frac{4}{7}$, i.e., capable of processing seven tasks every four seconds.

With the values of α_i, s_{ij}, and r_{ij} returned in the solution of the linear program, we retrieve the periodic steady state behavior. Every 12 time-units:

- The master processor P_1 computes 12 tasks ($\alpha_1 = 1$), sends two tasks to P_2 (in six time-steps, $s_{12} = 1/3$), and sends seven tasks to P_3 (in seven time-steps, $s_{13} = 7/12$).
- Processor P_3 receives seven tasks from the master P_1 (in seven times-steps, $r_{31} = 7/12$), computes three tasks ($\alpha_3 = 1$), and sends four tasks to P_4 (in 12 time-steps, $s_{34} = 1$).
- Processor P_4 receives four tasks from P_3 (in 12 time-steps, $r_{43} = 1$), computes two tasks ($\alpha_4 = 1$), and sends two tasks to P_2 (in six time-steps, $s_{42} = 1/2$).
- Processor P_2 receives four tasks, two from the master P_1 (in 4 time-steps, $r_{21} = 1/3$), and two from P_4 (in six time-steps, $r_{24} = 1/2$), and it computes four tasks ($\alpha_2 = 1$).

This makes a total of $12 + 3 + 2 + 4 = 21$ tasks every 12 time-steps, and we do retrieve the value $n_{\text{task}}(G) = \frac{21}{12} = \frac{7}{4}$. This steady state is illustrated in Fig. 3. Note that all processors are executing tasks all the time, so the solution achieves a full utilization of the computing resources. It is interesting to point out that P_2 receives its tasks along two paths, the first half directly from the master, and the second half being forwarded through P_3 and P_4.

In the introduction, we briefly mentioned the difficulty of extracting a spanning tree of high throughput from a given interconnection graph, and we come back to this point in Section 5. We can perform an exhaustive search for our little example because there are only four possible trees rooted in P_1. To compute the throughput of a given tree, we can either use the linear programming approach, or traverse the tree using the bandwidth-centric algorithm of Section 3 (with a cost linear in the processor number). In the example, we obtain the following results:

1. If we suppress the edge between P_1 and P_2, the throughput of the corresponding tree T_1 is $n_{\text{task}}(T_1) = \frac{38}{24}$.
2. If we suppress the edge between P_1 and P_3, the throughput of the corresponding tree T_2 is $n_{\text{task}}(T_2) = \frac{36}{24}$.
3. If we suppress the edge between P_2 and P_4, the throughput of the corresponding tree T_3 is $n_{\text{task}}(T_3) = \frac{41}{24}$.
4. If we suppress the edge between P_3 and P_4, the throughput of the corresponding tree T_4 is $n_{\text{task}}(T_4) = \frac{39}{24}$.

We see that the third tree T_3 is the best one, with a throughput $n_{\text{task}}(T_3) = \frac{41}{24}$ very close to the optimal solution $n_{\text{task}}(G) = \frac{42}{24}$ for the whole graph.

3 Bandwidth-Centric Strategy for Tree Shaped Platforms

In this section, we deal with tree-shaped platforms. The root of the tree is always the master. For such platforms, we derive a closed-form expression of the solution of the linear program MSSG(G). We also show that a simple traversal of the tree enables to compute the solution of MSSG(G) in linear time (linear in the size of the platform).

3.1 Star Graph

We start with simple star graphs before dealing with arbitrary tree graphs. A star graph F, as shown in Fig. 4, consists of a node P_0 and its k children $P_1, \ldots, P_k$. In the *full overlap, single-port* model, P_0 can communicate with a single child at a time: It needs c_i units of time to communicate a task to child P_i. Concurrently, P_0 can receive data from its own parent, say P_{-1}, requiring c_0 time per task. We give three examples in Figs. 5, 6, and 7: In the first, all children operate at full rate, and in the latter two, the communication bandwidth is the limiting factor.

Proposition 1. *With the above notations, the minimal value of $n_{\text{task}}(F)$ for the star graph F is obtained as follows:*

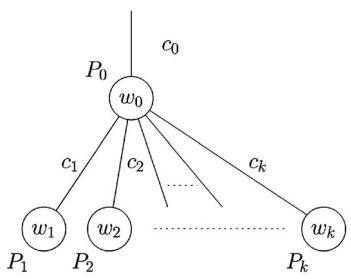


Fig. 4. Star graph.

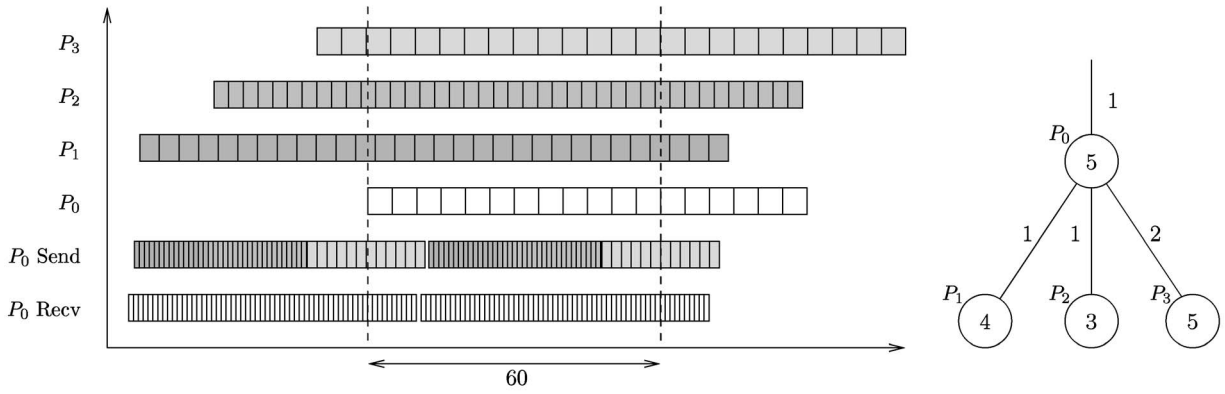


Fig. 5. First example without saturation of the communication bandwidth: All children can be kept fully active.

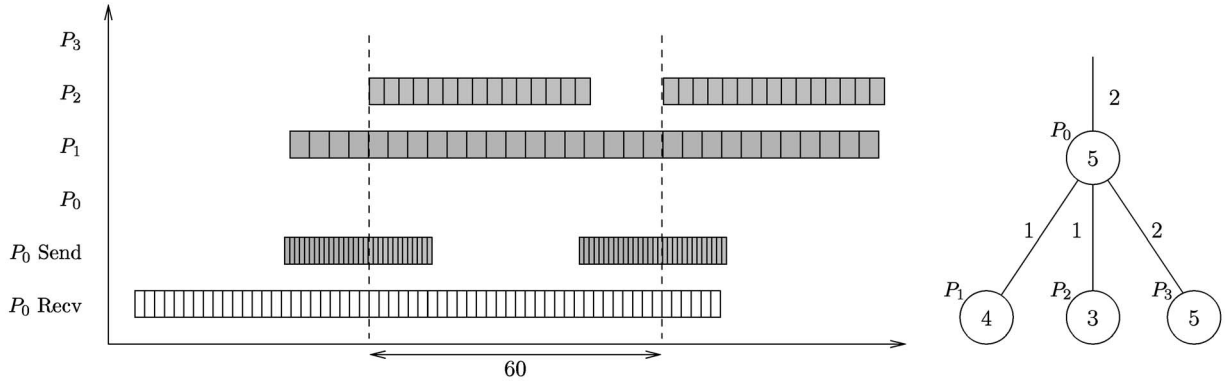


Fig. 6. Second example with saturation of the communication bandwidth: Some children are partially idle due to the low bandwidth between P_0 and its parent.

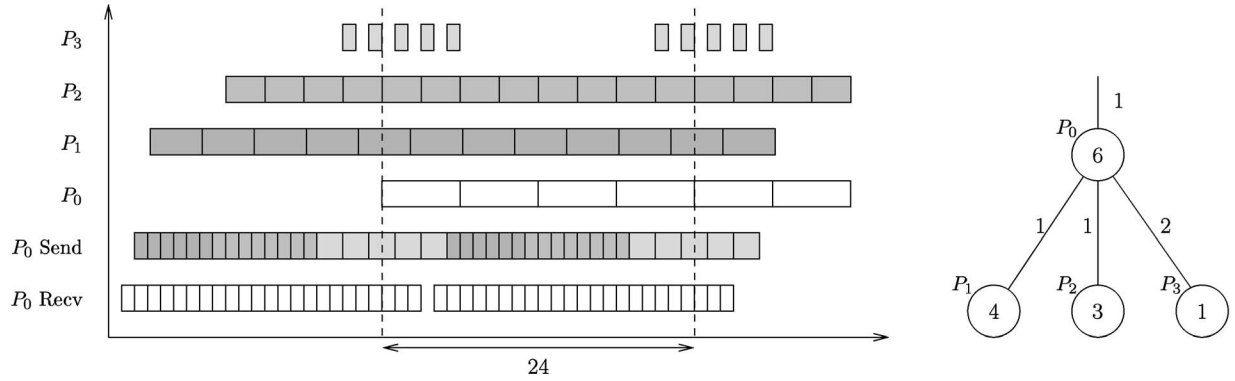


Fig. 7. Third example with saturation of the communication bandwidth: child P_3 is partially idle due to its high computation speed.

1. *Sort the children by increasing communication times. Renumber them so that* $c_1 \leq c_2 \ldots \leq c_k$.
2. *Let* p *be the largest index so that* $\sum_{i=1}^{p} \frac{c_i}{w_i} \leq 1$. *If* $p < k$, *let* $\varepsilon = 1 - \sum_{i=1}^{p} \frac{c_i}{w_i}$; *otherwise, let* $\varepsilon = 0$.
3. *Then,* $n_{\text{task}}(F) = \min\left(\frac{1}{c_0}, \frac{1}{w_0} + \sum_{i=1}^{p} \frac{1}{w_i} + \frac{\varepsilon}{c_{p+1}}\right)$.

Intuitively, the processors cannot consume more tasks than sent by P_{-1}, hence, the first term of the minimum. For the second term, when $p = k$, the result is expected: It basically says that children can be fed with tasks fast enough so that they are all kept computing steadily. However, if $p < k$, the result is surprising: In the situation when the communication bandwidth is limited, some children will partially starve: In the optimal solution, these are those with slow communication rates, whatever their processing speeds. In other words, a slow processor with a fast communication link is to be preferred to a fast processor with a slow communication link.

Proof. We use the same notation as in Section 2.3. For $0 \leq i \leq k$, α_i is the fraction of time spent by P_i computing. For $0 \leq i \leq k$, we let s_i (instead of $s_{0,i}$) be the fraction of time spent by P_0 sending tasks to child P_i. Note that we know that tasks flow from P_0 to its children, hence, $r_{0,i} = 0$ (where $r_{0,i}$ is the fraction of time spent by P_0 receiving tasks from P_i). Finally, let r_{-1} be the fraction of time spent by P_0 receiving tasks from its own parent. We have the following linear program:

$$\textbf{Maximize } \frac{\alpha_0}{w_0} + \sum_{i=1}^{k} \frac{\alpha_i}{w_i}$$

$$\textbf{subject to } \begin{cases} 0 \le \alpha_i \le 1 \text{ for } 0 \le i \le k \\ 0 \le r_{-1} \le 1 \\ \sum_{i=1}^{k} s_i \le 1 \end{cases} \quad \left| \quad \begin{array}{l} \frac{r_{-1}}{c_0} = \frac{\alpha_0}{w_0} + \sum_{i=1}^{k} \frac{s_i}{c_i} \\ \frac{s_i}{c_i} = \frac{\alpha_i}{w_i} \text{ for } 0 \le i \le k \end{array} \right.$$

The last equation stands for the conservation equation for each child P_i: Because no task is forwarded, each received task is consumed in place. We normalize these equations by introducing the rates R_i: $R_{-1} = \frac{s_{-1}}{c_0}$ is the rate of tasks per second received from the parent P_{-1}, $R_0 = \frac{\alpha_0}{w_0}$ is the rate they are executed in the parent node P_0, and $R_i = \frac{s_i}{c_i} = \frac{\alpha_i}{w_i}$, for $1 \le i \le k$, is the rate they are sent to and executed on the ith child. We obtain the following formulation:

$$\textbf{Base Problem : Maximize } \sum_{i=0}^{k} R_i, \textbf{ subject to}$$

$$(\textbf{B0})\ R_i \le \tfrac{1}{w_i} \text{ for } 0 \le i \le k \quad (\textbf{B2})\ R_i \le \tfrac{1}{w_i} \text{ for } 0 \le i \le k$$

$$(\textbf{B1})\ R_{-1} = \textstyle\sum_{i=0}^{k} R_i \le \tfrac{1}{c_0} \quad (\textbf{B3})\ \textstyle\sum_{i=1}^{k} R_i c_i \le 1$$

Let R be the solution of the Base Problem. We claim that $R = \min\left(\frac{1}{c_0}, \frac{1}{w_0} + S\right)$ (unless $c_0 = 0$, in that case $R = \frac{1}{w_0} + S$), where S is the solution to the following problem:

$$\textbf{Auxiliary problem : Maximize } \sum_{i=1}^{k} R_i, \textbf{ subject to}$$

$$(\mathbf{1})\ R_i w_i \le 1 \text{ for } 1 \le i \le k, \qquad (\mathbf{2})\ \sum_{i=1}^{k} R_i c_i \le 1.$$

Because the auxiliary problem is less constrained than the original one, we immediately have that $\frac{1}{w_0} + S \ge R$. We also have $\frac{1}{c_0} \ge R$ because of (B0) and (B1), hence, $\min\left(\frac{1}{c_0}, \frac{1}{w_0} + S\right) \ge R$. To show the reverse inequality, there are two cases, according to the value of $\min\left(\frac{1}{c_0}, \frac{1}{w_0} + S\right)$. Assume first that $\min\left(\frac{1}{c_0}, \frac{1}{w_0} + S\right) = \frac{1}{c_0}$. Let $(R_1, \ldots, R_k)$ be the optimal solution of the auxiliary problem: $S = \sum_{i=1}^{k} R_i$ and $\frac{1}{w_0} + S \ge \frac{1}{c_0}$. Let $\alpha = \frac{\frac{1}{c_0}}{\frac{1}{w_0}+S} \le 1$. Then, $\left(\frac{1}{c_0}, \frac{\alpha}{w_0}, \alpha R_1, \ldots, \alpha R_k\right)$ is a solution to the base problem whose objective function is equal to $\frac{1}{c_0}$. Therefore, $\frac{1}{c_0} \le R$.

Assume now that $\min\left(\frac{1}{c_0}, \frac{1}{w_0} + S\right) = \frac{1}{w_0} + S$. Let $(R_1, \ldots, R_k)$ be the optimal solution of the auxiliary problem: $S = \sum_{i=1}^{k} R_i$ and $\frac{1}{w_0} + S \le \frac{1}{c_0}$. Let $R_0 = \frac{1}{w_0}$ and $R_{-1} = \sum_{i=0}^{k} R_i$. Then, $(R_{-1}, R_0, R_1, \ldots, R_k)$ is a solution to the base problem (note that (B1) is satisfied because of the hypothesis). Hence, $\frac{1}{w_0} + S \le R$. This concludes the reduction to the auxiliary problem.

We can now come to the solution of the auxiliary problem. As in the statement of the theorem, let p be the largest index so that $\sum_{i=1}^{p} \frac{c_i}{w_i} \le 1$. Let $R_i^* = \frac{1}{w_i}$ for $1 \le i \le p$. If $p < k$, let $R_{p+1}^* = \frac{\varepsilon}{c_{p+1}}$, where $\varepsilon = 1 - \sum_{i=1}^{p} \frac{c_i}{w_i}$. If $p + 1 < k$, let $R_i^* = 0$ for $p + 2 \le i \le k$. We claim that (R_i^*) is the optimal solution of the optimization problem:

- First, it is indeed a solution. We have $R_{p+1}^* \le \frac{1}{w_{p+1}}$ when $p < k$. This comes directly from the definition of p: Since $\sum_{i=1}^{p+1} \frac{c_i}{w_i} > 1$, we have $\varepsilon = 1 - \sum_{i=1}^{p} \frac{c_i}{w_i} < \frac{c_{p+1}}{w_{p+1}}$, hence, $\frac{\varepsilon}{c_{p+1}} \le \frac{1}{w_{p+1}}$. As for (2), $\sum_{i=1}^{j} R_i c_i = \sum_{i=1}^{j} \frac{c_i}{w_i} \le 1$ for all $j \le p$, by definition of p. And. if $p < k$, $\sum_{i=1}^{p+1} R_i c_i = 1$, by definition of R_{p+1}.
- Second, it is the solution that maximizes the objective function. To see this, consider all the optimal solutions, i.e., all the solutions that achieve the optimal value of the objective function. Among these optimal solutions, consider one solution (R_i) such that R_1 is maximal. Assume by contradiction that $R_1 < R_1^* = \frac{1}{w_1}$. Then, there exists at least one index $j > 2$ such that $R_j > R_j^*$; otherwise, the solution would not be optimal. Now, since the c_i are sorted, we have $c_1 \le c_j$. We do not change the value of the objective function if we let $R_1 = R_1 + \tau$ and $R_j = R_j - \tau$, where τ is an arbitrary small nonnegative rational number. However, we do have a new solution to the optimization problem, because $(R_1 + \tau)c_1 + (R_j - \tau)c_j \le R_1 c_1 + R_j c_j$. Hence, we have an optimal solution with a larger R_1 than the original one, a contradiction. Hence, we have shown that there exist optimal solutions such that $R_1 = R_1^*$. We restrict to such solutions without loss of generality and we iterate the process: We finally derive that (R_i^*) is an optimal solution.

The optimal solution of the auxiliary problem is $S = \sum_{i=1}^{k} R_i = \sum_{i=1}^{p} \frac{1}{w_i} + \frac{\varepsilon}{c_{p+1}}$.

The optimal solution of the base problem is $R = \min\left(\frac{1}{c_0}, \frac{1}{w_0} + S\right)$, which establishes our claim.

A less formal (but much shorter) proof of Proposition 1 is the following: Sorting the c_i and feeding as many tasks as possible to the children taken in that order maximizes the number of tasks that are communicated to the children; hence, the number of tasks that are processed by the children. Add those processed by the parent, and take the minimum with the input rate to derive the optimal value. Note that the proof in Proposition 1 is fully constructive: The number of tasks to be computed by the parent and to be sent to each child is directly computed from the optimal solution (R_i^*). □

3.2 Arbitrary Tree Graphs

The best allocation of tasks to processor nodes is determined using a bottom-up traversal of the tree:

Proposition 2. *Let T be an arbitrary tree graph. The maximal value of $n_{\text{task}}(F)$ for the whole tree T is obtained as follows:*

1. *Consider any subtree F consisting of several leaves and their parent. Replace this tree with a single node whose weight is $w = \frac{1}{n_{\text{task}}(F)}$, where $n_{\text{task}}(F)$ is given by Proposition 1.*
2. *Iterate the process until there remains a single node.*

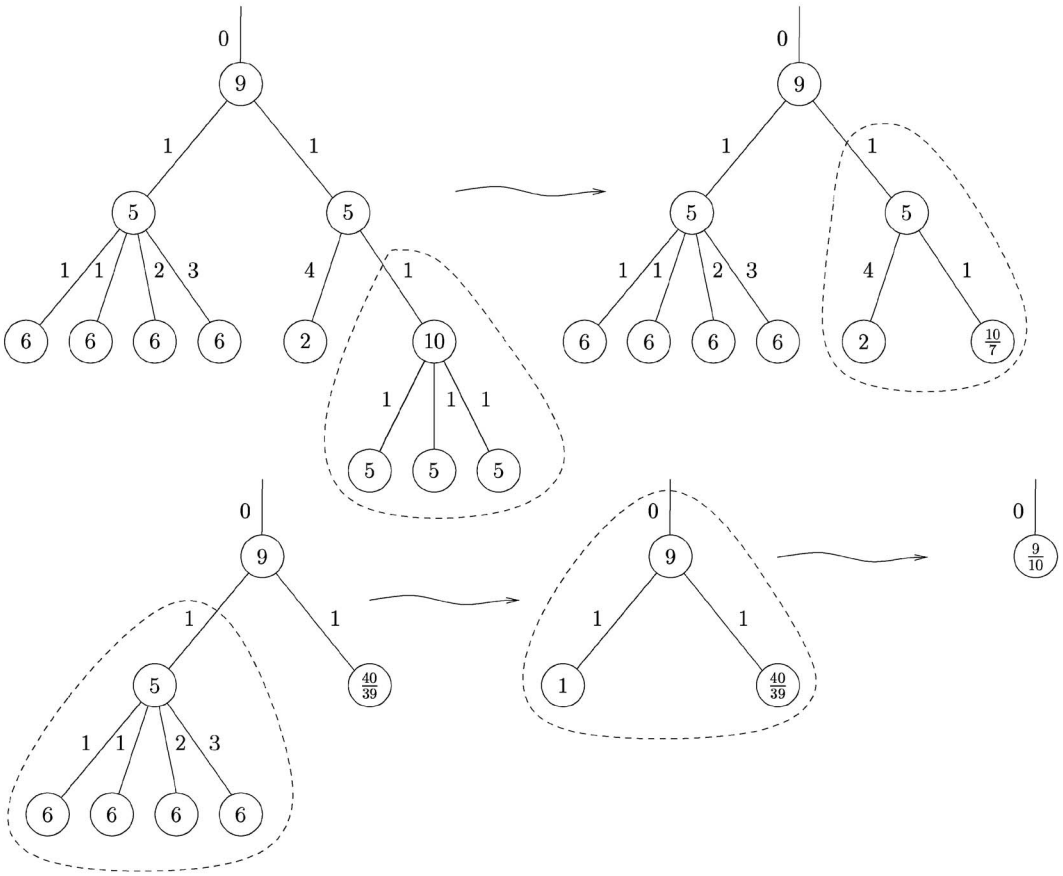


Fig. 8. Computing the best allocation for a tree graph.

Then, the minimal value is equal to the weight of the single node.

Proof. The proof is immediate: In steady state, a star graph F consisting of a parent and several leaves behaves exactly as a single node of weight $w = \frac{1}{n_{\text{task}}(F)}$, where $n_{\text{task}}(F)$ is determined by Proposition 1. For the root node, we can assume a link from a virtual parent with infinite capacity, i.e., $c_0 = 0$.

Again, the proof is fully constructive and the optimal task allocation is computed using the bottom-up approach. Fig. 8 is a small example. □

4 Other Operating Models

In this section, we discuss models of operation other than the full-overlap, single-port model, or base model. For each new model, we show which equations to modify in the linear program MSSG(G). In other words, we perform a series of model reductions to convert each instance of a processor operating a given model into an instance with the base model.

We list different models, starting from the most powerful machines down to purely sequential processors.

$\mathcal{M}(r^*||s^*||w)$: **Full overlap, multiple-port**. In this first model, a processor node can simultaneously receive data from all its neighbors, perform some (independent) computation, and send data to all of its neighbors. This model is not realistic if the number of neighbors is large.

$\mathcal{M}(r||s||w||)$: **Full overlap, single-port**. In this second model, a processor node can simultaneously receive data from one neighbor, perform some (independent) computation, and send data to one neighbor. At any given time-step, there are at most two communications taking place, one incoming and one outgoing. This model is representative of a large class of modern machines and is the *base model* which we have already dealt with.

$\mathcal{M}(r||s,w)$: **Receive-in-Parallel, single-port**. In this third model, as in the next two, a processor node has one single level of parallelism: It can perform two actions simultaneously. In the $\mathcal{M}(r||s,w)$ model, a processor can simultaneously receive data from one neighbor, and either perform some (independent) computation, or send data to one neighbor.

$\mathcal{M}(s||r,w)$: **Send-in-Parallel, single-port**. In this fourth model, a processor node can simultaneously send data to one neighbor and either perform some (independent) computation, or receive data from one neighbor.

$\mathcal{M}(w||r,s)$: **Work-in-Parallel, single-port**. In this fifth model, a processor node can simultaneously compute and execute a single communication, either sending to or receiving from one neighbor.

$\mathcal{M}(r,s,w)$: **No internal parallelism**. In this sixth and last model, a processor node can only do one thing at a time:

either receiving from one neighbor, or computing, or sending data to one neighbor.

Reduction for $\mathcal{M}(r^*||s^*||w)$, the Full overlap, multiple-port model. In this model, we allow for an unlimited number of simultaneous communications, either incoming or outgoing. It is quite easy to take this new constraint into account: Simply suppress (5) and (6) in the linear program! Indeed, under the new model, (2) and (3) are sufficient to characterize the activity of each processor.

Instead of allowing an unlimited number of simultaneous communications, we could be more restrictive and restrict each processor to k_1 incoming and k_2 outgoing communications. In other words, there are k_1 receiving ports and k_2 sending ports. Let r_{ij}^k be the time spent by processor P_i to receive tasks from processor P_j on receiving port k for $1 \leq k \leq k_1$. Similarly, let s_{ij}^k be the time spent by P_i to send tasks to P_j on sending port k, for $1 \leq k \leq k_2$. The new constraints simply are

$$\forall i, \forall k, 1 \leq k \leq k_1, 0 \leq \sum_{j \in n(i)} r_{ij}^k \leq 1, \tag{9}$$

$$\forall i, \forall k, 1 \leq k \leq k_2, 0 \leq \sum_{j \in n(i)} s_{ij}^k \leq 1. \tag{10}$$

Reduction for $\mathcal{M}(r||s, w)$, the Receive-in-Parallel, single port model. This model is less powerful than the base model: the processor can simultaneously receive a task from one of its neighbors, and either perform some computation, or send a task to one of its neighbors. To take this new constraint into account, simply replace (5) and (1) by

$$\forall i, \alpha_i + \sum_{j \in n(i)} s_{ij} \leq 1 \tag{11}$$

Reduction for $\mathcal{M}(s||r, w)$, the Send-in-Parallel, single port model. In this model, a processor can simultaneously send a task to one of its neighbors, and either perform some computation, or receive a task from one of its neighbors. To take this new constraint into account, simply replace (6) by

$$\forall i, \alpha_i + \sum_{j \in n(i)} r_{ij} \leq 1. \tag{12}$$

Reduction for $\mathcal{M}(w||r, s)$, the Work-in-Parallel, single port model. In this model, a processor can simultaneously perform some computation, and either receive from, or send a task to, one of its neighbors, To take this new constraint into account, simply replace (5) and (6) by

$$\forall i, \sum_{j \in n(i)} s_{ij} + \sum_{j \in n(i)} r_{ij} \leq 1. \tag{13}$$

Reduction for $\mathcal{M}(r, s, w)$, the No internal parallelism model. In this model, a processor can only do one thing at a time: receive, send, or compute tasks. This time, we have to replace (1), (5), and (6) by

$$\forall i, \sum_{j \in n(i)} s_{ij} + \alpha_i + \sum_{j \in n(i)} r_{ij} \leq 1. \tag{14}$$

Strongly heterogeneous platforms. Finally, it is important to point out that the processor nodes may operate under different models. Instead of writing the same equations for each node, we pick up different equations for each node, those corresponding to the desired operation models.

5 Complexity Results

5.1 Asymptotic Optimality

In this section, we prove that steady state scheduling is asymptotically optimal. Given a platform graph $G = (V, E, w, c)$ and a time bound K, define $opt(G, K)$ as the optimal number of tasks that can be computed using the whole platform, within K time-units. We have the following result.

Lemma 1. $opt(G, K) \leq n_{\text{tasks}}(G) \times K$.

Proof. Consider an optimal scheduling. For each processor P_i, let $t_i(K)$ be the total number of tasks that have been executed by P_i within the K time-units. Similarly, for each edge e_{ij} in the graph, let $t_{i,j}(K)$ be the total number of tasks that have been forwarded by P_i to P_j within the K time-units. The following equations hold true.

- $t_i(K) \cdot w_i \leq K$ (time for P_i to process its tasks).
- $\sum_{j \in n(i)} t_{i,j}(K) \cdot c_{ij} \leq K$ (time for P_i to forward outgoing tasks in the one-port model).
- $\sum_{j \in n(i)} t_{j,i}(K). \cdot c_{ij} \leq K$ (time for P_i to receive incoming tasks in the one-port model).
- $\sum_{j \in n(i)} t_{j,i}(K) = t_i(K) + \sum_{j \in n(i)} t_{i,j}(K)$ (conservation equation).

Let $\alpha_i = \frac{w_i t_i(K)}{K}$, $s_{ij} = \frac{c_{ij} t_{i,j}(K)}{K}$, and $r_{ij} = \frac{c_{ij} t_{j,i}(K)}{K} = s_{ji}$. All the equations of the linear program MSSG(G) hold, hence, $\sum_i \frac{\alpha_i}{w_i} \leq n_{\text{tasks}}(G)$, the optimal value. Going back to the original variables, we derive:

$$opt(G, K) = \sum_i t_i(K) \leq n_{\text{tasks}}(G) \times K.$$

□

Basically, Lemma 1 says that no scheduling can execute more tasks than the steady state. There remains to bound the initialization and the clean-up phase to come up with a well-defined scheduling algorithm based upon steady state operation. Consider the following algorithm (assume K is large enough):

- Solve the linear program MSSG(G): Compute the maximal throughput $n_{\text{tasks}}(G)$, compute all the values α_i, r_{ij}, and r_{ij}, and determine the time-period T. For each processor P_i, determine per_i, the total number of tasks that it receives per period. Note that all these quantities are independent of K: They depend only upon the characteristics w_i and c_{ij} of the platform graph.
- Initialization: The master sends per_i tasks to each processor P_i. This requires I units of time, where I is a constant independent of K.
- Let J be the maximum time for each processor to consume per_i tasks ($J = \max_i\{per_i.w_i\}$). Again, J is a constant independent of K.
- Let $r = \lfloor \frac{K-I-J}{T} \rfloor$.
- Steady state scheduling: During r periods of time T, operate the platform in steady state, according to the solution of MSSG(G).

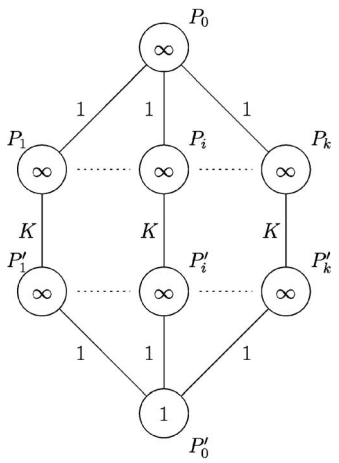


Fig. 9. Graph G used for the proof of the inapproximability.

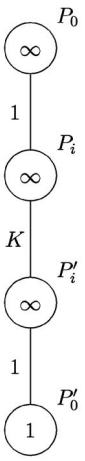


Fig. 10. Description of a tree T extracted from G.

- Clean-up: Do not forward any task, but consume in place all remaining tasks. This requires at most J time-units. Do nothing during the very last units ($K - I - J$ may not be evenly divisible by T).
- The number of tasks processed by this algorithm within K time-units is equal to $steady(G, K) = (r + 1) \times T \times n_{\text{tasks}}(G)$.

Proposition 3. *The previous scheduling algorithm based upon steady state operation is asymptotically optimal:*

$$\lim_{K \to +\infty} \frac{steady(G, K)}{opt(G, K)} = 1.$$

Proof. Using Lemma 1, $opt(G, K) \leq n_{\text{tasks}}(G).K$. From the description of the algorithm, we have $steady(G, K) = ((r + 1)T).n_{\text{tasks}}(G) \geq (K - I - J).n_{\text{tasks}}(G)$, whence the result because I, J, T, and $n_{\text{tasks}}(G)$ are constants independent of K. □

5.2 Spanning Trees

For a general interconnection graph, the solution of the linear programming problem may lead to the use of multiple paths (this is the case for the toy example of Section 2.4). As already mentioned, it may be of interest to extract the best spanning tree (the one with maximum throughput) out of the graph. Using a tree greatly simplifies the implementation (because of the unique route from the master to any processor). Also, the bandwidth-centric algorithm presented in [4] is local and demand-driven, therefore, is very robust to small variations in resources capabilities.

This section provides "negative" results: First, extracting the best tree is NP-hard. But, even if we are ready to pay a high (probably exponential) cost to determine the best tree, there exist graphs for which the throughput of the best tree is arbitrarily bad compared to the throughput that can be achieved while the whole graph.

5.2.1 Finding the Best Spanning Tree

Our aim is to find the spanning tree that maximizes the throughput, i.e., the number of tasks that can be processed within one unit of time at steady state. Formally, we can state the problem as follows:

Defintion 1 (BEST-TREE(G)). *Let $G = (V, E, w, c)$ be the node-weighted edge-weighted graph representing the architectural framework. Find the tree $T = (V, E', w, c)$ that is a subgraph of G rooted at the master, such that the number of tasks $n_{\text{task}}(T)$ that can be processed in steady state within one time-unit, using only those edges of the tree, is maximized.*

The associated decision problem is the following:

Definition 2 (BEST-TREE-DEC(G, α)). *Let $G = (V, E, w, c)$ be the node-weighted edge-weighted graph representing the architectural framework. Is there a tree $T = (V, E', w, c)$ that is a subgraph of G rooted at the master, such that $n_{task}(T) \geq \alpha$?*

Theorem 2. BEST-TREE-DEC*(G, α) is NP-Complete.*

Proof. The problem BEST-TREE-DEC(G, α) is proven to be NP-complete in [3] by reduction from 2-PARTITION, which is known to be NP-Complete [9]. □

5.2.2 Inapproximability of a Graph by a Tree

One natural and interesting question is the following: How bad may the approximation of a graph by a tree be? The following theorem states the inapproximability of a general graph by a tree with respect to throughput.

Theorem 3. *Given any positive integer K, there exists a graph G such that, for any tree T that is a subgraph of G and rooted at the master, we have*

$$\frac{n_{task}(G)}{n_{task}(T)} \geq K.$$

Proof. Consider the graph depicted in Fig. 9.

One can easily check that, using all the communication resources, it is possible to process one task within each time unit, i.e., $n_{task}(G) = 1$. However, any tree T extracted from G is equivalent to the chain depicted in Fig. 10 since P'_0 is the only computing resource. Moreover, because of the slow link between P_i and P'_i, the number of tasks that

can be processed within one unit of time is bounded by $\frac{1}{K}$ and, thus,

$$\forall T, \quad \frac{n_{task}(G)}{n_{task}(T)} \geq K.$$

□

6 Related Problems

We classify several related papers along the following lines:

Scheduling task graphs on heterogeneous platforms. Several heuristics have been introduced to schedule (acyclic) task graphs on different-speed processors, see [21], among others. Unfortunately, all these heuristics assume no restriction on the communication resources, which renders them somewhat unrealistic to model real-life applications. Recent papers [12], [19] suggest taking communication contention into account.

Scheduling divisible load. Instead of scheduling several communications, one for each task, the divisible load approach consists in scheduling a single communication at the beginning of the operation. The cost of this communication is proportional to the amount of computation performed. A star graph is targeted in [20], with homogeneous links and different-speed processors. The extension to heterogeneous links is dealt with in [6]. See also [14], [1] for more results on sharing bag of tasks in heterogeneous clusters.

Master-slave on the computational grid. Master-slave scheduling on the grid can be based on a network-flow approach [17] or on an adaptive strategy [11]. Note that the network-flow approach of [17] is possible only when using a full multiple-port model, where the number of simultaneous communications for a given node is not bounded. Enabling frameworks to facilitate the implementation of master-slave tasking are described in [10], [22].

7 Conclusion

In this paper, we have dealt with master-slave tasking on a heterogeneous platform. We have shown how to determine the best steady state scheduling strategy for a general interconnection graph, using a linear programming approach. We derive from this steady-state strategy a asymptotically optimal schedule. We have also given a closed-form expression and a more efficient algorithm (linear in the processor number) for tree-shaped platform graphs using a *bandwidth-centric* approach. Some simulation results on demand-driven heuristics based on the *bandwidth-centric* approach are given in [4] and [13].

We have also derived negative theoretical results, namely, that general interconnection graphs may be arbitrarily more powerful than spanning trees, and that determining the best spanning tree is NP-hard. Nevertheless, several low-cost heuristics that achieve very good performances on a wide range of simulations are proposed in [2]. These positive experiments show that, in practice, it is safe to rely on spanning trees to implement master-slave tasking.

This work can be extended in the following two directions:

- On the theoretical side, we could try to solve the problem of maximizing the number of tasks that can be executed within T time-steps, where T is a given time-bound. This scheduling problem is more complicated than the search for the best steady state. Taking the initialization phase into account renders the problem quite challenging.
- On the practical side, we need to run actual experiments rather than simulations. Indeed, it would be interesting to capture actual architecture and application parameters, and to compare heuristics on real-life problems.

Acknowledgments

The authors would like to thank the reviewers for their numerous comments and suggestions, which greatly improved the final version of the paper.

References

[1] M. Adler, Y. Gong, and A.L. Rosenberg, "Optimal Sharing of Bags of Tasks in Heterogeneous Clusters," *Proc. 15th ACM Symp. Parallelism in Algorithms and Architectures,* pp. 1-10, 2003.

[2] C. Banino, O. Beaumont, A. Legrand, and Y. Robert, "Scheduling Strategies for Master-Slave Tasking on Heterogeneous Processor Grids," *Proc. Int'l Conf. Applied Parallel Computing,* pp. 423-432, 2002.

[3] C. Banino, O. Beaumont, A. Legrand, and Y. Robert, "Scheduling Strategies for Master-Slave Tasking on Heterogeneous Processor Grids," Technical Report 2002-12, LIP, Mar. 2002.

[4] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert, "Bandwidth-Centric Allocation of Independent Tasks on Heterogeneous Platforms," *Proc. Int'l Parallel and Distributed Processing Symp.,* 2002.

[5] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert, "Optimal Algorithms for the Pipelined Scheduling of Task Graphs on Heterogeneous Systems," Technical Report RR-2003-29, LIP, ENS Lyon, France, May 2003.

[6] S. Charcranoon, T.G. Robertazzi, and S. Luryi, "Optimizing Computing Costs Using Divisible Load Analysis," *IEEE Trans. Computers,* vol. 49, no. 9, pp. 987-991, Sept. 2000.

[7] *Scheduling Theory and Its Applications.* P. Chrétienne, E.G. Coffman Jr., J.K. Lenstra, and Z. Liu, eds. John Wiley and Sons, 1995.

[8] Entropia, http://www.entropia.com, 2003.

[9] M.R. Garey and D.S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1991.

[10] J.P Goux, S. Kulkarni, J. Linderoth, and M. Yoder, "An Enabling Framework for Master-Worker Applications on the Computational Grid," *Proc. Ninth IEEE Int'l Symp. High Performance Distributed Computing,* 2000.

[11] E. Heymann, M.A. Senar, E. Luque, and M. Livny, "Adaptive Scheduling for Master-Worker Applications on the Computational Grid," R. Buyya and M. Baker, eds., *Proc. Workshop Grid Computing,* pp. 214-227, 2000.

[12] T.S. Hsu, J.C. Lee, D.R. Lopez, and W.A. Royce, "Task Allocation on a Network of Processors," *IEEE Trans. Computers,* vol. 49, no. 12, pp. 1339-1353, 2000.

[13] B. Kreaseck, L. Carter, H. Casanova, and J. Ferrante, "Autonomous Protocols for Bandwidth-Centric Scheduling of Independent Task Applications," *Proc. Int'l Parallel and Distributed Processing Symp.,* 2003.

[14] A.L. Rosenberg, "On Sharing Bags of Tasks in Heterogeneous Networks of Workstations: Greedier is Not Better," *Proc. Int'l Conf. Cluster Computing,* pp. 124-131, 2001.

[15] A. Schrijver, "Combinatorial Optimization: Polyhedra and Efficiency," *Algorithms and Combinatorics,* vol. 24, Springer-Verlag, 2003.

[16] SETI, http://setiathome.ssl.berkeley.edu, 2003.

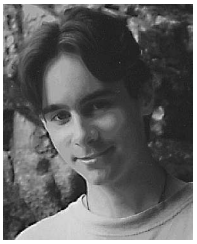
[17] G. Shao, F. Berman, and R. Wolski, "Master/Slave Computing on the Grid," *Proc. Heterogeneous Computing Workshop,* 2000.
[18] B.A. Shirazi, A.R. Hurson, and K.M. Kavi, *Scheduling and Load Balancing in Parallel and Distributed Systems.* IEEE Computer Science Press, 1995.
[19] O. Sinnen and L. Sousa, "Comparison of Contention-Aware List Scheduling Heuristics for Cluster Computing," *Proc. Workshop Scheduling and Resource Management for Cluster Computing,* pp. 382-387, 2001.
[20] J. Sohn, T.G. Robertazzi, and S. Luryi, "Optimizing Computing Costs Using Divisible Load Analysis," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 3, pp. 225-234, Mar. 1998.
[21] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Task Scheduling Algorithms for Heterogeneous Processors," *Proc. Eighth Heterogeneous Computing Workshop,* 1999.
[22] J.B. Weissman, "Scheduling Multi-Component Applications in Heterogeneous Wide-Area Networks," *Proc. Heterogeneous Computing Workshop,* 2000.

Cyril Banino is currently a PhD student in NTNU, the Norvegian University of Science and Technology. He is mainly interested in high-performance scientific computing and in combinatorial optimization.

Olivier Beaumont received the PhD degree from the Université de Rennes in January 1999. He is currently an associate professor in the LaBRI laboratory in Bordeaux. His main research interests are parallel algorithms on distributed memory architectures.

Larry Carter received the AB degree from Dartmouth College in 1969 and the PhD degree in mathematics from the University of California at Berkeley in 1974. He worked at as a research staff member and manager at IBM's T.J. Watson Research Center for nearly 20 years in the areas of probabilistic algorithms, compilers, VLSI testing, and high-performance computation. Since 1994, Dr. Carter has been a professor in the Computer Science and Engineering Department of the University of California at San Diego. Between 1996 and 2000, he served as vice chair and then chair of the department. His current research interests include scientific computation, performance programming, parallel computation, and computer architecture. Prof. Carter is a senior fellow at the San Diego Supercomputing Center and a fellow of the IEEE.

Jeanne Ferrante received the PhD degree in mathematics from MIT in 1974. She was a research staff member at the IBM T.J. Watson Research Center from 1978 to 1994, and currently is a professor of computer science and associate dean of the Jacobs School of Engineering at the University of California, San Diego. Her work has included the development of intermediate representations for optimizing and parallelizing compilers, most notably the Program Dependence Graph and Static Single Assignment form. Her interests also include optimizing for parallelism and memory hierarchy and scheduling on large distributed platforms. She is a fellow of the ACM and a senior member of the IEEE.



Arnaud Legrand is currently a PhD student in the Computer Science Laboratory LIP at ENS Lyon. He is mainly interested in parallel algorithm design for heterogeneous platforms and in scheduling techniques.

Yves Robert received the PhD degree from Institut National Polytechnique de Grenoble in January 1986. He is currently a full professor in the Computer Science Laboratory LIP at ENS Lyon. He is the author of four books, 80 papers published in international journals, and 100 papers published in international conferences. His main research interests are scheduling techniques and parallel algorithms for clusters and grids. He is a member of ACM and IEEE, and serves as an associate editor of *IEEE Transactions on Parallel and Distibuted Systems*.

Scheduling Divisible Loads on Star and Tree Networks: Results and Open Problems

Olivier Beaumont, Henri Casanova, *Member*, *IEEE*, Arnaud Legrand, Yves Robert, *Senior Member*, *IEEE*, and Yang Yang

Abstract—Many applications in scientific and engineering domains are structured as large numbers of independent tasks with low granularity. These applications are thus amenable to straightforward parallelization, typically in master-worker fashion, provided that efficient scheduling strategies are available. Such applications have been called *divisible loads* because a scheduler may *divide* the computation among worker processes arbitrarily, both in terms of number of tasks and of task sizes. Divisible load scheduling has been an active area of research for the last 15 years. A vast literature offers results and scheduling algorithms for various models of the underlying distributed computing platform. Broad surveys are available that report on accomplishments in the field. By contrast, in this paper, we propose a unified theoretical perspective that synthesizes previously published results, several novel results, and open questions, in a view to foster novel divisible load scheduling research. Specifically, we discuss both one-round and multiround algorithms, and we restrict our scope to the popular star and tree network topologies, which we study with both linear and affine cost models for communication and computation.

Index Terms—Parallel computing, scheduling, divisible load.

1 Introduction

Scheduling the tasks of a parallel application on the resources of a distributed computing platform efficiently is critical for achieving high performance. The scheduling problem has been studied for a variety of application models, such as the well-known directed acyclic task graph model for which many scheduling heuristics have been developed [1]. Another popular application model is that of independent tasks with no task synchronizations and no intertask communications. Applications conforming to this admittedly simple model arise in most fields of science and engineering. A possible model for independent tasks is one for which the number of tasks and the task sizes, i.e., their computational costs, are set in advance. In this case, the scheduling problem is akin to bin-packing and a number of heuristics have been proposed in the literature. Another flavor of the independent tasks model is one in which the number of tasks and the task sizes can be chosen arbitrarily. This corresponds to the case when the application consists of an amount of computation, or *load*, that can be divided into any number of independent pieces. This corresponds to a perfectly parallel job: Any subtask can itself be processed in parallel, and on any number of workers. In practice, this model is an approximation of an application that consists of large numbers of identical, low-granularity computations. This *divisible load* model has been widely studied in the last several years, and *Divisible Load Theory* (DLT) has been popularized by the landmark book written in 1996 by Bharadwaj et al. [2].

DLT provides a practical framework for the mapping on independent tasks onto heterogeneous platforms, and has been applied to a large spectrum of scientific problems, including linear algebra [3], image processing [4], [5], video and multimedia broadcasting [6], [7], database searching [8], [9], and the processing of large distributed files [10]. These applications are amenable to the simple master-worker programming model and can thus be easily implemented and deployed on computing platforms ranging from small commodity clusters to computational grids [11]. From a theoretical standpoint, the success of the divisible load model is mostly due to its analytical tractability. Optimal algorithms and closed-form formulas exist for the simplest instances of the divisible load problem. This is in sharp contrast with the theory of task graph scheduling, which abounds in NP-completeness theorems and in inapproximability results.

There exists a vast literature on DLT. In addition to the landmark book [2], two introductory surveys have been published recently [12], [13]. Furthermore, a special issue of the *Cluster Computing* journal is entirely devoted to divisible load scheduling [14], and a Web page collecting DLT-related papers is maintained [15]. Consequently, the goal of this paper is not to present yet another survey of DLT theory and its various applications. Instead, we focus on relevant theoretical aspects: We aim at synthesizing some important results for realistic platform models. We give a new presentation of several previously published results, and we add a number of new contributions. The material in this paper provides the level of detail and, more importantly, the

- *O. Beaumont is with LaBRI, Domaine Universitaire, 351, cours de la Libération, F-33405 Talence, Cedex, France. E-mail Olivier.Beaumont@labri.fr.*
- *H. Casanova and Y. Yang are with the Department of Computer Science and Engineering, and the San Diego Supercomputer Center, 9500 Gilman Drive, Mail Stop 0114, University of California, San Diego, La Jolla, CA 92093-0114. E-mail: {casanova, yangyang}@cs.ucsd.edu.*
- *A. Legrand and Y. Robert are with LIP, UMR CNRS-ENS Lyon-INRIA 5668, Ecole Normale Supérieure de Lyon, F-69364 Lyon Cedex 07, France. E-mail: {Yves.Robert, Arnaud.Legrand}@ens-lyon.fr.*

Manuscript received 4 Sept. 2003; revised 7 Mar. 2004; accepted 14 July 2004. For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0153-0903.

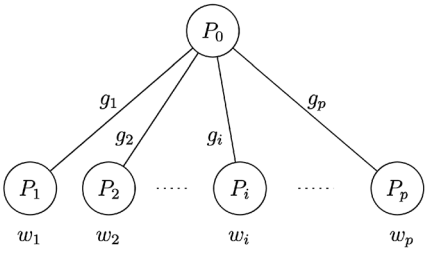


Fig. 1. Heterogeneous star graph with the linear cost model.

unifying perspective that are necessary for fostering new research in the field.

In this paper, we limit our discussion to star-shaped and tree-shaped logical network topologies because they often represent the solution of choice to implement master-worker computations. Note that the star network encompasses the case of a bus, which is a homogeneous star network. We consider two types of model, for communication and computation: linear or affine in the data size. Our major results include several optimality results in the case where each processor receives a single message (one-round strategies). We also provide new analytical formulations and performance characterizations in the more general case where each processor receives its load divided into several submessages (multiround strategies).

The extended version of this paper [16] reviews works that study other network topologies. It also surveys constant-cost models. Numerous extensions to the original DLT framework have been proposed recently. Release times and buffer-size capacities are taken into account in [17], multiple applications competing for resources are dealt with in [18], and extensions to grid platforms are considered in [19], [20]. In contrast, we limit ourselves to the basic case of a single load to be distributed on a star-shaped or tree-shaped platform.

The rest of this paper is organized as follows: In Section 2, we detail our platform and cost models. We also introduce the algorithmic techniques that have been proposed to schedule divisible loads: one-round and multiround algorithms. One-round algorithms are described in detail in Section 3 and multiround algorithms in Section 4. Finally, we conclude in Section 5.

2 Framework

2.1 Target Architectures and Cost Models

We consider either star-graphs or tree-graphs, and either linear or affine costs, which leads to four different platform combinations.

As illustrated in Fig. 1, a *star network* $\mathcal{S} = \{P_0, P_1, P_2, \ldots, P_p\}$ is composed of a master P_0 and of p workers $P_i, 1 \leq i \leq p$. There is a communication link from the master P_0 to each worker P_q. In the linear cost model, each worker P_q has a (relative) computing power w_q: It takes $X.w_q$ time units to execute X units of load on worker P_q. Similarly, it takes $X.g_q$ time unites to send X units of load from P_0 to P_q. Without loss of generality, we assume that the master has no processing capability (otherwise, add a fictitious extra worker paying no communication cost to simulate computation at the master).

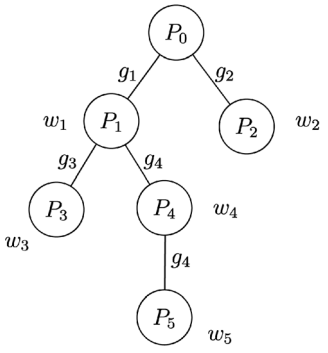


Fig. 2. Heterogeneous tree graph.

In the affine cost model, a latency is added to computation and communication costs: It takes $W_q + X.w_q$ time units to execute X units of load on worker P_q, and $G_q + X.g_q$ time units to send X units of load from P_0 to P_q. It is acknowledged that these latencies make the model more realistic.

For communications, the one-port model is used: The master can only communicate with a single worker at a given time-step. We assume that communications can overlap computations on the workers: A worker can compute a load fraction while receiving the data necessary for the execution of the next load fraction. This corresponds to workers *equipped with a front end* as in [2]. A *bus network* is a star network such that all communication links have the same characteristics: $g_i = g$ and $G_i = G$ for each worker P_i, $1 \leq i \leq p$.

Essentially, the same one-port model, with overlap of communication with computation, is used for tree-graph networks. A tree-graph $\mathcal{T} = \{P_0, P_1, P_2, \ldots, P_p\}$ (see Fig. 2) simply is an arborescence rooted at the master P_0. We still call the other resources *workers*, even though nonleaf workers have other workers (their children in the tree) to which they can delegate work. In this model, it is assumed that a worker in the tree can simultaneously perform some computation, receive data from its parent, and communicate to at most one of its children (sending previously received data).

2.2 Algorithmic Strategies: One-Round versus Multiround

We denote by W_{total} the total load to be executed. The key assumption of DLT is that this load is perfectly divisible into an arbitrary number of pieces, or *chunks*. The master can distribute the chunks to the workers in a single *round* (also called "installment" in [2]), so that there is a single communication between the master and each worker. The problem is to determine the size of these chunks and the order in which they are sent to the workers.

We review one-round algorithms in Section 3. For large loads, the single round approach is not efficient due to the idle time incurred by the last workers to receive chunks. To reduce the makespan, i.e., the total execution time, the master can send chunks to the workers in multiple rounds so that communication is pipelined and overlapped with computation. Additional questions in this case are: "How many rounds should be scheduled?" and "What are the best chunk sizes at each round?" We discuss multiround algorithms in Section 4.

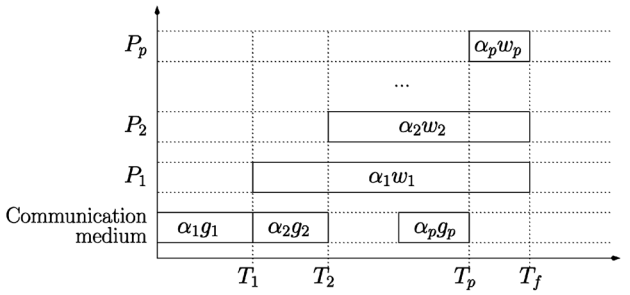


Fig. 3. Pattern of a solution for dispatching a divisible load, using a star network and the linear cost model. All workers complete execution at the same time-step T_f.

In fact, we point out that there is no formal definition of a *round*, which intuitively is a sequence of communications from the master to different workers. The situation is clear for one-round algorithms, formally defined as computations where each participating worker receives a single message. This is not so clear for multiround algorithms, which we define by contraposition: There is some worker that receives at least two messages. However, from the *multiround* denomination, we expect some periodicity, i.e., some regular pattern reproduced repeatedly, in the sequence of communications emitted by the master. All published multiround algorithms exhibit such a regular pattern.

3 One-Round Algorithms

For one-round algorithms, the first problem is to determine in which order the chunks should be sent to the different workers (or, equivalently, to sort the workers), given that the master can perform only one communication at a time. Once the communication order has been determined, the second problem is to decide how much work should be allocated to each worker P_i: Each P_i receives α_i units of load, where $\sum_{i=1}^{p} \alpha_i = W_{\text{total}}$. The final objective is to minimize the makespan, i.e., the total execution time.

3.1 Star Network and Linear Cost Model

This is the simplest platform combination, denoted as STARLINEAR. Let α_i denote the number of units of load sent to worker P_i, such that $\sum_{i=1}^{p} \alpha_i = W_{\text{total}}$. Fig. 3 depicts the execution, where T_i denotes idle time of P_i, i.e., the time elapsed before P_i begins its processing. The goal is to minimize the total execution time, $T_f = \max_{1 \le i \le p}(T_i + \alpha_i w_i)$, according to the linear model defined in Section 2. In Fig. 3, all the workers participate in the computation, and they all finish computing at the same time (i.e., $T_i + \alpha_i w_i = T_f,\ \forall i$). This is a general result.

Proposition 1. *In any optimal solution of the STARLINEAR problem, all workers participate in the computation, and they all finish computing simultaneously.*

Note that Proposition 1 has been proven for the case of a bus in [2]. To the best of our knowledge, this is a new result for the case of a heterogeneous star network. Furthermore, in the case of a bus, any load distribution order is optimal [2], but this property does not extend to the heterogeneous case (for which the optimal ordering will be characterized in Proposition 2 below).

Proof. We first prove that in an optimal solution all workers participate to the computation. Then, we prove that in any optimal solution, all workers finish computing simultaneously. □

Lemma 1. *In any optimal solution, all workers participate in the computation.*

Proof. Consider an optimal solution: up to a renumbering of the processors, assume that the ordering is $P_1, P_2, \ldots, P_p$. Suppose that at least one worker is kept fully idle. In this case, at least one of the α_i, $1 \le i \le p$, is zero. Let us denote by k the largest index such that $\alpha_k = 0$.

Case $k < p$. Consider the solution of STARLINEAR, where we add P_k at the end of the initial solution (hence, we use the ordering $P_1, \ldots, P_{k-1}, P_{k+1}, \ldots, P_p, P_k$). By construction, $\alpha_p \neq 0$, so that the communication medium is not used during at least the last $\alpha_p w_p$ time units. Therefore, it would be possible to process at least $\frac{\alpha_p w_p}{g_k + w_k} > 0$ additional units of load with worker P_k, which contradicts the assumption that the original solution was optimal: P_k does more work than in the original solution where it was kept idle, and all the other processors execute the same amount of load units.

Case $k = p$. We modify the optimal solution of STARLINEAR by giving some work to P_p without increasing the execution time. Let k' be the largest index such that $\alpha_{k'} \neq 0$. By construction, the communication medium is not used during at least the last $\alpha_{k'} w_{k'} > 0$ time units. Thus, as previously, it would be possible to process at least $\frac{\alpha_{k'} w_{k'}}{g_p + w_p} > 0$ additional units of load with worker P_p, which leads to a similar contradiction.

Therefore, in any optimal solution, all workers participate in the computation. □

It is worth pointing out that the above property does not hold true if we consider solutions in which the communication ordering is fixed a priori. For instance, consider a platform comprising two workers: P_1 (with $g_1 = 4$ and $w_1 = 1$) and P_2 (with $g_2 = 1$ and $w_2 = 1$). If the first chunk has to be sent to P_1 and the second chunk to P_2, the optimal number of units of load that can be processed within 10 time units is 5, and P_1 is kept fully idle in this solution. On the other hand, if the communication ordering is not fixed, then six units of load can be performed within 10 time units (five units of load are sent to P_2, and then 1 to P_1). In the optimal solution, both workers perform some computation, and both workers finish computing at the same time, which is stated in the following lemma.

Lemma 2. *In the optimal schedule, all workers finish computing simultaneously.*

Proof. Consider an optimal solution. All the α_is have strictly positive values (Lemma 1). Consider the following linear program:

$$\text{Maximize } \sum \beta_i,$$
$$\text{subject to}$$
$$\begin{cases} \text{LB}(i) & \forall i, \quad \beta_i \ge 0 \\ \text{UB}(i) & \forall i, \quad \sum_{k=1}^{i} \beta_k g_k + \beta_i w_i \le T. \end{cases}$$

The α_is satisfy the set of constraints above, and from any set of β_is satisfying the set of inequalities, we can build a

valid solution of the STARLINEAR problem that process exactly $\sum \beta_i$ units of load. Therefore, if we denote by $(\beta_1, \ldots, \beta_p)$ an optimal solution of the linear program, then $\sum \beta_i = \sum \alpha_i$.

It is known that one of the extremal solutions $\mathcal{S}_1$ of the linear program is one of the convex polyhedron $\mathcal{P}$ induced by the inequalities [21, chapter 11]. This means that in the solution $\mathcal{S}_1$, there are at least p inequalities among the $2p$ equalities. Since we know that for any optimal solution of the STARLINEAR problem, all the β_is are strictly positive (Lemma 1), then this vertex is the solution of the following (full rank) linear system

$$\forall i, \quad \sum_{k=1}^{i} \beta_k g_k + \beta_i w_i = T.$$

Thus, we derive that there is an optimal solution where all workers finish their work at the same time.

Let us denote by $\mathcal{S}_2 = (\alpha_1, \ldots, \alpha_p)$ another optimal solution, with $\mathcal{S}_1 \neq \mathcal{S}_2$. As already pointed out, $\mathcal{S}_2$ belongs to the polyhedron $\mathcal{P}$. Now, consider the following function f:

$$f : \begin{cases} \mathbb{R} & \rightarrow & \mathbb{R}^n \\ x & \mapsto & \mathcal{S}_1 + x(\mathcal{S}_2 - \mathcal{S}_1). \end{cases}$$

By construction, we know that $\sum \beta_i = \sum \alpha_i$. Thus, with the notation $f(x) = (\gamma_1(x), \ldots, \gamma_p(x))$:

$$\forall i, \gamma_i(x) = \beta_i + x(\alpha_i - \beta_i)$$

and, therefore,

$$\forall x, \ \sum \gamma_i(x) = \sum \beta_i = \sum \alpha_i.$$

Therefore, all the points $f(x)$ that belong to $\mathcal{P}$ are extremal solutions of the linear program.

Since $\mathcal{P}$ is a convex polyhedron and both $\mathcal{S}_1$ and $\mathcal{S}_2$ belong to $\mathcal{P}$, then $\forall 0 \leq x \leq 1, \ f(x) \in \mathcal{P}$. Let us denote by x_0 the largest value of $x \geq 1$ such that $f(x)$ still belongs to $\mathcal{P}$: At least one constraint of the linear program is an equality in $f(x_0)$, and this constraint is not satisfied for $x > x_0$. Could this constraint be one of the UB(i)s? The answer is no because otherwise this constraint would be an equality along the whole line $(\mathcal{S}_2 f(x_0))$, and would remain an equality for $x > x_0$. Hence, the constraint of interest is one of the LB(i)s. In other terms, there exists an index i such that $\gamma_i(x_0) = 0$. This is a contradiction since we have proven that the γ_is correspond to an optimal solution of the STARLINEAR problem. Therefore, $\mathcal{S}_1 = \mathcal{S}_2$, the optimal solution is unique, and in this solution, all workers finish computing simultaneously.

Altogether, this concludes the proof of Proposition 1.□

To be able to characterize the optimal solution, there remains to determine the best ordering for the master P_0 to send work to the workers.

Proposition 2. *An optimal ordering for the STARLINEAR problem is obtained by serving the workers in the ordering of non decreasing link capacities g_i.*

We give a new, shorter proof of Proposition 2, which originally appeared in [2, chapter 7], but under the hypothesis that all workers participate and finish computing simultaneously. Note that we have shown that this latter property indeed holds for any optimal schedule.

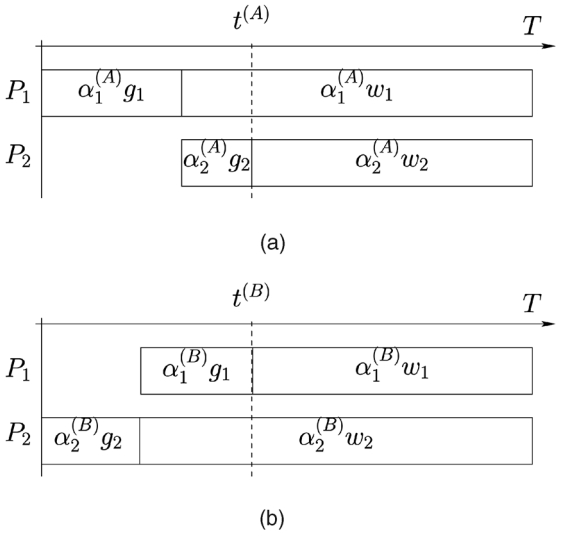


Fig. 4. Comparison of the two possible orderings. (a) P_1 starts before P_2. (b) P_2 starts before P_1.

Proof. The proof is based on the comparison of the amount of work that is performed by the first two workers, and then proceeds by induction. To simplify notations, assume that P_1 and P_2 have been selected as the first two workers. There are two possible orderings, as illustrated in Fig. 4. For each ordering, we determine the total number of units of load $\alpha_1 + \alpha_2$ that are processed in T time-units, and the total occupation time, t, of the communication medium during this time interval. We denote with upper-script (A) (respectively, (B)) all the quantities related to the first (respectively, second) ordering.

Let us first determine the different quantities $\alpha_1^{(A)}$, $\alpha_2^{(A)}$, and $t^{(A)}$ for the upper ordering in Fig. 4:

- From the equality $\alpha_1^{(A)}(g_1 + w_1) = T$, we get:

$$\alpha_1^{(A)} = \frac{T}{g_1 + w_1}. \tag{1}$$

- Using the equality $\alpha_1^{(A)} g_1 + \alpha_2^{(A)}(g_2 + w_2) = T$, we obtain (from (1)):

$$\alpha_2^{(A)} = \frac{T}{g_2 + w_2} - \frac{T g_1}{(g_1 + w_1)(g_2 + w_2)}. \tag{2}$$

Therefore, the overall number of processed units of load is equal to (by (1) and (2)):

$$\alpha_1^{(A)} + \alpha_2^{(A)} = \frac{T}{g_1 + w_1} + \frac{T}{g_2 + w_2} - \frac{T g_1}{(g_1 + w_1)(g_2 + w_2)} \tag{3}$$

and the overall occupation time of the network medium is equal to (using the previous equalities and $t^{(A)} = \alpha_1^{(A)} g_1 + \alpha_2^{(A)} g_2$):

$$t^{(A)} = \frac{Tg_1}{g_1+w_1} + \frac{Tg_2}{g_2+w_2} - \frac{Tg_1g_2}{(g_1+w_1)(g_2+w_2)}. \quad (4)$$

A similar expression can be obtained for scenario (B) and we derive that:

$$(\alpha_1^{(A)} + \alpha_2^{(A)}) - (\alpha_1^{(B)} + \alpha_2^{(B)}) = \frac{T(g_2-g_1)}{(g_1+w_1)(g_2+w_2)}, \quad (5)$$

and

$$t^{(A)} = t^{(B)}. \quad (6)$$

Thanks to these expressions, we know that the occupation of the communication medium does not depend on the communication ordering. Therefore, we only need to consider the number of processed units of load in both situations. Equation (5) indicates that one should send chunks to the worker with the smallest g_i first: If $g_2 > g_1$, more work is done in time T with the ordering (A) than with (B).

We now proceed to the general case. Suppose that the workers are already sorted so that $g_1 \leq g_2 \leq \ldots \leq g_p$. Consider an optimal ordering of the communications σ, where chunks are sent successively to $P_{\sigma(1)}, P_{\sigma(2)}, \ldots P_{\sigma(p)}$. Let us assume that there exists an index i such that $\sigma(i) > \sigma(i+1)$. Furthermore, let us consider the smallest such index if multiple ones exist. Consider now the following ordering:

$$P_{\sigma(1)}, \ldots, P_{\sigma(i-1)}, P_{\sigma(i+1)}, P_{\sigma(i)}, P_{\sigma(i+2)}, \ldots P_{\sigma(p)}.$$

Then, $P_{\sigma(1)}, \ldots, P_{\sigma(i-1)}, P_{\sigma(i+2)}, \ldots P_{\sigma(p)}$ perform exactly the same number of units of load, since the exchange does not affect the overall communication time, but together, $P_{\sigma(i+1)}$ and $P_{\sigma(i)}$ perform $\frac{T(g_{\sigma(i)}-g_{\sigma(i+1)})}{(g_{\sigma(i+1)}+w_{\sigma(i+1)})(g_{\sigma(i)}+w_{\sigma(i)})}$ more units of load, where T denotes the remaining time after communications to $P_{\sigma(1)}, \ldots, P_{\sigma(i-1)}$. Therefore, the initial ordering σ is not optimal, which is a contradiction. Therefore, index i does not exist, which proves that, in an optimal ordering, the workers are sorted by nondecreasing values of the g_is. □

According to Proposition 2, we now reorder the workers so that $g_1 \leq g_2 \leq \ldots \leq g_p$. The following linear program aims at computing the optimal distribution of the load:

$$\begin{array}{l}\text{Minimize } T_f,\\ \text{subject to}\\ \begin{cases}(1)\ \alpha_i \geq 0 & 1 \leq i \leq p\\ (2)\ \sum_{i=1}^{p} \alpha_i = W_{\text{total}} & \\ (3)\ \alpha_1 g_1 + \alpha_1 w_1 \leq T_f & \text{(first communication)}\\ (4)\ \sum_{j=1}^{i} \alpha_j g_j + \alpha_i w_i \leq T_f & (i\text{th communication}).\end{cases}\end{array}$$

Theorem 1. *The optimal solution for the STARLINEAR problem is given by the solution of the linear program above.*

Proof. Direct consequence of Propositions 1 and 2. Note that inequalities (3) and (4) will be in fact equalities in the solution of the linear program, so that we can easily derive a closed-form expression for T_f. □

We point out that this is linear programming with rational numbers, hence of polynomial complexity. Finally, we consider the variant where the master is capable of processing chunks (with computing power w_0) while communicating to one of its children. It is easy to see that the master is kept busy at all times (otherwise, more units of load could be processed). The optimal solution is therefore given by the following linear program (where $g_1 \leq g_2 \leq \ldots \leq g_p$ as before):

$$\begin{array}{l}\text{Minimize } T_f,\\ \text{subject to}\\ \begin{cases}(1)\ \alpha_i \geq 0 & 0 \leq i \leq p\\ (2)\ \sum_{i=0}^{p} \alpha_i = W_{\text{total}} & \\ (3)\ \alpha_0 w_0 \leq T_f & \text{(computation of the master)}\\ (4)\ \alpha_1 g_1 + \alpha_1 w_1 \leq T_f & \text{(first communication)}\\ (5)\ \sum_{j=1}^{i} \alpha_j g_j + \alpha_i w_i \leq T_f & (i\text{th communication}).\end{cases}\end{array}$$

Closed-form expressions have been derived for the optimal solution of the STARLINEAR problem, both for homogeneous and heterogeneous star networks [2]. The formulation in terms of linear program is useful because it can be directly extended to deal with tree networks (Section 3.2). It also serves as the basis for the mixed linear program formulation in the affine cost model (Section 3.3), and to derive the asymptotically optimal multiround algorithm (Section 4.3). Finally, we point out that the first use of linear programming in the DLT literature appeared in [22].

3.2 Tree Network and Linear Cost Model

All the results in the previous section can be extended to a tree-shaped network. There is, however, a key difference with the beginning of Section 3.1: Each worker now is capable of computing and communicating to one of its children simultaneously. However, because of the one-round hypothesis, no overlap can occur with the incoming communication from the node's parent.

We use a recursive approach, which replaces any set of leaves and their parent by a single worker of equivalent computing power. This idea of collapsing subnets into equivalent processors originates in [23], [24].

Lemma 3. *A single-level tree network with parent P_0 (with input link of capacity g_0 and cycle-time w_0) and p children P_i, $1 \leq i \leq p$ (with input link of capacity g_i and cycle-time w_i), where $g_1 \leq g_2 \leq \ldots \leq g_p$, is equivalent to a single node with same input link capacity g_0 and cycle-time $w_{-1} = 1/W$ (see Fig. 5), where W is the solution to the linear program:*

$$\begin{array}{l}\text{Maximize } W,\\ \text{subject to}\\ \begin{cases}(1)\ \alpha_i \geq 0 & 0 \leq i \leq p\\ (2)\ \sum_{i=0}^{p} \alpha_i = W & \\ (3)\ W g_0 + \alpha_0 w_0 \leq 1 & \\ (4)\ W g_0 + \alpha_1 g_1 + \alpha_1 w_1 \leq 1 & \\ (5)\ W g_0 + \sum_{j=1}^{i} \alpha_j g_j + \alpha_i w_i \leq 1. & \end{cases}\end{array}$$

Proof. Here, instead of minimizing the time T_f required to execute load W, we aim at determining the maximum

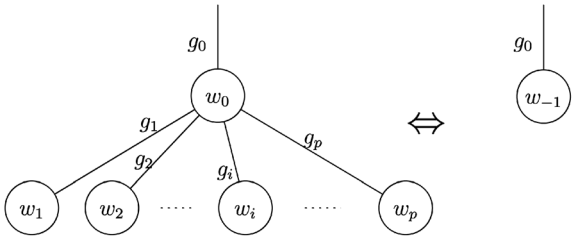


Fig. 5. Replacing a single-level tree by an equivalent node.

amount of work W that can be processed in one time-unit. Obviously, after the end of the incoming communication, the parent should be constantly computing. We know that all children 1) participate in the computation and 2) terminate execution at the same time. Finally, the optimal ordering for the children is given by Proposition 2. This completes the proof. Note that inequalities (3), (4), and (5) will be in fact equalities in the solution of the linear program, so that we can easily derive a closed-form expression for $w_{-1} = 1/W$. □

Lemma 3 provides a constructive way of solving the problem for a general tree. First, we traverse it from bottom to top, replacing each single-level tree by the equivalent node. We do this until there remains a single star. We solve the problem for the star, using the results of Section 3.1. Then, we traverse the tree from top to bottom, and undo each transformation in the reverse ordering. Going back to a reduced node, we know which amount of time it is working. Knowing the ordering, we know which amount of time each of the children is working. If one of these children is a leaf node, we have computed its load. If it is a reduced node, we apply the transformation recursively.

Instead of this pair of tree traversals, we could write down the linear program for the whole tree: When it receives something, a given node knows exactly what to do: compute itself all the remaining time, and feed its children in decreasing bandwidth order. However, the size of the linear program would grow proportionally to the size of the tree, hence the recursive solution is to be preferred.

3.3 Star Network and Affine Cost Model

To the best of our knowledge, the complexity of the STARAFFINE problem is open. The main difficulty arises from resource selection: contrarily to the linear case where all workers participate in the optimal solution, it seems difficult to decide which resources to use when latencies are introduced. However, the second property proved in Proposition 1, namely, simultaneous termination, still holds true.

Proposition 3. *In an optimal solution of the STARAFFINE problem, all participating workers finish computing at the same time.*

Proof. The proof is very similar to the STARLINEAR case. Details can be found in the extended version [16]. □

Proposition 4. *If the load is large enough, then for any optimal solution 1) all workers participate and 2) chunks must be sent in the order of nondecreasing link capacities g_i.*

Proof. Consider a valid solution of the STARAFFINE problem with time bound T. Suppose, without loss of generality, that $\alpha_{\sigma(1)}$ units of load are sent to $P_{\sigma(1)}$, then $\alpha_{\sigma(2)}$ to $P_{\sigma(2)}$, ... and, finally, $\alpha_{\sigma(k)}$ to P_k, where $\mathcal{S} = \{P_{\sigma(1)}, \ldots, P_{\sigma(k)}\}$ is the set of workers that participate to the computation. Here, σ represents the communication ordering and is a one-to-one mapping from $[1 \ldots k]$ to $[1 \ldots p]$. Moreover, let n^{TASK} denote the optimal number of units of load that can be processed using this set of workers and this ordering.

- Consider the following instance of the STARLINEAR problem, with k workers $P'_{\sigma(1)}, \ldots, P'_{\sigma(k)}$, where $\forall i,\ G'_i = 0, W'_i = 0, g'_i = g_i, w'_i = w_i$ and $T' = T$. Since all computation and communication latencies have been taken out, the optimal number of units of load n_1^{TASK} processed by this instance is larger than the number of units of load n^{TASK} processed by the initial platform. From Theorem 1, the value of n_1^{TASK} is given by a formula
$$n_1^{\mathrm{TASK}} = f(\mathcal{S}, \sigma) \cdot T,$$
where $f(\mathcal{S}, \sigma)$ is either derived from the linear program, or explicitly given by a closed form expression in [25]. What matters here is that the value of n_1^{TASK} is proportional to T.
- Consider now the following instance of the STARLINEAR problem, with k workers $P'_{\sigma(1)}, \ldots, P'_{\sigma(k)}$, where $\forall i,\ G'_i = 0, W'_i = 0, g'_i = g_i, w'_i = w_i$ and $T' = T - \sum_{i \in \mathcal{S}} (G_i + W_i)$. Clearly, the optimal number of units of load n_2^{TASK} processed by this instance of the STARLINEAR problem is lower than n^{TASK} since it consists in adding all the communication and computation latencies before the beginning of the processing. Moreover, as previously n_2^{TASK} is given by the formula
$$n_2^{\mathrm{TASK}} = f(\mathcal{S}, \sigma)(T - \sum_{i \in \mathcal{S}} (G_i + W_i)).$$

Therefore, we have

$$f(\mathcal{S}, \sigma)\left(1 - \frac{\sum_{i \in \mathcal{S}} (G_i + W_i)}{T}\right) \leq \frac{n^{\mathrm{TASK}}}{T} \leq f(\mathcal{S}, \sigma).$$

Hence, when T becomes arbitrarily large, then the throughput of the platform, $\frac{n^{\mathrm{TASK}}}{T}$, becomes arbitrarily close to $f(\mathcal{S}, \sigma)$, i.e., the optimal throughput if there were no communication and computation latencies. Moreover, we have proven that if there are no latencies, then $f(\mathcal{S}, \sigma)$ is maximal when $\mathcal{S}$ is the set of all the workers, and when σ satisfies

$$g_j > g_i \Longrightarrow \sigma(i) > \sigma(j).$$

Therefore, when T is sufficiently large, then all the workers should be used and the chunks should be sent to workers in the ordering of non decreasing link capacities

g_i. In this case, if $g_1 \leq \ldots \leq g_p$, then the following linear system provides an asymptotically optimal solution

$$\forall i, \quad \sum_{k=1}^{i}(G_k + g_k\alpha_k) + W_i + g_i w_i = T.$$

This solution is optimal if all g_i are different. Determining the best way to break ties among workers having the same bandwidth is an open question. □

In the general case, we do not know whether there exists a polynomial-time algorithm to solve the STARAFFINE problem. However, we can provide the solution (with potentially exponential cost) as follows: We start from the mixed linear programming formulation of the problem proposed by Drozdowski [8], and we extend it to include resource selection. In the following program, y_j is a Boolean variable that equals 1 if P_j participates in the solution, and $x_{i,j}$ is a Boolean variable that equals 1 if P_j is chosen for the ith communication from the master:

$$\text{Minimize } T_f,$$
$$\text{subject to}$$
$$\begin{cases} (1)\ \alpha_i \geq 0 & 1 \leq i \leq p \\ (2)\ \sum_{i=1}^{p} \alpha_i = W_{\text{total}} & \\ (3)\ y_j \in \{0,1\} & 1 \leq j \leq p \\ (4)\ x_{i,j} \in \{0,1\} & 1 \leq i,j \leq p \\ (5)\ \sum_{i=1}^{p} x_{i,j} = y_j & 1 \leq j \leq p \\ (6)\ \sum_{j=1}^{p} x_{i,j} \leq 1 & 1 \leq i \leq p \\ (7)\ \alpha_j \leq W y_j & 1 \leq j \leq p \\ (8)\ \sum_{j=1}^{p} x_{1,j} & \\ (G_j + \alpha_j g_j + W_j + \alpha_j w_j) \leq T_f \ \text{(first communication)} & \\ (9)\ \sum_{k=1}^{i-1}\sum_{j=1}^{p} x_{k,j}(G_j + \alpha_j g_j) + & \\ \sum_{j=1}^{p} x_{i,j}(G_j + \alpha_j g_j + W_j + \alpha_j w_j) \leq T_f & \\ 2 \leq i \leq p \quad (i\text{th communication}). & \end{cases}$$

Equation (5) implies that P_j is involved in exactly one communication if $y_j = 1$, and in no communication otherwise. Equation (6) states that at most one worker is activated for the ith communication; if $\sum_{j=1}^{p} x_{i,j} = 0$, the ith communication disappears. Equation (7) states that no work is given to nonparticipating workers (those for which $y_j = 0$), but is automatically fulfilled by participating ones. Equation (8) is a particular case of (9), which expresses that the worker selected for the ith communication (where $i = 1$ in (8) and $i \geq 2$ in (9)) must wait for the previous communications to complete before starting its own communication and computation, and that all this quantity is a lower bound of the makespan. Contrarily to the formulation of Drozdowski [8], this mixed linear program always has a solution, even if a strict subset of the resources are participating. We state this result formally.

Proposition 5. *The optimal solution for the STARAFFINE problem is given by the solution of the mixed linear program above (with potentially exponential cost).*

3.4 Tree Network and Affine Cost Model

This is the most difficult platform/model combination, and very few results are known. However, we point out that Proposition 4 can be extended to arbitrary tree networks: When T becomes arbitrarily large, latencies become negligible, and an asymptotically optimal behavior is obtained by involving all resources and by having each parent communicate with its children in order of non decreasing link capacities.

4 Multiround Algorithms

Under the one-port communication model described in Section 2.1, one-round algorithms lead to poor utilization of the workers. As seen in Fig. 3, worker P_i remains idle from time 0 to time T_i. To alleviate this problem, *multiround* algorithms have been proposed. These algorithms dispatch the load in multiple rounds of work allocation and, thus, improve overlap of communication with computation. By comparison with one-round algorithms, work on multiround algorithms has been scarce. The two main questions that must be answered are: 1) What should the chunk sizes be at each round? and 2) how many rounds should be used? The majority of works on multiround algorithms assume that the number of rounds is fixed and we review corresponding results and open questions in Section 4.1. In Section 4.2, we describe recent work that attempts to answer question 2). Finally, we deal with asymptotic results in Section 4.3, which of course are of particular interest when the total load W_{total} is very large.

4.1 Fixed Number of Rounds, Homogeneous Star Network, Affine Costs

As for one-round algorithms, a key question is that of the order in which chunks should be sent to the workers. However, to the best of our knowledge, all previous work on multiround algorithms with fixed number of rounds only offer solution for homogeneous platforms, in which case worker ordering is not an issue. Given a fixed number of rounds M, the load is divided into $p \times M$ chunks, each corresponding to α_j ($j = 0, \ldots, pM-1$) units of load such that $\sum_{j=0}^{pM-1} \alpha_j = W_{\text{total}}$. The objective is to determine the α_j values that minimize the overall makespan.

Intuitively, the chunk size should be small in the first rounds, so as to start all workers as early as possible and, thus, maximize overlap of communication with computation. It has be shown that the chunk sizes should then increase to optimize the usage of the total available bandwidth of the network and to amortize the potential overhead associated with each chunk. In the last round, chunk sizes should be decreasing so that all workers finish computing at the same time (following the same principle as in Section 3). Such a schedule is depicted in Fig. 6 for four workers.

Bharadwaj et al. were the first to address this problem with the multiinstallment scheduling algorithm described in [26]. They reduce the problem of finding an optimal schedule to that of finding a schedule that has essentially the following three properties:

1. there is no idle time between consecutive communications on the bus;
2. there is no idle time between consecutive computation on each worker; and

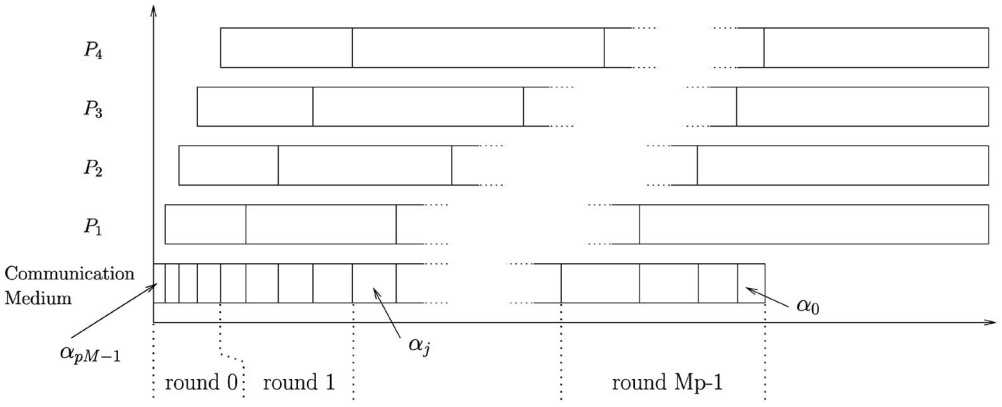


Fig. 6. Pattern of a solution for dispatching the load of a divisible job, using a bus network ($g_i = g$), in multiple rounds, for four workers. All four workers complete execution at the same time. Chunk sizes increase during each of the first $M-1$ rounds and decrease during the last round.

3. all workers should finish computing at the same time.

These properties guarantee that the network and compute resources are at maximum utilization.

In [26], the authors consider only linear costs for both communication and computation. The three conditions above make it possible to obtain a recursion on the α_j series. This recursion must then be solved to obtain a close form expression for the chunk sizes. One method to solve the recursion is to use generating functions and the rational expansion theorem [27].

We recently extended the multiinstallment approach to account for affine costs [28]. This was achieved by rewriting the chunk size recursion in a way that is more amenable to the use of generating functions when fixed latencies are incurred for communications and computations. Since it is more general but similar in spirit, we only present the affine case here.

For technical reasons, as in [26], we number the chunks in the reverse order in which they are allocated to workers: the last chunk is numbered 0 and the first chunk is numbered $Mp-1$. Instead of developing a recursion on the α_j series directly, we define $\gamma_j = \alpha_j * w$, i.e., the time to compute a chunk of size α_j on a worker not including the W latency. Recall that in this section we only consider homogeneous platforms and, thus, $w_q = w$, $W_q = W$, $g_q = g$, and $G_q = G$ for all workers $q = 1, \ldots, p$. The time to communicate a chunk of size α_j to a worker is $G + \gamma_i/R$, where R is the computation-communication ratio of the platform: w/g. We can now write the recursion on the γ_j series:

$$\forall\, j \geq P \;\; W + \gamma_j = (\gamma_{j-1} + \gamma_{j-2} + \gamma_{j-3} + \cdots + \gamma_{j-N})/R + P \times G, \quad (7)$$

$$\forall\, 0 \leq j < P \;\; W + \gamma_j = (\gamma_{j-1} + \gamma_{j-2} + \gamma_{j-3} + \cdots + \gamma_{j-N})/R + j \times G + \gamma_0, \quad (8)$$

$$\forall\, j < 0 \;\; \gamma_j = 0. \quad (9)$$

Equation (7) ensures that there is no idle time on the bus and at each worker in the first $M-1$ rounds. More specifically, (7) states that a worker must compute a chunk in exactly the time required for all the next P chunks to be communicated, including the G latencies. This equation is valid only for $j \geq P$. For $j < P$, i.e., the last round, the recursion must be modified to ensure that all workers finish computing at the same time, which is expressed in (8). Finally, (9) ensures that the two previous equations are correct by taking care of out-of-range α_j terms. This recursion describes an infinite α_j series, and the solution to the scheduling problems is given by the first pM values.

As in [26], we use generating functions as they are convenient tools for solving complex recursions elegantly. Let $\mathcal{G}(x)$ be the generating function for the series γ_j, that is, $\mathcal{G}(x) = \sum_{j=0}^{\infty} \gamma_j x^j$. Multiplying (7) and (8) by x^j, manipulating the indices, and summing the two gives:

$$\mathcal{G}(x) = \frac{(\gamma_0 - P \times G)(1 - x^P) + (P \times G - W) + G\left(\frac{x(1 - x^{P-1})}{1 - x} - (P-1)x^P\right)}{(1-x) - x(1-x^P)/R}.$$

The rational expansion method [27] can then be used to determine the coefficients of the above polynomial fraction, given the roots of the denominator polynomial, $Q(x)$. The values of the γ_j series and, thus, of the α_j series, follow directly. If $Q(x)$ has only roots of degree 1, then the simple rational expansion theorem can be used directly. Otherwise, the more complex general rational expansion theorem must be used. In [28], we show that if $R \neq P$, then $Q(x)$ has only roots of degree one. If $R = P$, then the only root of degree higher than 1 is root $x = 1$ and it is of degree 2, which makes the application of the general theorem straightforward. Finally, the value of γ_0 can be computed by writing that $\sum_{j=0}^{Mp-1} \gamma_j = W_{\text{total}} \times w$. All technical details on the above derivations are available in a technical report [28]. We have thus obtained a closed-form expression for optimal multiinstallment schedule on a homogeneous star network with affine costs.

4.2 Computed Number of Rounds, Star Network, Affine Costs

The work presented in the previous section assumes that the number of rounds is fixed and provided as input to the scheduling algorithm. In the case of linear costs, the authors in [2] recognize that infinitely small chunks would lead to an optimal multiround schedule, which implies an infinite number of rounds. When considering more realistic affine costs there is a clear trade off. While using more rounds

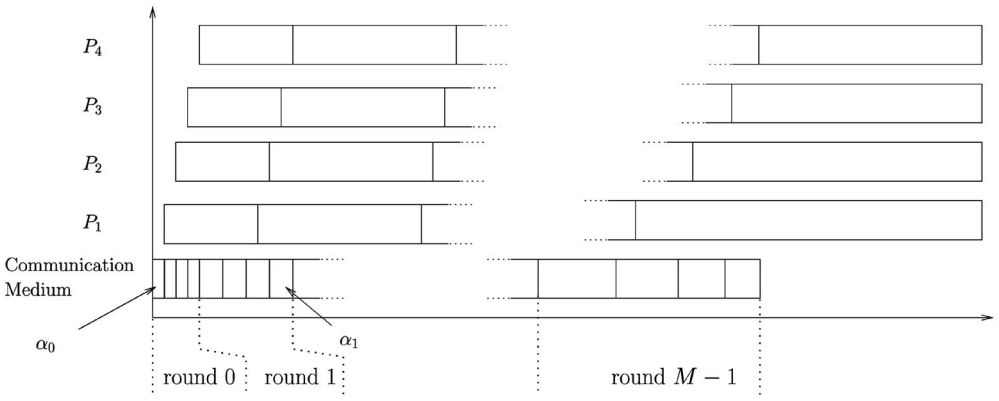


Fig. 7. Pattern of a solution for dispatching the load of a divisible job, using a bus network ($g_i = g$), in multiple **uniform** rounds, for four workers. All workers complete execution at the same time. Chunk sizes are fixed within the first $M-1$ rounds but increase from round to round. Chunk sizes decrease during the last round.

leads to better overlap of communication with computation, using fewer rounds reduces the overhead due to the fixed latencies. Therefore, a key question is: What is the optimal number of rounds for multiround scheduling on a star network with affine costs?

While this question is still open for the recursion described in Section 4.1, our work in [29] proposes a scheduling algorithm, Uniform Multi-Round (UMR), that uses a restriction on the chunk size: All chunks sent to workers during a round are identical. This restriction limits the ability to overlap communication with computation, but makes it possible to derive an optimal number of rounds due to a simpler recursion on chunk sizes. Furthermore, this approach is applicable to both homogeneous and heterogeneous platforms. We only describe here the algorithm in the homogeneous case. The heterogeneous case is similar but involves more technical derivations and we refer the reader to [30] for all details.

As seen in Fig. 7, chunks of identical size are sent out to workers within each round. Because chunks are uniform it is not possible to obtain a schedule with no idle time in which each worker finishes receiving a chunk of load right when it can start executing it. Note in Fig. 7 that workers can have received a chunk entirely while not having finished to compute the previous chunk. The condition that a worker finishes receiving a chunk right when it can start computing is only enforced for the worker P_p, which is also seen in the figure. Finally, the uniform round restriction is removed for the last round. As in the multiinstallment approach described in Section 4.1, chunks of decreasing sizes are sent to workers in the last round so that they can all finish computing at the same time.

Let α_j be the chunk size at round j, which is used for all workers during that round. We derive a recursion on the chunk size. To maximize bandwidth utilization, the master must finish sending work for round $j+1$ to all workers right when worker P finishes computing for round j. This can be written as

$$W + \alpha_j w = P(G + \alpha_{j+1} g), \tag{10}$$

which reduces to

$$\alpha_j = \left(\frac{g}{Pw}\right)^j (\alpha_0 - \gamma) + \gamma, \tag{11}$$

where $\gamma = \frac{1}{w - Pg} \times (PG - W)$. The case in which $w - Pg = 0$ leads to a simpler recursion and we do not consider it here for the sake of brevity.

Given this recursion on the chunk sizes, it is possible to express the scheduling problem as a constrained minimization problem. The total makespan, $\mathcal{M}$, is:

$$\mathcal{M}(M, \alpha_0) = \frac{W_{\text{total}}}{P} + MW + \frac{1}{2} \times P(G + g\alpha_0),$$

where the first term is the time for worker P to perform its computations, the second term the overhead incurred for each of these computations, and the third term is the time for the master to dispatch all the chunks during the first round. Note that the $\frac{1}{2}$ factor in the above equation is due to the last round during which UMR does not keep chunk sizes uniform so that all workers finish computing at the same time (see [29] for details).

Since all chunks must satisfy the constraint that they add up to the entire load, one can write that:

$$\mathcal{G}(M, \alpha_0) = \sum_{j=0}^{M-1} P\alpha_j - W_{\text{total}} = 0. \tag{12}$$

The scheduling problem can now be expressed as the following constrained optimization problem: minimize $\mathcal{M}(M, \alpha_0)$ subject to $\mathcal{G}(M, \alpha_0) = 0$. An analytical solution using the Lagrange Multiplier method [31] is given in [29], which leads to a single equation for the optimal number of round, M^*. This equation cannot be solved analytically, but is eminently amenable to a numerical solution, e.g., using a bisection method.

The UMR algorithm is a heuristic and has been evaluated in simulation for a large number of scenarios [30]. In particular, a comparison of UMR with the multiinstallment algorithm discussed in Section 4.1 demonstrates the following: The uniform chunk restriction minimally degrades performance compared to multiinstallment when latencies are small (i.e., when costs are close to being linear). However, as soon as latencies become significant, this performance degradation is offset by the fact that an

optimal number of rounds can be computed and UMR outperforms multiinstallment consistently. Finally, note that a major benefit of UMR is that, unlike multiinstallment, it is applicable to heterogeneous platforms. In this case, the question of worker ordering arises and UMR uses the same criterion as that given in Proposition 2: Workers are ordered by nondecreasing link capacities.

4.3 Asymptotic Performance, Star Network, Affine Costs

In this section, we derive asymptotically optimal algorithms for the multiround distribution of divisible loads. An algorithm is asymptotically optimal if the ratio of the time to execute a workload W_{total} over the optimal time to execute this workload tends to 1 as W_{total} tends to infinity. As in previous sections, we use a star network with affine costs.

The sketch of the algorithm that we propose is as follows: The overall processing time T is divided into k regular periods of duration T_p so that $T = kT_p$, with k to be determined. Intuitively, the trade off is as follows: We will describe a steady-state operation, divided into periods of duration T_p. Initialization and clean-up will lead to "loosing" a few periods at the beginning and at the end of the execution, so there should be enough periods to render this sacrifice negligible. However, within each period, we want the impact of start-up costs to become negligible too. Hence, the period length should be large enough. Altogether, a good trade off will be to use periods whose length T_p is proportional to the square-root of the optimal execution time $T^\star$, hence the number of periods k will also be proportional to $T^\star$. The major difficulty is to perform the resource selection. To decide which resource to involve, we come back to a linear programming formulation, as detailed below.

During a period of duration T_p, the master sends α_i units of load to worker P_i. It may well be the case that not all the workers are involved in the computation. Let $\mathcal{I} \subset \{1, \ldots, p\}$ represent the subset of indices of participating workers. For all $i \in \mathcal{I}$, the α_is must satisfy the following inequality, stating that communication resources are not exceeded:

$$\sum_{i \in \mathcal{I}} (G_i + \alpha_i g_i) \leq T_p. \tag{13}$$

Since the workers can overlap communications and processing, the following inequalities also hold true:

$$\forall i \in \mathcal{I}, \quad W_i + \alpha_i w_i \leq T_p.$$

Let us denote by $\frac{\alpha_i}{T_p}$ the average number of units of load that worker P_i processes during one time unit, then the system becomes

$$\begin{cases} \forall i \in \mathcal{I}, \quad \frac{\alpha_i}{T_p} w_i & \leq 1 - \frac{W_i}{T_p} \quad \text{(overlap)} \\ \sum_{i \in \mathcal{I}} \frac{\alpha_i}{T_p} g_i & \leq 1 - \frac{\sum_{i \in \mathcal{I}} G_i}{T_p} \quad (1\text{-port model}), \end{cases}$$

and our aim is to maximize the overall number of units of load processed during one time unit, i.e., $\rho = \sum_{i \in \mathcal{I}} \frac{\alpha_i}{T_p}$.

Consider the following linear program:

$$\text{Maximize} \sum_{i=1}^{p} \frac{\alpha_i}{T_p},$$

subject to

$$\begin{cases} \forall 1 \leq i \leq p, \quad \frac{\alpha_i}{T_p} w_i \leq 1 - \frac{\sum_{i=1}^{p} G_i + W_i}{T_p} \\ \sum_{i=1}^{p} \frac{\alpha_i}{T_p} g_i \leq 1 - \frac{\sum_{i=1}^{p} G_i + W_i}{T_p}. \end{cases}$$

This linear program is more constrained than the previous one since $1 - \frac{W_i}{T_p}$ and $1 - \frac{\sum_{i \in \mathcal{I}} G_i}{T_p}$ have been replaced by $1 - \frac{\sum_{i=1}^{p} G_i + W_i}{T_p}$ in p inequalities. The linear program can be solved using a package similar to Maple (we have rational numbers), but it turns out that the technique developed in [32] enables us to obtain the solution in closed form. We refer the reader to [32] for the complete proof. Let us sort the g_is so that $g_1 \leq g_2 \leq \ldots \leq g_p$, and let q be the largest index so that $\sum_{i=1}^{q} \frac{g_i}{w_i} \leq 1$. If $q < p$, let ς denote the quantity $1 - \sum_{i=1}^{q} \frac{g_i}{w_i}$. If $p = q$, we set $\varsigma = g_{q+1} = 0$, in order to keep homogeneous notations. This corresponds to the case where the full use of all the workers does not saturate the 1-port assumption for outgoing communications from the master. The optimal solution to the linear program is obtained with

$$\forall 1 \leq i \leq q, \quad \frac{\alpha_i}{T_p} = \frac{1 - \frac{\sum_{i=1}^{p} G_i + W_i}{T_p}}{g_i},$$

and (if $q < p$):

$$\frac{\alpha_{q+1}}{T_p} = \left(1 - \frac{\sum_{i=1}^{p} G_i + W_i}{T_p}\right)\left(\frac{\varsigma}{g_{q+1}}\right),$$

and $\alpha_{q+2} = \alpha_{q+3} = \ldots = \alpha_p = 0$.

With these values, we obtain a distribution whose throughput is:

$$\rho = \sum_{i=1}^{p} \frac{\alpha_i}{T_p} = \left(1 - \frac{\sum_{i=1}^{p} G_i + W_i}{T_p}\right)\rho_{opt}$$

$$\text{with } \rho_{opt} = \left(\sum_{i=1}^{q} \frac{1}{w_i} + \frac{\varsigma}{g_{p+1}}\right).$$

Consider now an optimal multiround distribution of B load units, and denote by $\rho^\star$ the average number of load units that is processed within one unit of time. If we denote by $\beta_i^\star$ the average number of units of load that is processed by worker P_i within one unit of time, the $\beta_i^\star$s satisfy the following set of inequalities, in which the G_is have been removed:

$$\begin{cases} \forall 1 \leq i \leq p, \quad \beta_i^\star w_i \leq 1 \\ \sum_{i=1}^{p} \beta_i^\star g_i \leq 1. \end{cases}$$

Here, because we have no latencies, we can safely assume that all the workers are involved (and let $\beta_i^\star = 0$ for some of them). We derive that:

$$\rho^\star \leq \rho_{opt}.$$

If we denote by $T^\star$ the optimal time necessary to process B units of load, then

$$T^\star \geq \frac{B}{\rho^\star} \geq \frac{B}{\rho_{opt}}.$$

Let us denote by T the time necessary to process all B units of load with the algorithm that we propose. Since the first period is lost for processing, then the number k of necessary periods satisfies $\rho T_p(k-1) \geq B$ so that we choose

$$k = \left\lceil \frac{B}{\rho T_p} \right\rceil + 1.$$

Therefore,

$$T \leq \frac{B}{\rho} + 2T_p \leq \frac{B}{\rho_{opt}} \left(\frac{1}{1 - \sum_{i=1}^{p} \frac{G_i + W_i}{T_p}} \right) + 2T_p$$

and, therefore, if $T_p \geq 2\sum_{i=1}^{p} G_i + W_i$,

$$T \leq \frac{B}{\rho_{opt}} + 2\frac{B}{\rho_{opt}} \sum_{i=1}^{p} \frac{(G_i + W_i)}{T_p} + 2T_p.$$

Finally, if we set $T_p = \sqrt{\frac{B}{\rho_{opt}}}$, we check that

$$T \leq T^\star + 2\left(\sum_{i=1}^{p} (G_i + W_i) + 1 \right) \sqrt{T^\star} = T^\star + O(\sqrt{T^\star})$$

and, thus, that

$$\frac{T}{T^\star} \leq 1 + 2\left(\sum_{i=1}^{p} (G_i + W_i) + 1 \right) \frac{1}{\sqrt{T^\star}} = 1 + O\left(\frac{1}{\sqrt{T^\star}} \right),$$

which completes the proof of the asymptotic optimality of our algorithm.

Note that resource selection is part of our explicit solution to the linear program: To give an intuitive explanation of the analytical solution, workers are greedily selected, fast-communicating workers first, as long as the communication to communication-added-to-computation ratio is not exceeded.

We formally state our main result, for which a detailed proof can be found in [33].

Theorem 2. *For arbitrary values of G_i, g_i, W_i, and w_i and assuming communication-computation overlap, the previous periodic multiround algorithm is asymptotically optimal. Closed-form expressions for resource selection and task assignment are provided by the algorithm, whose complexity does not depend upon the total amount of work to execute.*

5 Conclusion

The goal of this paper was to present a unified discussion of divisible load scheduling results for star and tree networks. In Section 3, we have discussed one-round algorithms for which the two main issues are: 1) selection and ordering of the workers and 2) computation of the chunk sizes. Section 4 focused on multiround algorithms, with the two main issues: 1) computation of chunk sizes at each round and 2) choice of the number of rounds. Section 4 also discussed multiround scheduling for maximizing asymptotic application performance. For both classes of algorithms, we have revisited previously published results, presented novel results, and clearly identified open questions. Our overall goal was to identify promising research directions and foster that research thanks to our unified and synthesized framework.

We have discussed affine cost models and have seen that they often lead to significantly more complex scheduling problems than when linear models are assumed. These models are generally considered more realistic, and we even contend that, given current trends, linear models are quickly becoming increasingly inappropriate. In terms of communication, technology trends indicate that available network bandwidth is rapidly augmenting. Therefore, latencies account for an increasingly large fraction of communication costs. A similar observation can be made in terms of computation. Due to the absence of stringent synchronization requirements, divisible workload applications are amenable to deployment on widely distributed platforms. For instance, computational grids [11] are attractive for deploying large divisible workloads. However, initiating computation on these platforms incurs potentially large latencies (i.e., due to resource discovery, authentication, creation of new processes, etc.). Consequently, it is clear that divisible workload research should focus on affine cost models for both communication and computation.

Acknowledgments

The authors thank the reviewers for their helpful comments and suggestions, which greatly improved the final version of the paper.

References

[1] B.A. Shirazi, A.R. Hurson, and K.M. Kavi, *Scheduling and Load Balancing in Parallel and Distributed Systems.* IEEE Press, 1995.

[2] V. Bharadwaj, D. Ghose, V. Mani, and T. Robertazzi, *Scheduling Divisible Loads in Parallel and Distributed Systems.* IEEE Press, 1996.

[3] S. Chan, V. Bharadwaj, and D. Ghose, "Large Matrix-Vector Products on Distributed Bus Networks with Communication Delays Using the Divisible Load Paradigm: Performance and Simulation," *Math. and Computers in Simulation,* vol. 58, pp. 71-92, 2001.

[4] C. Lee and M. Hamdi, "Parallel Image Processing Applications on a Network of Workstations," *Parallel Computing,* vol. 21, pp. 137-160, 1995.

[5] X. Li, V. Bharadwaj, and C. Ko, "Distributed Image Processing on a Network of Workstations," *Int'l J. Computers and Applications,* (ACTA Press), vol. 25, no. 2, pp. 1-10, 2003.

[6] D. Altilar and Y. Paker, "An Optimal Scheduling Algorithm for Parallel Video Processing," *Proc. IEEE Int'l Conf. Multimedia Computing and Systems,* 1998.

[7] "Optimal Scheduling Algorithms for Communication Constrained Parallel Processing," *Proc. Euro-Par 2002,* pp. 197-206, 2002.

[8] M. Drozdowski, "Selected Problems of Scheduling Tasks in Multiprocessor Computing Systems," PhD dissertation, Instytut Informatyki Politechnika Poznanska, Poznan, 1997.

[9] J. Blazewicz, M. Drozdowski, and M. Markiewicz, "Divisible Task Scheduling—Concept and Verification," *Parallel Computing,* vol. 25, pp. 87-98, 1999.

[10] R. Wang, A. Krishnamurthy, R. Martin, T. Anderson, and D. Culler, "Modeling Communication Pipeline Latency," *Measurement and Modeling of Computer Systems (Proc. SIGMETRICS '98 Conf.),* pp. 22-32, 1998.

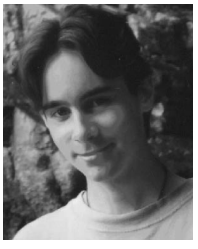
[11] *The Grid: Blueprint for a New Computing Infrastructure,* I. Foster and C. Kesselman, eds., San Francisco, Calif.: Morgan Kaufmann Publishers, Inc., 1999.

[12] V. Bharadwaj, D. Ghose, and T. Robertazzi, "Divisible Load Theory: A New Paradigm for Load Scheduling in Distributed Systems," *Cluster Computing,* vol. 6, no. 1, pp. 7-17, 2003.

[13] T. Robertazzi, "Ten Reasons to Use Divisible Load Theory," *Computer,* vol. 36, no. 5, pp. 63-68, 2003.
[14] *Cluster Computing,* special issue on divisible load scheduling, D. Ghose and T. Robertazzi, eds., 2003.
[15] T. Robertazzi, "Divisible Load Scheduling," http://www.ece.sunysb.edu/tom/dlt.html, year?
[16] O. Beaumont, H. Casanova, A. Legrand, Y. Robert, and Y. Yang, "Scheduling Divisible Loads for Star and Tree Networks: Main Results and Open Problems," Technical Report RR-2003-41, LIP, ENS Lyon, France, Sept. 2003.
[17] V. Bharadwaj and G. Barlas, "Scheduling Divisible Loads with Processor Release Times and Finite Size Buffer Capacity Constraints in Bus Networks," *Cluster Computing,* vol. 6, no. 1, pp. 63-74, 2003.
[18] "Efficient Scheduling Strategies for Processing Multiple Divisible Loads on Bus Networks," *J. Parallel and Distributed Computing,* vol. 62, pp. 132-151, 2002.
[19] D. Yu and T. Robertazzi, "Divisible Load Scheduling for Grid Computing," *Proc. 15th Int'l Conf. Parallel and Distributed Computing and Systems,* 2003.
[20] H. Wong, D. Yu, V. Bharadwaj, and T. Robertazzi, "Data Intensive Grid Scheduling: Multiple Sources with Capacity Constraints," *Proc. 15th Int'l Conf. Parallel and Distributed Computing and Systems,* 2003.
[21] A. Schrijver, *Theory of Linear and Integer Programming.* New York: John Wiley & Sons, 1986.
[22] R. Agrawal and H. Jagadish, "Partitioning Techniques for Large-Grained Parallelism," *IEEE Trans. Computers,* vol. 37, no. 12, pp. 1627-1634, 1988.
[23] T. Robertazzi, "Processor Equivalence for a Linear Daisy Chain of Load Sharing Processors," *IEEE Trans. Aerospace and Electronic Systems,* vol. 29, pp. 1216-1221, 1993.
[24] S. Bataineh, T. Hsiung, and T.G. Robertazzi, "Closed Form Solutions for Bus and Tree Networks of Processors Load Sharing a Divisible Job," *IEEE Trans. Computers,* vol. 43, no. 10, pp. 1184-1196, Oct. 1994.
[25] S. Charcranoon, T. Robertazzi, and S. Luryi, "Optimizing Computing Costs Using Divisible Load Analysis," *IEEE Trans. Computers,* vol. 49, no. 9, pp. 987-991, Sept. 2000.
[26] V. Bharadwaj, D. Ghose, and V. Mani, "Multi-Installment Load Distribution in Tree Networks with Delays," *IEEE Trans. Aerospace and Electronc Systems,* vol. 31, no. 2, pp. 555-567, 1995.
[27] R. Graham, D. Knuth, and O. Patashnik, *Concrete Mathematics.* Wiley, 1994.
[28] Y. Yang and H. Casanova, "Extensions to the Multi-Installment Algorithm: Affine Costs and Output Data Transfers," Technical Report CS2003-0754, Dept. of Computer Science and Eng., Univ. of California, San Diego, July 2003.
[29] "UMR: A Multiround Algorithm for Scheduling Divisible Workloads," *Proc. Int'l Parallel and Distributed Processing Symp.,* Apr. 2003.
[30] "Multi-Round Algorithm for Scheduling Divisible Workload Applications: Analysis and Experimental Evaluation," Technical Report CS2002-0721, Dept. of Computer Science and Eng., University of California, San Diego, 2002.
[31] D. Bertsekas, *Constrained Optimization and Lagrange Multiplier Methods.* Belmont, Mass.: Athena Scientific, 1996.
[32] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert, "Bandwidth-Centric Allocation of Independent Tasks on Heterogeneous Platforms," *Proc. Int'l Parallel and Distributed Processing Symp.,* 2002.
[33] O. Beaumont, A. Legrand, and Y. Robert, "Optimal Algorithms for Scheduling Divisible Workloads on Heterogeneous Systems," Technical Report 2002-36, LIP, ENS Lyon, France, Oct. 2002.

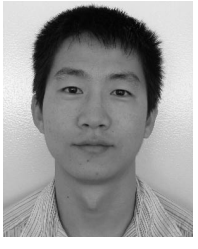
Olivier Beaumont received the PhD degree from the Université de Rennes in 1999. He is currently an associate professor in the LaBRI laboratory in Bordeaux. His main research interests are parallel algorithms on distributed memory architectures.

Henri Casanova received the BS degree from the Ecole Nationale Supérieure d'Electronique, d'Electrotechnique, d'Informatique et d'Hydraulique de Toulouse, France in 1993, the MS degree from the Université Paul Sabatier, Toulouse, France in 1994, and the PhD degree from the University of Tennessee, Knoxville in 1998. He is an adjunct assistant professor of computer science and engineering at the University of California, San Diego, an associate research scientist at the San Diego Supercomputer Center, and the founder and director of the Grid Research And Development Laboratory (GRAIL). His research interests are in the area of parallel and distributed, with a focus on modeling, simulation, and scheduling. He is a member of the IEEE and the IEEE Computer Society.



Arnaud Legrand received the PhD degree from École normale supérieure de Lyon in 2003. He is currently a postdoctoral researcher in the LIP laboratory at ENS Lyon. He is mainly interested in parallel algorithm design for heterogeneous platforms and in scheduling techniques.

Yves Robert received the PhD degree from the Institut National Polytechnique de Grenoble in 1986. He is currently a full professor in the Computer Science Laboratory LIP at ENS Lyon. He is the author of four books, 90 papers published in international journals, and 110 papers published in international conferences. His main research interests are scheduling techniques and parallel algorithms for clusters and grids. He is a senior member of IEEE and the IEEE Computer Society, and serves as an associate editor of *IEEE Transactions on Parallel and Distributed Systems*.



Yang Yang received the BE degree from Tsinghua University, P.R. China, and the MS degree from the University of California at San Diego in 2000 and 2003, respectively. He is a PhD student in the Computer Science and Engineering Department at the University of California, San Diego. His research interests are scheduling of parallel and distributed systems, including the scheduling of divisible load.