Chapitre 1

Modélisation de pipelines hétérogènes

O. Beaumont, V. Boudet, P. Ramet, J. Roman, C. Travers
(LaBRI ALiENor et ScAlApplix INRIA Futurs, Bordeaux)
F. Desprez (LIP et ReMaP INRIA Rhône-Alpes, Lyon)

1.1 Introduction

L'utilisation des machines parallèles apporte un gain important en performances et en taille mémoire mais amène en contre partie un surcoût en communications. Afin d'obtenir des programmes performants et extensibles, il convient de masquer ce surcoût. La solution consiste en un choix judicieux de la distribution des données qui réduira au maximum le nombre et la taille des communications. De plus, si les dépendances le permettent, il conviendra de découper les messages et d'initialiser les communications au plus tôt de manière asynchrone. Dans ce cas, une réduction du grain de calcul permet d'extraire du parallélisme.

Dans une architecture hétérogène ou hiérarchique (typiquement un cluster de SMP), on est amené à considérer le problème du recollement de pipelines présentant des grains de calcul différents.

L'étude des pipelines a fait l'objet de nombreuses recherches. Ils ont par exemple été modélisés à l'aide de réseaux de Pétri [5] et étudiés dans le cadre de compilation automatique [11]. Plusieurs articles abordent le problème du choix de la taille de grain optimal de manière statique [1, 9, 8] ou dynamique [6, 10]. Dans le cadre des grappes de SMP, les auteurs de [3] modélisent les exécutions d'applications par vagues dans le cas de pipelines synchrones sans changer, ni le grain de calcul, ni la distribution des données, à la différence des auteurs de [2] qui optimisent la distribution pour une application de différences finies. Un bibliothèque portable pour le support du parallélisme à grain fin a été presentée dans [7]. Enfin, dans [12], on trouve une modélisation de pipelines de communication dans le cas d'un réseau aux performances hétérogènes.

On se propose d'étudier la modélisation et la recherche des tailles de paquets qui minimisent le temps d'exécution d'un pipeline hétérogène. Dans la première partie, nous présentons quelques résultats d'optimalité. Les premiers résultats sont extraits de l'article de Wang et al [12] et concernent le problème de l'initialisation d'un pipeline (quand la taille du message à transférer est petite). Nous démontrons également un résultat d'optimalité asymptotique quand la taille du message est grande. Dans un seconde partie nous nous intéressons au problème particulier du pipeline sur une chaine de nœuds SMP.

1.2 Résultats dans le cas général

1.2.1 Optimalité

On considère le cas d'un message de taille B qui doit traverser un pipeline de n étages. Chaque étage j est caractérisé par une latence g_j, qui représente le coût minimal de transfert d'un paquet et par l'inverse de la bande passante du lien G_j, de sorte que le temps de transfert d'un paquet de taille x à travers l'étage j est égal à $g_j + xG_j$. Il est important de noter que cette modélisation des étages est très générale. En effet, si le transfert du paquet s'accompagne de calculs successifs sur ce paquet, il est possible de modéliser les étages de calcul par des étages tels que g_j est nul et G_j correspond à la vitesse de calcul d'un bloc (le fait de considérer une granularité, le bloc, permet de conserver une formulation linéaire). De la même façon, le terme G_j peut également représenter des caractéristiques de chargement et déchargement de la carte réseau vers la mémoire.

Le message à envoyer est de taille B. Nous considérerons dans un premier temps qu'il est partitionné en k messages de tailles $x_i,\ \ 1 \leq i \leq k$ avec $\sum_i x_i = B$, où les x_i et k restent à déterminer. On notera $\tau_{i,j}$ la date d'arrivée du paquet i à la fin de l'étape j.

1.2.1.a Solution sans fragmentation

Dans [12], les auteurs proposent une solution au problème général, dans le cas où agrégation et fragmentation de paquets ne sont pas considérées. Ils décrivent la solution optimale sous la forme d'un programme linéaire, qui peut être résolu en nombres rationnels en temps polynomial. Pour k fixé, on doit résoudre :

$$\text{MINIMISER} \quad \tau_{k,n},$$
$$\begin{cases} \forall\, 1 \le j \le k, & \tau_{1,j} \ge \sum_{h=1}^{j}(g_h + x_0 G_h) \\ \forall\, 1 \le i \le k,\ \ \forall\, 2 \le j \le n & \tau_{i,j} \ge \tau_{i,j-1} + (g_j + x_i G_j) \\ \forall\, 2 \le i \le k,\ \ \forall\, 2 \le j \le n & \tau_{i,j} \ge \tau_{i-1,j} + (g_j + x_i G_j)\ . \end{cases}$$

Le premier ensemble de contraintes décrit le comportement du premier paquet. Le deuxième ensemble de contraintes signifie qu'il s'écoule au moins $(g_j + x_i G_j)$ entre le passage du paquet i dans l'étage $j-1$ et sa sortie de l'étage j. Enfin, le dernier ensemble de contraintes signifie qu'il s'écoule au moins $(g_j + x_i G_j)$ entre le passage du paquet $i-1$ dans l'étage j et la sortie du paquet i.

Cette formulation est séduisante. Néanmoins, il ne semble pas aisé de décrire la fragmentation et l'agrégation de paquets à l'aide de contraintes linéaires. De ce fait, puisqu'on sait que de telles modifications des paquets sont parfois utiles, il est également intéressant de résoudre le problème analytiquement sur des exemples plus simples.

1.2.1.b Quelques cas particuliers

Des solutions dans des contextes particuliers sont proposés dans [12]. Nous citerons ici uniquement les résultats principaux, et nous invitons le lecteur à consulter cette référence pour y trouver les preuves. Le premier résultat concerne le cas particulier où on impose que les paquets sont tous de la même taille. Dans ce cas, il existe un étage (à déterminer) qui constitue le goulot d'étranglement. On peut facilement déterminer pour chaque intervalle de variation de x, la taille des paquets et l'étage b_x qui constitue le goulot d'étranglement.

Théorème 1 *Sur un intervalle où b_x constitue le goulot d'étranglement, la taille optimale des paquets est celle qui minimise*

$$T(x) = \frac{Bg_b}{x} + x\sum_{j\neq b} G_j + (\sum_{j\neq b} g_j + BG_b),$$

sur cet intervalle.

Pour déterminer la valeur optimale de la taille des paquets, il suffit donc de calculer la fonction précédente en au plus $3n$ points.

Si on supprime l'hypothèse de paquets de taille constante, la situation est beaucoup moins favorable. Le seul cas, dont on connaît la solution optimale indépendamment de toute hypothèse sur les g_i et les G_i, est le cas où le pipeline n'est constitué que de deux étages. On sait alors montrer que les deux étages doivent être utilisés en permanence pour assurer un débit optimal. Cette remarque conduit au théorème suivant, qui fournit la taille optimale des paquets x_i.

Théorème 2 *Les paquets doivent vérifier la relation de récurrence suivante :*

$$x_{i+1} = \frac{g_2 - g_1}{G_1} + x_i \frac{G_2}{G_1}, \quad \forall\, 1 \leq i \leq k.$$

Pour obtenir des résultats dans le cas de trois étages, il est nécessaire de pouvoir comparer les différents étages de façon absolue, i.e. indépendamment de la taille des paquets. Cette propriété est vérifiée en particulier quand :

$$\forall\, 1 \leq i, j \leq n, \quad g_i \leq g_j \Longleftrightarrow G_i \leq G_j.$$

On peut alors montrer le résultat suivant dans le cas d'un pipeline à trois étages (un lent, un rapide puis un autre lent), qui exprime le fait que les paquets doivent être de tailles croissantes puis décroissantes.

Théorème 3 *Dans la solution optimale, il existe un paquet c tel que :*

$$\begin{cases} x_{i+1} = \frac{g_2 - g_1}{G_1} + x_i \frac{G_2}{G_1}, & \forall\, 1 \leq i < c \\ x_{i+1} = \frac{g_3 - g_2}{G_2} + x_i \frac{G_3}{G_2}, & \forall\, c < i \leq k\ . \end{cases}$$

Dans le cas d'un pipeline à trois étages (un rapide, un lent puis un autre encore plus lent), il est démontré dans [12] que l'agrégation des paquets peut être utile pour compenser une latence importante du dernier étage. Malheureusement, il n'existe pas à notre connaissance d'algorithme permettant de déterminer la taille optimale des paquets.

Néanmoins, comme nous allons le démontrer dans le paragraphe suivant, il est possible de construire des algorithmes asymptotiquement optimaux pour la traversée d'un pipeline hétérogène, indépendamment de toute hypothèse sur la possibilité de diviser les paquets (fragmentation, agrégation), ou de comparer de façon absolue les étages. Ce résultat nous semble important, particulièrement quand la taille du message à envoyer est grande.

1.2.2 Pipeline périodique

Dans cette partie, nous cherchons un découpage périodique asymptotiquement optimal, y compris dans le cas où on suppose que les paquets peuvent être découpés et agrégés. Nous notons toujours (g_b, G_b), les valeurs respectives de la latence et de l'inverse de la bande passante de l'étage b qui vérifie $\forall i,\ G_b \geq G_i$ et B est la taille du paquet à transférer. De façon triviale, nous ne pouvons espérer faire mieux que $T^* = BG_b$ qui sera le temps minimal passé dans le goulot d'étranglement.

Nous considérons un ordonnancement des communications constitué de k périodes de durée T_p. La taille des paquets est constante, et la période T_p vérifie donc $\forall i, g_i + x \times G_i \leq T_p$, ce qui implique $x \leq \frac{T_p - g_i}{G_i}$. Nous choisissons la période T_p

suffisamment grande pour assurer que b est bien le goulot d'étranglement pendant une période. En prenant $x = \lfloor \frac{T_p - g_b}{G_b} \rfloor$, toutes les inégalités précédentes seront vérifiées dès que $T_p \geq \lceil \frac{g_b G_i - g_i G_b}{G_b - G_i} \rceil$.

En k périodes, nous allons traiter un volume de données au moins égal à $(k - n)(\frac{T_p - g_b}{G_b})$, puisque aucun message ne traverse l'étage n avant la $n^{ième}$ période. Pour assurer le transfert complet du message, il suffit donc de réaliser k périodes, où $k = \lceil \frac{BG_b}{T_p - g_b} + n \rceil$. Avec cette valeur de k, le temps nécessaire pour traiter le message est de $k \times T_p$, soit :

$$T = T_p \lceil \frac{BG_b}{T_p - g_b} + n \rceil \Longrightarrow T \leq \frac{BG_b T_p}{T_p - g_b} + (n+1) T_p \ .$$

Regardons l'évolution du rapport entre T et T^* soit $\frac{T}{T^*} = \frac{1}{1 - \frac{g_b}{T_p}} + 2n \frac{T_p}{T^*}$. Donc, si on choisit $T_p = \sqrt{T^*}$, on peut calculer le comportement asymptotique de T quand B, et donc T^*, tendent vers $+\infty$. On obtient finalement :

$$T = T^* + (g_b + n + 1)\sqrt{T^*} + o(\sqrt{T^*}).$$

On vérifie donc que la solution que nous proposons est asymptotiquement optimale, et que la vitesse de convergence est en $O\left(\frac{1}{\sqrt{T^*}}\right) = O\left(\frac{1}{\sqrt{B}}\right)$ ce qui achève la preuve du théorème suivant.

Théorème 4 *Il existe un pipeline périodique asymptotiquement optimal.*

1.3 Pipelines sur cluster de SMP

Un nœud SMP (*Shared Memory MultiProcessor*) est constitué de plusieurs processeurs qui partagent la même zone mémoire avec un espace d'adressage commun. Il est possible de relier plusieurs nœuds avec un réseau d'interconnection dans un cluster. Un cluster de nœuds SMP implique donc deux niveaux de communications avec des caractéristiques différentes qui amènent à considérer un parallélisme à grain fin à l'intérieur d'un nœud, et un parallélisme à grain moyen lorsque les communications mettent en jeu des nœuds différents. Le problème est donc celui du recollement de deux pipelines de granularités différentes. Après une présentation des notations utilisées, on présente dans cette section le calcul du temps total d'exécution du pipeline sur une chaîne de SMP, premièrement dans un cadre général, puis avec une hypothèse particulière sur le recollement.

1.3.1 Notations

On pose :
- n_{in} le nombre de processeurs par nœud et n_{out} le nombre de nœuds dans le cluster;
- B la taille des données à pipeliner;
- $L_{in}(x)$ et $L_{out}(x)$ les latences à l'envoi et à la réception d'un message de taille x entre deux processeurs internes ou externes à un nœud;
- $T_{in}(x)$ et $T_{out}(x)$ les temps de transmission d'un message de taille x entre deux processeurs internes ou externes à un nœud (par exemple $T_{in}(x) = g_{in} + xG_{in}$);
- $C(x)$ le temps de calcul d'un bloc de taille x;
- x_{in} et x_{out} les tailles de blocs interne et externe.

Pour évaluer le temps d'exécution, les temps intermédiaires intéressants sont :

- $t_{f,n}(k)$: temps sur l'étage n du pipeline au bout duquel le bloc k peut effectivement être transmis sur le réseau de communication. De fait, à l'instant suivant, l'étage n peut initialiser la réception du bloc $k+1$. A cet instant, sur l'étage n tout les blocs $j, j \leq k$ sont à jour et leur communication a été lancée.
- $t_{a,n}(k)$: le temps, sur l'étage n du pipeline, au bout duquel le bloc k est prêt à être reçu. En d'autres termes, à cet instant le bloc k a effectivement été transmis et est disponible dans un buffer interne sur le processeur exécutant l'étage n.

1.3.2 Expression du temps d'exécution total

1.3.2.a Temps d'exécution d'un pipeline sur un nœud

On commence par chercher le temps d'exécution du pipeline sur un nœud.

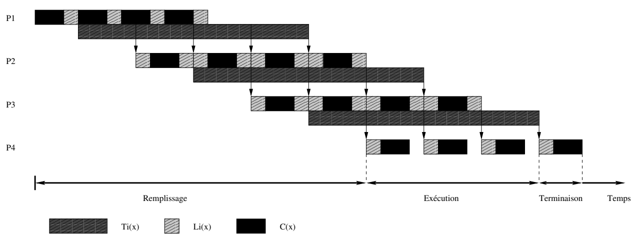

Figure 1.1 : Exécution pipelinée sur un nœud.

Pour un étage n du pipeline, pour un bloc k et pour une taille de bloc x_{in}, on a :

$$t_{a,n+1}(k) = \max\{t_{f,n}(k) + T_{in}(x_{in}), t_{f,n}(1) + k \cdot T_{in}(x_{in})\} .$$

Par récurence on trouve les expressions suivantes :

$$
\begin{aligned}
t_{f,n}(k) &= C(x_{in}) + L_{in}(x_{in}) + T_{in}(x_{in}) + (n-2)\cdot[C(x_{in}) + 2L_{in}(x_{in}) + T_{in}(x_{in})] \\
&\quad +(k-1)\cdot \max\{T_{in}(x_{in}), C(x_{in}) + 2L_{in}(x_{in})\} + 2L_{in}(x_{in}) + C(x_{in}) \ , \\
t_{a,n}(k) &= C(x_{in}) + L_{in}(x_{in}) + T_{in}(x_{in}) + (n-2)\cdot[C(x_{in}) + 2L_{in}(x_{in}) + T_{in}(x_{in})] \\
&\quad +(k-1)\cdot \max\{T_{in}(x_{in}), C(x_{in}) + 2L_{in}(x_{in})\} \ .
\end{aligned}
$$

Théorème 5 *Le temps total d'exécution d'un pipeline logiciel sur un nœud comportant n_{in} processeurs pour des données de taille B est :*

$$
\begin{aligned}
T(B) &= C(x_{in}) + L_{in}(x_{in}) + T_{in}(x_{in}) + (n_{in}-2)\cdot[C(x_{in}) + 2L_{in}(x_{in}) + T_{in}(x_{in})] \\
&\quad +(B/x_{in}-1)\cdot[\,\max\{T_{in}(x_{in}), C(x_{in}) + L_{in}(x_{in})\} + L_{in}(x_{in})] \\
&\quad +L_{in}(x_{in}) + C(x_{in}) \ .
\end{aligned}
$$

Le temps de l'application pipelinée est égal à la somme du temps de remplissage du pipeline, du temps d'exécution des blocs de calcul sans le premier, et du temps de terminaison du pipeline (c.f. fig.1.1) :

- Temps de remplissage: $C(x_{in}) + L_{in}(x_{in}) + T_{in}(x_{in}) + (n_{in} - 2)\cdot[C(x_{in}) + 2L_{in}(x_{in}) + T_{in}(x_{in})]$.
- Temps d'exécution: $(B/x_{in}-1)\cdot[\,\max\{T_{in}(x_{in}), C(x_{in}) + L_{in}(x_{in})\} + L_{in}(x_{in})]$.
- Terminaison: $L_{in}(x_{in}) + C(x_{in})$.

1.3.2.b Temps d'exécution sur une chaîne de SMP

On considère maintenant l'exécution du pipeline sur l'ensemble du cluster. La relation entre le rapport entre x_{in} et x_{out} , en dehors des recouvrements calculs/communications peut pénaliser le temps de remplissage de l'ensemble du pipeline. Pour étudier ce phénomène, on autorise dans le pipeline interne, une taille de bloc variable de façon à lancer au plus tôt le calcul sur l'ensemble des nœuds du cluster.

Une possibilité pour démarrer au plus tôt le calcul sur l'ensemble des nœuds est d'utiliser un pipeline interne sur une suite de blocs de taille variable telle que la somme des tailles soit égale à la taille de bloc externe x_{out}. Dans le cas général, on cherche une taille de bloc externe x_{out} et une suite $(x^i)_{i\in[1,n]}$ telle que $\sum x^i = x_{out}$ qui minimisent le temps total d'exécution de l'ensemble du pipeline.

On commence par examiner le déroulement d'un pipeline sur des blocs de taille variable sur un nœud. On simplifie le modèle en ne considérant plus que le temps de transmission et de calcul. Les temps de latence ne sont plus pris en compte. On considère donc que les temps de latence sont pris en compte dans les temps de calcul.

Soit donc une suite de n blocs $(x^i)_{i\in[1,n]}$. Le temps total d'exécution du pipeline sur deux processeurs est :

$$T = \max_{k=1}^{n} \max_{l=k}^{n} \Big\{ \sum_{i=1}^{k} C(x^i) + \sum_{i=k}^{l} T(x^i) + \sum_{i=l}^{n} C(x^i) \Big\} .$$

Cette expression est la valeur d'un chemin critique dans le pipeline. Un tel chemin débute par un certain nombre de calculs sur le premier étage suivis par plusieurs étapes de communications, et se terminent par des calculs sur le deuxième étage. Ce résultat peut être étendu à P processeurs. Une fois les décalages créés sur sur le deuxième étage, ils se répercutent sur la totalité des étages suivants (c.f. fig.1.2). Considérer le chemin critique sur les deux premiers étages suffit. On note k_{max} et l_{max} les deux entiers qui donnent le maximum de l'équation précédente.

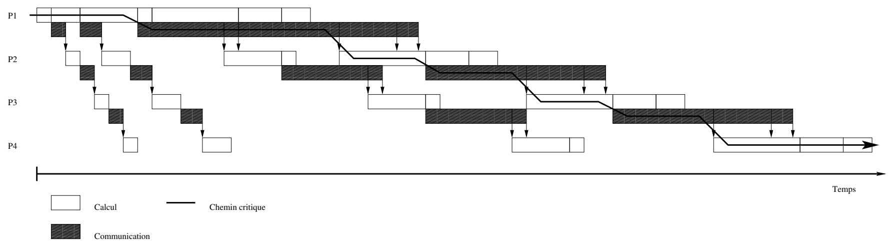

Figure 1.2 : Exécution pipelinée avec taille variable sur 4 processeurs.

Théorème 6 *Soit $(x^i)_{i\in[1,n]}$ la suite des tailles de blocs. Le temps total d'exécution du pipeline sur P processeurs est :*

$$T = \sum_{i=1}^{k_{max}} C(x^i) + \sum_{i=k_{max}}^{l_{max}} T(x^i) + (P-2)\cdot(C(x^{l_{max}}) + T(x^{l_{max}})) + \sum_{i=l_{max}}^{n} C(x^i) .$$

Si la taille de bloc externe est donnée, le problème est de déterminer le meilleur découpage des blocs internes. Pour une suite de tailles de blocs, on peut donc se demander comment ordonner les blocs pour minimiser le temps total.

Une heuristique généralement employée est d'ordonnancer d'abord les tâches de plus grand poids pour obtenir le plus grand recouvrement. Dans le cadre du pipeline, en supposant que les fonctions de coût des modèles de calcul et de communication sont croissantes, cet algorithme ordonne les blocs de taille x^i par ordre décroissant.

Pour poursuivre l'étude, on considère une hypothèse supplémentaire pour le recollement des pipelines internes et externes.

1.3.2.c Un cas particulier

On étudie le temps total d'exécution du pipeline sur l'ensemble du cluster dans la cas où une relation simple existe entre la taille de bloc interne et la taille de bloc externe. On suppose que *la taille de bloc externe est un multiple de la taille de bloc interne*, $x_{out} = \alpha . x_{in}$ avec α entier. Le problème se réduit donc a trouver x_{in} et α qui minimisent le temps total d'exécution.

Théorème 7 *Pour un pipeline sur n_{out} nœuds comportant chacun n_{in} processeurs, avec des tailles de blocs intra et extra–nœuds valant respectivement x_{in} et $\alpha . x_{in}$ sur des données de taille B, le temps total d'exécution est la somme de quatre termes (c.f. fig.1.3) :*

- *Remplissage "interne" :* $n_{out} \cdot (C(x_{in}) + L_{in}(x_{in}) + T_{in}(x_{in}) + (n_{in} - 2) \cdot [C(x_{in}) + 2L_{in}(x_{in}) + T_{in}(x_{in})])$.
- *Remplissage "externe" :* $n_{out} \cdot (\alpha - 1) \cdot M_1(x_{in}) + (n_{out} - 1) \cdot [C(x_{in}) + L_{in}(x_{in}) + 2L_{out}(\alpha . x_{in}) + T_{out}(\alpha . x_{in})]$.
- *Exécution :* $(B/\alpha - 1) \cdot M_2(x_{in}, x_{out})$.
- *Terminaison :* $L_{in}(x_{in}) + C(x_{in})$.

avec :

$$\begin{aligned} M_1(x_{in}) &= \max\{T_{in}(x_{in}), C_{in}(x_{in}) + 2L_{in}(x_{in})\} \ , \\ M_2(x_{in}, x_{out}) &= \max\{\alpha \cdot C(x_{in}) + L_{out}(x_{out}), T_{out}(x_{out}), \alpha \cdot (C(x_{in}) + L_{in}(x_{in})), \\ &\quad \alpha \cdot T_{in}(x_{in})\} \ . \end{aligned}$$

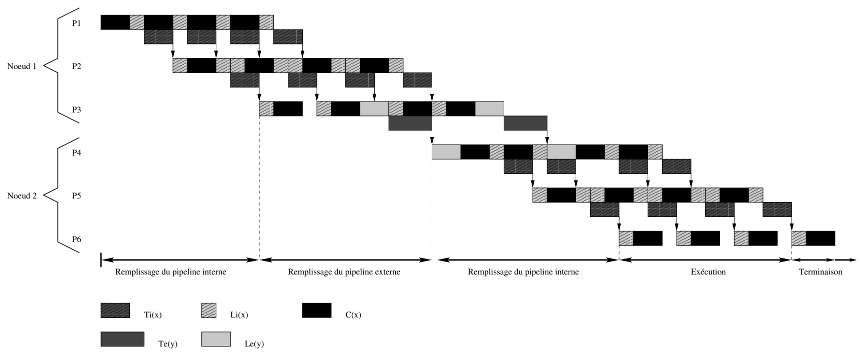

Figure 1.3 : Temps d'exécution d'un pipeline sur 2 nœuds.

Lemme 1 *Si on suppose que les fonctions de coût du modèle sont linéaires, on peut montrer qu'il existe une solution unique pour x_{in} et α.*

1.4 Conclusion

La modélisation des pipelines hétérogènes est un problème difficile dans le cas général, et cette étude algorithmique ne constitue qu'une première étape. Un autre objectif sera de valider expérimentalement ces résultats sur des applications cibles mettant en œuvre les techniques de pipeline.

Bibliographie

[1] F. Chaussumier. *Recouvrement des communications et des calculs, du matériel au logiciel.* PhD thesis, LIP, Ecole Normale Supérieure de Lyon, France, December 2001.

[2] S. J. Fink and S. Baden. Non-Uniform Partitioning for Finite Difference Methods Running on SMP Clusters.

[3] Adolfy Hoisie, Olaf M. Lubeckand Harvey J. Wasserman, Fabrizio Petrini, and Hank Alme. A general predictive performance model for wavefront algorithms on clusters of smps. In *ICPP*, page 219, 2000.

[4] J. Hurink and S. Knust. Flow–shop problems with transportation times and a single robot. Technical report, Osnabrücker Schriften zur Mathematik, 1998.

[5] C.-T. King, W.-W. Chou, and L.M. Ni. Pipelined data-parallel algorithms – concepts and modeling. In *International Conference on Supercomputing*, pages 385–395, juin 1988.

[6] David K. Lowenthal, Gregory R. Andrews, and Vincent W. Freeh. Using fine–grain threads and run–time decision making in parallel computing. Technical report, Departement of Computer Science, The University of Arizona, Dec 1995.

[7] David K. Lowenthal, Vincent W. Freeh, and Gregory R. Andrews. Efficient fine-grain parallelism on shared-memory machines. *Concurrency–Practice and Experience*, 10(3):157–173, 1998.

[8] D.J. Palermo. *Compiler Techniques for Optimizing Communications and Data-distribution for Distributed Memory Multicomputers.* PhD thesis, University of Illinois at Urbana-Champaign, 1996.

[9] P. Ramet, F. Desprez, and J. Roman. Optimal grain size computation for pipelined algorithms. In *Proceedings of Euro-Par'96, Lyon, France*, number 1123 in LNCS, pages 165–172. Springer Verlag, 1996.

[10] Bruce S. Siegell and Peter Steenkiste. Automatic selection of load balancing parameters using compile-time and run-time information. *Concurrency - Practice and Experience*, 4(9):215–317, 1997.

[11] C.-W. Tseng. *An Optimizing Fortran D Compiler for MIMD Distributed-M emory Machines.* PhD thesis, Rice University, January 1993.

[12] R. Y. Wang, A. Krishnamurthy, R. P. Martin, T. E. Anderson, and D. E. Culler. Modeling and optimizing communication pipelines. In *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems.* ACM, 1998.