RedGRID : un environnement pour la redistribution d'objets complexes

Aurélien Esnard*

* *Université de Bordeaux I, LaBRI et INRIA Futurs (projet ScAlApplix).*
351, cours de la Libération, 33405 Talence
aurelien.esnard@labri.fr

RÉSUMÉ. *Dans le cadre du couplage de codes, la redistribution efficace des données est un enjeu majeur pour obtenir de bonnes performances. Or la plupart des travaux dans ce domaine se limitent le plus souvent à l'étude d'objets simples avec des distributions régulières, comme par exemple des tableaux denses avec une distribution bloc-cyclique. Afin de prendre en compte les applications modernes du couplage de codes (simulations multiphysiques, environnements de pilotage), il s'avère important de définir un modèle de description pour des objets plus complexes que des tableaux multidimensionnels (*i.e. *grilles structurées, ensembles de particules ou maillages non structurés) et de proposer de nouveaux algorithmes de redistribution adaptés. Dans ce papier, nous présentons deux approches de la redistribution : l'approche spatiale et l'approche placement. Au final, ces algorithmes sont implantés dans la bibliothèque RedGRID et sont validés par des résultats expérimentaux.*

ABSTRACT. *In the context of code coupling, efficient data redistribution is a crucial issue to reach high-performances. However, most of the works in this area have limited their studies to simple objects with regular distribution patterns, such as multidimensional arrays distributed in a block-cyclic fashion. In order to deal with modern code-coupling applications (e.g. multiphysics simulations, computational steering environments), it becomes necessary to define a description model for more complex objects than multidimensional arrays (e.g. structured grids, particle sets or unstructured meshes) and to propose new redistribution algorithms based upon this model. In this paper, we present two approaches for data redistribution: the spatial approach and the placement approach. Finally, these algorithms are implemented in the RedGRID library and are validated by experimental results.*

MOTS-CLÉS : *Redistribution de données, couplage de codes, pilotage des simulations.*

KEYWORDS: *Data redistribution, code coupling, computational steering.*

RSTI - TSI – 27/2008. RenPar'06, pages 427 à 455

1. Introduction

Le problème de la redistribution des données a été beaucoup étudié dans le cadre de la programmation des architectures parallèles à mémoire distribuée, en particulier au titre des travaux menés autour de HPF (Koebel *et al.*, 1994) et de ScaLAPACK (Blackford *et al.*, 1997). Sur ce type d'architecture, comme par exemple une grappe de PCs, le placement des données sur les processeurs est de première importance. En effet, c'est ce placement qui conditionne à la fois l'efficacité des algorithmes parallèles et la réduction du surcoût des communications. Le problème de la redistribution survient dans ce contexte, car il peut s'avérer utile au cours des étapes de calcul d'un programme parallèle de redistribuer les données pour traiter plus efficacement la suite des calculs. Dans ce cas, la redistribution s'effectue « sur place », au sein du même code parallèle et chaque processeur envoie et reçoit des données. C'est par exemple le rôle de la directive *redistribute* dans le langage HPF. La redistribution des données est également primordiale dans le contexte du couplage de codes. Une application est alors vue comme un assemblage de plusieurs codes, le plus souvent parallèles, collaborant à la réalisation d'un travail commun (simulations multiphysiques). Ces codes sont typiquement distribués sur une grille de calcul, c'est-à-dire une grappe de calculateurs hétérogènes interconnectés par des réseaux plus ou moins performants. La communication entre les codes parallèles couplés nécessite de passer d'une distribution à une autre et donc de « redistribuer » les données (figure 1).

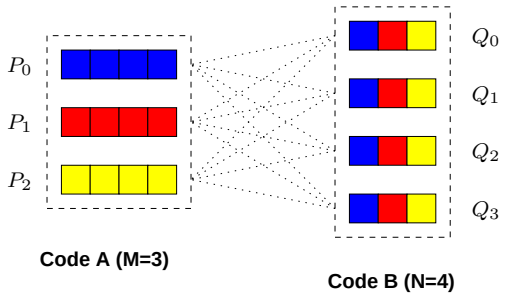

Figure 1. *Le problème de la redistribution des données entre deux codes couplés*

On décompose généralement le problème de la redistribution des données en quatre étapes consécutives : la description des données distribuées, la génération des messages, l'ordonnancement des communications et le transfert effectif des messages.

– *Description des données distribuées.* Le développement d'une solution générique au problème de la redistribution nécessite de définir un modèle de description également standard. Cette description doit spécifier d'une part, la distribution des données entre les processeurs, et, d'autre part, l'agencement des données en mémoire sur chaque processeur.

– *Génération des messages.* Le problème de la génération des messages consiste à déterminer, à partir des informations sur la distribution, les messages qui doivent être

échangés entre chaque couple de processeurs. L'ensemble des messages ainsi calculés forme ce qu'on appelle la matrice de communication.

– *Ordonnancement.* Une fois la matrice de communication calculée, il faut encore déterminer dans quel ordre effectuer les communications. En effet, les flux de communication parallèles, s'ils permettent d'agréger la bande-passante, peuvent entraîner des contentions réseaux qui dégradent les performances. Ainsi pour réduire le temps global de communication, il peut s'avérer utile de découper les messages afin de les ordonnancer en plusieurs étapes de communication.

– *Communication.* Lors de l'étape de communication, les codes couplés échangent l'ensemble des messages générés en respectant l'ordonnancement calculé. Le schéma de communication global s'apparente à un *all-to-all* personnalisé, transférant effectivement les données d'un code à l'autre.

Dans ce papier, nous nous intéressons au problème de la redistribution de données dans le contexte du couplage de codes, pour des couplages classiques du type simulation/simulation et pour des couplages plus spécifiques du type simulation/visualisation (problématique du pilotage). De tels couplages peuvent conduire à des redistributions de données plus irrégulières que celles étudiées classiquement en algèbre linéaire. Ainsi, nous cherchons à définir des algorithmes de redistribution généralistes pour des données complexes, telles qu'elles existent dans les applications scientifiques. Nous effectuons quelques hypothèses importantes pour la compréhension de cet article. Tout d'abord, nous supposons que la redistribution s'effectue sans transformation des données, à cause de discrétisations différentes par exemple. Par ailleurs, nous nous plaçons dans le cas d'un couplage « statique », c'est-à-dire pour lequel le choix de la distribution des codes n'est pas remis en cause au cours du temps.

Notre démarche s'articule en deux étapes principales. Premièrement, nous proposons un modèle de description des objets complexes (grilles structurées, ensembles de particules ou maillages non structurés) avec des distributions pouvant être irrégulières. A titre d'exemple, la simulation POP (Kerbyson *et al.*, 2005) représente la surface des océans avec une grille structurée, découpée par bloc et creuse à l'endroit des continents. Dans un autre domaine, la simulation NAMD (Kalé *et al.*, 1998) utilise une décomposition spatiale en boîte d'atomes pour représenter les molécules. Deuxièmement, nous proposons des algorithmes pour la génération des messages qui sont adaptés aux objets complexes décrits dans notre modèle. Nous n'aborderons pas ici le problème de l'ordonnancement des communications, pour lequel nous souhaitons reposer sur des résultats existants, en particulier (Jeannot *et al.*, 2004).

Les travaux présentés dans ce papier sont en partie motivés par le projet EPSN (Esnard *et al.*, 2004; Esnard *et al.*, 2006) qui cherche à définir un environnement pour le pilotage des simulations numériques, permettant de coupler une simulation parallèle avec un code de visualisation lui-même parallèle, ce qui nous appelons le problème du pilotage $M \times N$ (figure 2). Dans ce contexte, le code de visualisation se connecte dynamiquement au code de simulation pour produire une visualisation en ligne des résultats intermédiaires de la simulation. A la différence du couplage simulation/simulation, il faut noter que le code de visualisation n'impose pas de contraintes

a priori quant à la distribution de ses données, ce qui laisse un degré de liberté supplémentaire pour la génération des messages.

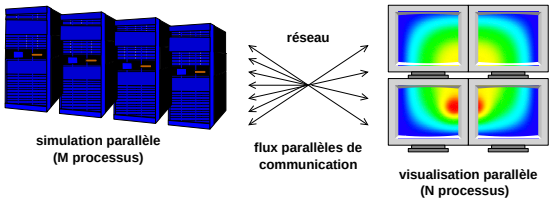

Figure 2. *Problématique du pilotage $M \times N$ dans EPSN (*`www.labri.fr/epsn`*)*

A la section suivante, nous présentons plusieurs travaux existants autour de la redistribution. Nous introduisons ensuite un modèle de description des données en termes « d'objet complexe » ainsi qu'un modèle de description des messages, appelé « message symbolique », qui permet de s'abstraire de la couche de communication utilisée (section 3). Puis, nous proposons à la section 4 deux approches de la redistribution : *l'approche spatiale* et *l'approche placement*, cette dernière étant bien adaptée au contexte du pilotage. Pour terminer, nous présentons brièvement à la section 5 la bibliothèque RedGRID implantant nos algorithmes et nous validons nos travaux en présentant quelques résultats expérimentaux dans la section 6.

2. Travaux existants

Le problème de la redistribution a été initialement étudié dans le contexte de l'algèbre linéaire sous l'impulsion de travaux comme HPF (Koebel *et al.*, 1994) ou ScaLAPACK (Blackford *et al.*, 1997). Dans ce contexte, il s'agit de passer d'une distribution bloc-cyclique d'un tableau dense à une autre distribution bloc-cyclique, ce qui conduit à un schéma de communication régulier (Prylli *et al.*, 1996; Thakur *et al.*, 1996; Walker *et al.*, 1996).

Dans le cadre du couplage de codes, la plupart des travaux existants comme PAWS (Beckman *et al.*, 1998), CUMULVS (Kohl *et al.*, 1997) ou CCA M×N (Bertrand *et al.*, 2005) traitent essentiellement de la redistribution de données régulières : *i.e.* des tableaux denses, multidimensionnels avec des distributions denses par blocs plus ou moins complexes. Si CUMULVS se restreint essentiellement à des distributions bloc-cycliques, PAWS quant à lui permet de redistribuer des tableaux utilisant une décomposition rectilinéaire d'un domaine global. Plus récemment, le projet CCA M×N cherche à définir une interface de redistribution standard pour les composants logiciels, en intégrant plusieurs technologies dont celles de PAWS et de CUMULVS. Tous ces travaux se limitent aux cas de données régulières pour des distributions relativement simples. Ainsi, il n'est pas possible avec ces logiciels de prendre en compte des structures creuses comme celles que nous avons décrites en introduction pour la simulation POP.

Concernant la redistribution de données irrégulières, on distingue deux types de travaux : d'une part, des travaux spécifiques comme MpCCI pour la redistribution de maillages non structurés ; et d'autre part des travaux plus généralistes basés sur le principe de linéarisation.

MpCCI (*Mesh based Parallel Code Coupling Interface* (Joppich *et al.*, 2006)) permet de coupler des applications parallèles à base de maillages. Le point fort de MpCCI réside dans le fait qu'il utilise des schémas d'interpolation permettant de transférer les données même lorsque les maillages ne sont pas identiques.

Les outils basés sur le principe de linéarisation, comme MetaChaos (Ranganathan *et al.*, 1996), InterComm (Lee *et al.*, 2004) ou MPI-I/O M×N (Bertrand *et al.*, 2003), apparaissent plus puissants, car ils peuvent s'appliquer de manière très générique à des structures de données *a priori* quelconques (e.g. arbres, graphes, maillages, etc.) Le principe de linéarisation consiste à ordonner totalement les éléments d'un objet « source » et d'un objet « destination », ce qui permet de mettre implicitement et indirectement en correspondance tous les éléments. Cependant, cette approche a un coût si l'on considère la taille des représentations intermédiaires utilisées pour décrire la distribution des éléments (au pire, du même ordre de grandeur que les données elles-mêmes). De plus, l'absence d'un modèle de description unifié ne permet pas d'assurer l'interopérabilité requise entre les codes couplés. En effet, si deux codes utilisent des linéarisations différentes, le résultat de la redistribution n'aura *a priori* aucun sens, ce qui est une limitation grave de cette approche. Cela est d'autant plus vrai que c'est à l'utilisateur qu'incombe la responsabilité de définir le schéma de linéarisation. Nous soulignons que ce problème est essentiellement lié au fait que Meta-Chaos comme InterComm ne proposent pas de modèle de représentation des objets, ni de schémas de linéarisation standards pour ces objets. Les structures de données sont simplement perçues comme des tableaux de données en mémoire, sans relation avec la nature réelle de ces objets.

Une limitation importante à tous ces travaux, mis à part CCA M×N, est liée au fait qu'ils ne dissocient pas la couche algorithmique de redistribution de la couche de communication, ce qui réduit considérablement la réutilisabilité de ces outils. Seul CCA M×N a choisi de clairement séparer à l'aide d'interfaces distinctes la description des données, la génération des messages et le transfert de ces messages.

3. Objets complexes et messages symboliques

Le modèle de description des données que nous proposons est basé sur la notion *d'objet complexe*. Ce modèle a pour objectif de prendre en compte une grande variété de structures de données présentes dans les codes de simulation numérique, tout en permettant une génération efficace des messages pour la redistribution. Dans ce modèle, les objets ne sont pas simplement vus comme des tableaux de données en mémoire ; ils encapsulent un ensemble d'informations utiles à leur redistribution ainsi qu'à leur visualisation. Ces dernières informations, comme par exemple la position

des éléments, sont particulièrement utiles dans le contexte du pilotage pour produire une représentation visuelle des objets considérés.

3.1. *Les objets complexes*

Considérons un ensemble d'éléments E totalement ordonné (ordre global). Un objet complexe $\mathcal{O}$ se présente comme une partie de E découpée en sous-ensembles appelées *régions* et composée de plusieurs *séries de données*. Chaque élément de l'objet possède une valeur dans chaque série de données, dont une fonction d'adressage sert à préciser l'adresse en mémoire. Lorsqu'un objet complexe est distribué sur un ensemble de processeurs P de taille M, cela revient à se donner une collection de M objets complexes associés à chaque processeur P_i et notés $\mathcal{O}_i$. L'objet $\mathcal{O}_i$ se compose alors d'un ensemble de régions, notées $R_i = (R_{i,0}, R_{i,1}, \ldots)$. Ainsi, chaque élément e d'indice global k dans E se trouve associé à une région $R_{i,r}$ d'indice r sur un processeur P_i et possède un indice local x dans cette région. On note $e = E[k] = R_{i,r}[x]$.

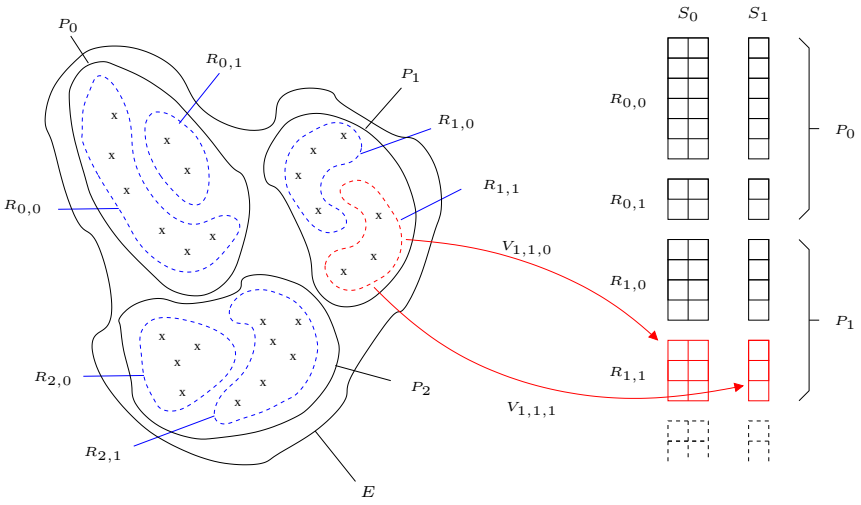

Figure 3. *Exemple d'un objet complexe distribué sur 3 processeurs, possédant deux séries de données (S_0 et S_1) et deux régions par processeur (notées $R_{i,r}$)*

La figure 3 donne un exemple d'objet complexe distribué sur 3 processeurs, possédant deux séries de données (S_0 et S_1). En pratique, les séries servent à décrire les variables associées aux éléments d'un objet (e.g. pression-température ; position-vitesse-force, etc.). Par la suite, l'utilisation des séries va permettre de factoriser le calcul de la redistribution pour toutes les séries d'un même objet. L'objet représenté sur la figure possèdent deux régions par processeur. Ces régions sont utiles pour regrouper des éléments connectés logiquement comme par exemple les composantes connexes d'un maillage ou les blocs d'éléments dans un tableau multidimensionnel.

Par ailleurs, la notion de région permet de considérer le problème de la redistribution avec une granularité moins fine que l'élément.

En outre, pour chaque région $R_{i,r}$ et chaque série S_s, une fonction d'adressage $v_{i,r,s}$ est définie sur les bases du modèle de stockage proposé à la section 3.3. Ce modèle permet de décrire l'agencement en mémoire des données telles qu'elles sont utilisés dans la plupart des codes, sans qu'il soit nécessaire d'adapter la structure de ces données.

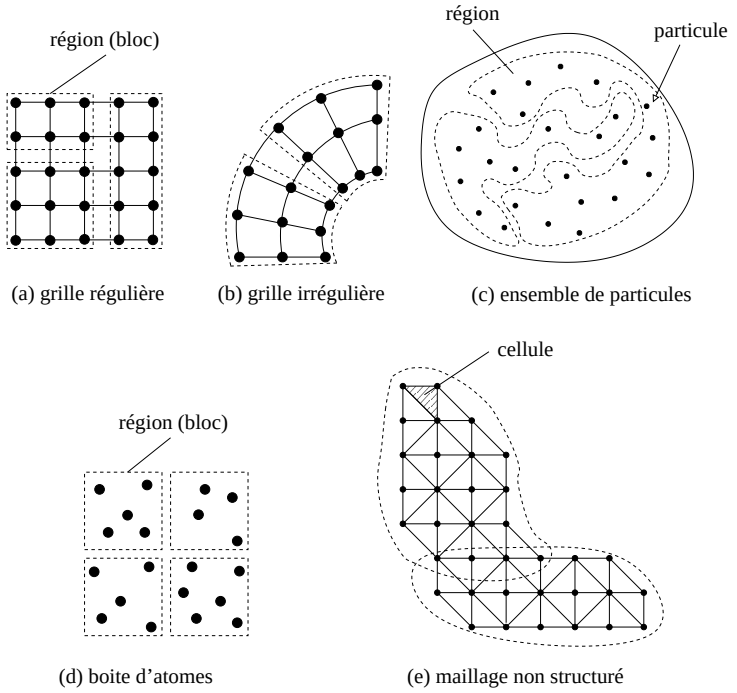

Figure 4. *Les différentes classes d'objets complexes*

3.2. *Les différentes classes d'objets*

Sur la base de ce modèle générique, nous proposons la définition de plusieurs classes d'objets complexes : les grilles structurées, les ensembles de particules et les maillages non structurés.

Tous ces objets entrent dans la famille des *objets spatiaux*. Par définition, un objet complexe $\mathcal{O}$ est dit spatial dans $\mathbb{R}^n$ si il existe une fonction coordonnée φ de E vers $\mathbb{R}^n$, telle que $\forall e \in E, \quad \varphi(e) = (x, y, ...)$ désigne la coordonnée spatiale de e. Les objets spatiaux sont très fréquents dans les codes de simulation numérique. Ils présentent un intérêt tout particulier dans le contexte du pilotage, car il est possible de

leur associer naturellement une représentation graphique dans un espace physique à n dimensions, et donc de les visualiser. Le nombre de dimensions n est généralement égal à 2 ou 3 mais peut atteindre 4 ou 6 dans certains codes manipulant l'espace des phases (position, vitesse) comme par exemple en physique des plasmas.

Les objets complexes tels que nous les avons définis sont découpés en régions logiques. Ces régions jouent un rôle important dans notre modèle car elles permettent de décrire la distribution des éléments par « paquet ». Selon la classe de l'objet considéré, les éléments et les régions prennent des significations différentes.

3.2.1. *Les grilles structurées*

Les grilles structurées (figures 4 (a) et (b)) sont une classe d'objet très répandue dans les codes de simulation en mécanique des fluides ou en sismique, servant le plus souvent à représenter une discrétisation de l'espace physique. Ces objets possèdent une topologie régulière, formée de quadrilatères en 2D et d'hexaèdres en 3D. D'un point de vue informatique, les grilles structurées correspondent à des tableaux de données multidimensionnels. Les éléments que nous considérons pour cette classe d'objet sont les points de la grille, auxquels sont associés les séries de données. Par ailleurs, les grilles structurées possèdent un système de coordonnées naturel $(i, j, \ldots)$ dans $\mathbb{Z}^n$ (espace topologique), permettant d'identifier de manière unique chaque point dans la grille. De plus, chaque point possède une coordonnée $(x, y, \ldots)$ dans l'espace physique $\mathbb{R}^n$. Si la grille est régulière (figure 4 (a)), cette dernière coordonnée peut être calculée simplement à partir de sa coordonnée topologique $(i, j, \ldots)$ en prenant en compte l'espacement inter-points selon toutes les directions de l'espace $(dx, dy, \ldots)$. En revanche, si la grille est irrégulière (figure 4 (b)), nous utilisons une série de données particulière de type $\mathbb{R}^n$, contenant les coordonnées spatiales de tous les points. Cette distinction entre grille régulière ou irrégulière se retrouve classiquement dans des outils de visualisation comme AVS (Upson *et al.*, 1989) ou VTK (Schroeder *et al.*, 2002).

Dans le cas des grilles structurées, les régions que nous considérons sont des *blocs* d'éléments. Plus formellement, un bloc B, noté $B = (a, b)$, est un parallélépipède rectangle de l'espace topologique $\mathbb{Z}^n$ décrit par deux points extrêmes $a, b \in \mathbb{Z}^n$. Ce bloc joue le rôle d'un conteneur pour les éléments de la grille. Ainsi, si l'on considère une liste de régions $R_i = (R_{i,0}, R_{i,1}, \ldots)$ affectées au processeur P_i, la distribution des éléments est explicitement définie par la liste des blocs $B_i = (B_{i,0}, B_{i,1}, \ldots)$ respectivement associés aux régions. Ce modèle de description par blocs – même s'il est moins compact que les modèles classiques « à la HPF » – s'avère très générique et très flexible pour représenter les distributions de données qui sont réellement présentes dans les codes de simulation. Ainsi nous ne nous limitons pas à des distributions denses rectilinéaires ou simplement bloc-cyliques, qui sont en fait des cas particuliers dans notre modèle.

3.2.2. *Les ensembles de particules et les boîtes d'atomes*

Les *ensembles de particules* (figure 4 (c)) sont des objets complexes qui servent à représenter des particules dispersées dans l'espace physique à n dimensions. Ces objets sont très fréquents en dynamique moléculaire ou en astrophysique. Une série de données particulière sert à décrire la position des particules dans l'espace. L'ensemble des particules formant une région est *a priori* quelconque et permet de regrouper des « paquets de particules » ayant le plus souvent une caractéristique commune dans la simulation, comme par exemple des particules chargées positivement ou négativement, ou encore différentes espèces chimiques.

Les *boîtes d'atomes* (figure 4 (d)) sont un cas particulier des ensembles de particules, qui associe à chaque région « logique » $R_{i,r}$ la description d'un bloc $B_{i,r}$ dans l'espace $\mathbb{R}^n$. Ce bloc joue simplement le rôle d'un conteneur pour les particules appartenant à cette région, ce que nous appelons également une boîte d'atomes par référence aux codes en dynamique moléculaire manipulant de telles *décompositions spatiales* (Plimpton *et al.*, 1995).

3.2.3. *Les maillages non structurés*

Les *maillages non structurés* (figure 4 (e)) sont des objets complexes représentant un ensemble de cellules géométriques dans l'espace physique à 2 ou 3 dimensions. Actuellement, nous ne prenons en compte que le cas de maillages formés d'un seul type de cellule : triangle, quadrilatère, hexaèdre ou tetraèdre. Les éléments de l'objet complexe (que nous considérons du point de vue de la redistribution) sont à proprement parler les cellules du maillage. Chaque région décrit une partie du maillage (typiquement une composante connexe) regroupant un ensemble de cellules reliant plusieurs nœuds. Chaque cellule est définie à l'aide d'une série de données particulière représentant la liste des connectivités. Cette liste contient les indices locaux des nœuds dans la région servant à former chaque élément. Les coordonnées des nœuds associés à la région sont représentées à l'aide d'une série de données particulière.

3.3. *Modèle de stockage*

Considérons un objet $\mathcal{O}$ constitué de s séries de données et défini dans un espace à n dimensions. Pour chaque série, il est nécessaire de décrire le stockage des données en mémoire, afin de pouvoir construire physiquement les messages calculés. Le modèle que nous utilisons est basé sur une approche classique en *stride* illustré par la figure 5.

Pour accéder physiquement à la valeur des éléments en mémoire, il faut se donner s fonctions d'adressage, une par série. Par définition, une fonction d'adressage $v : l \longrightarrow a$ associe à chaque élément d'indice local l dans une région R_i, r de P_i une adresse physique a dans la mémoire du processeur. Considérons un espace de stockage L à p dimensions, de taille $L_0 \times L_1 \times ... \times L_{p-1}$. A chaque dimension est associé un espacement ou *stride* (noté T_i, en octets) qui permet de passer à l'élément suivant

selon la i-ème direction de l'espace. La fonction d'adressage s'écrit alors : $a = A + \mathit{off} + c_0.T_0 + c_1.T_1 + ... + c_{p-1}.T_{p-1}$, où A est l'adresse de base, *off* l'offset initial et $(c_0, \ldots, c_{p-1})$ la coordonnée de stockage obtenue par simple division euclidienne de l par les dimensions de l'espace de stockage L. Ce modèle ne permet pas d'accéder à une séquence d'adresses mémoire quelconque, toutefois il est bien adapté à notre problématique, car il permet de prendre en compte une grande variété de structure de données (e.g. données continues, données filtrées, zones de recouvrement, etc.), tout en offrant un accès aléatoire à la mémoire en $\mathcal{O}(p)$ avec $p \simeq n$ dans le cas général et $p = 1$ si les données sont continues.

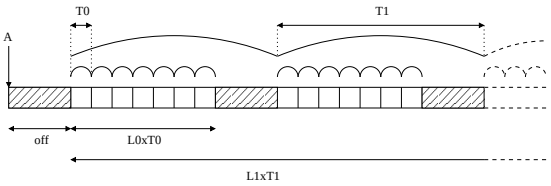

Figure 5. *Modèle de stockage mémoire en stride*

3.4. *Les messages symboliques et physiques*

Pour résoudre le problème de la redistribution, il faut calculer l'ensemble des messages échangés entre deux ensembles de processeurs P et Q (respectivement de taille M et N). Par définition, le message $m_{i,j}$ est l'ensemble des éléments qu'il faut échanger entre P_i et Q_j. Nous allons maintenant poser quelques définitions et introduire la notion de *message symbolique*, une représentation intermédiaire des messages que nous utilisons par la suite pour exprimer les messages indépendamment de l'espace de stockage et de la couche de communication. Un message symbolique $\hat{m}_{i,j}$ de P_i vers Q_j est une séquence de sous-messages, notée $(\hat{m}_{i,j,0}, \hat{m}_{i,j,1}, \hat{m}_{i,j,2}, \ldots)$, telle que le k-ème sous-message symbolique, $\hat{m}_{i,j,k}$, soit un triplet de la forme $(r_k, r'_k, \mathcal{M}_k)$ avec : r_k le numéro de la région source R_{i,r_k} de P_i, r'_k le numéro de la région cible R'_{j,r'_k} de Q_j, et $\mathcal{M}_k$ un masque d'extraction sur la région source. Le masque $\mathcal{M}_k$ est simplement défini par une séquence d'intervalles d'indices qui désigne la liste ordonnée des éléments de la région source relatifs au k-ème sous-message (figure 6).

Un message physique $\bar{m}_{i,j,s}$ de P_i vers Q_j désigne la séquence des adresses et des tailles de données représentant l'emplacement en mémoire des données de la série S_s à envoyer ou à recevoir, d'une manière compréhensible par une couche de communication. Notons que le message physique peut être généré très simplement à partir de la connaissance du message symbolique et de l'espace de stockage relatif à la série considérée.

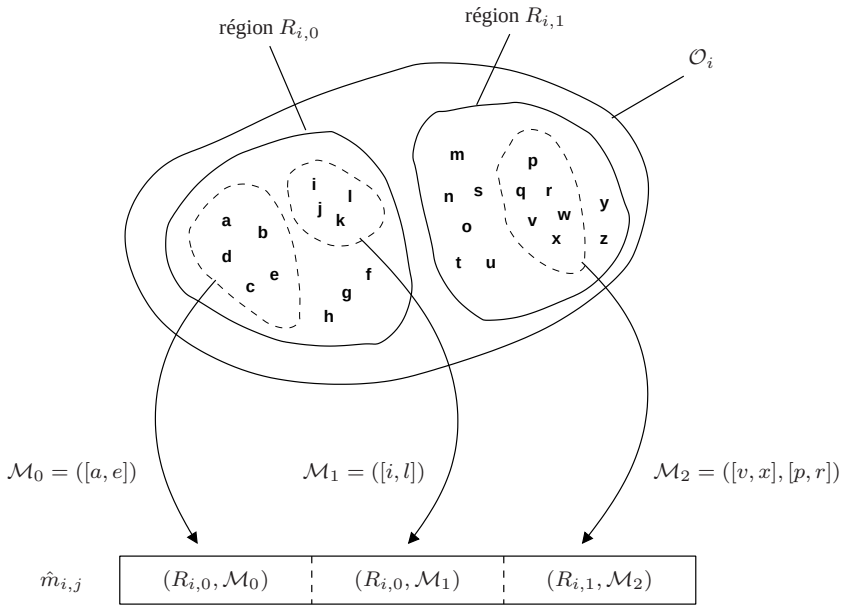

Figure 6. *Construction d'un message symbolique $\hat{m}_{i,j}$ à l'aide de masques d'extraction sur les deux régions de l'objet $\mathcal{O}_i$. Les lettres de l'alphabet désignent les éléments de l'objet (ordre lexicographique). L'utilisation des masques permet de désigner précisément les éléments que P_i doit échanger avec Q_j*

4. Algorithmes de redistribution

4.1. *Introduction*

Dans cette section, nous définissons un algorithme générique pour la redistribution d'objets complexes, tel que nous venons de les définir à la section précédente, et s'appuyant sur la notion de messages symboliques afin de s'abstraire entièrement de la couche de communication.

Considérons deux codes couplés A et B, respectivement distribués sur deux ensembles de processeurs P et Q, de tailles M et N. Soient $\mathcal{O}^A$ et $\mathcal{O}^B$ deux objets complexes respectivement distribués sur P et Q. On note $\mathcal{O}_i^A$ et $\mathcal{O}_j^B$ les objets complexes respectivement distribués sur chaque P_i et chaque Q_j. Le problème de la redistribution consiste typiquement à générer les messages qui vont permettre de passer de la distribution de $\mathcal{O}^A$ à celle de $\mathcal{O}^B$.

L'algorithme que nous proposons est *parallèle*, ce qui signifie que chaque P_i calcule uniquement une ligne de la matrice de communication $\mathbb{M}$, regroupant l'ensemble des messages $\{m_{i,j} \mid 0 \leq j < N\}$ qu'il doit transmettre à tous les Q_j. Par conséquent, il n'est en aucun cas nécessaire de centraliser la matrice de communication,

qui se trouve distribuée. Par ailleurs, notre algorithme est dit *symbolique* car il est indépendant d'une couche de communication donnée. Afin d'offrir le maximum de flexibilité vis-à-vis de la couche de communication, nous utilisons la représentation intermédiaire des messages, appelée *message symbolique*, notée $\hat{m}_{i,j}$ (cf. section 3.4). Cet algorithme se décompose en deux étapes principales, décrites sur la figure 7 : premièrement, le calcul des messages symboliques grâce à une opération abstraite *redistribute*, puis la génération des messages physiques à partir des messages symboliques grâce à l'opération *serialize*.

Algorithme 1
1 Pour chaque processeur P_i faire en parallèle
2 Pour chaque processeur distant Q_j faire
3 $\hat{m}_{i,j} \leftarrow$ *redistribute*$(\hat{\mathcal{O}}_i^A, \hat{\mathcal{O}}_j^B)$
4 Pour toutes les séries de rang s faire
5 $\bar{m}_{i,j,s} \leftarrow$ *serialize*$(\hat{m}_{i,j}, S_{i,s})$
6 Fin pour
7 Fin pour
8 Fin pour

Figure 7. *Algorithme parallèle pour la génération des messages sur le code A*

La définition concrète de l'opération *redistribute* va dépendre d'une part, de la classe de l'objet considérée, et, d'autre part, de l'approche choisie pour redistribuer les objets. Un message symbolique $\hat{m}_{i,j}$ est structuré en sous-messages logiques $\hat{m}_{i,j,k}$ construits à partir de masques d'extraction sur les régions. Ce masque désigne les éléments d'une région locale qui seront échangés avec une région distante. Ainsi posé, le problème de la redistribution consiste à calculer des masques d'extraction. Dans notre modèle, le message symbolique remplit deux fonctions primordiales : d'une part, il permet de désigner l'ensemble des éléments échangés entre chaque paire de processeurs, et d'autre part, il sert à définir un ordre sur les éléments du message, qui sera utilisé pour l'envoi et la réception des données. La génération des messages physiques à partir des messages symboliques est typiquement sous la responsabilité de la couche de communication, qui réalise une opération de sérialisation, notée *serialize*. Cette opération nécessite d'accéder à l'espace de stockage pour chaque série de données $S_{i,s}$. Dans la suite du papier, on se placera du point de vue du code A (processeurs P_i), qui sera dit *local*, par opposition au code B (processeurs Q_j) qui sera dit *distant*.

Nous allons maintenant présenter les deux approches que nous avons définies pour la redistribution des objets complexes : *l'approche spatiale* et *l'approche placement*.

4.2. *Approche spatiale de la redistribution*

Dans cette section, nous présentons un algorithme symétrique pour la redistribution des objets spatiaux distribués par blocs, comme les grilles structurées ou les ensembles de particules (boîtes d'atomes). Cette approche se base sur l'utilisation d'un

critère d'intersection « spatial » entre les blocs pour construire les messages à échanger entre les codes couplés.

Considérons un objet spatial distribué par blocs sur P_i et sur Q_j. Chaque processeur possède localement une liste de blocs, notés $B_i^A = (B_{i,0}^A, B_{i,1}^A, \ldots)$ pour P_i et $B_j^B = (B_{j,0}^B, B_{j,1}^B, \ldots)$ pour Q_j. On rappelle que dans ce contexte les régions correspondent à des *blocs* d'éléments. Plus précisément, les blocs vont jouer le rôle de « conteneur » pour les éléments, qui possèdent une coordonnée à l'intérieur du bloc. Ces ensembles de blocs sont tout à fait quelconques et servent à décrire la distribution des éléments dans l'espace. Dans le cas des grilles structurées, les blocs sont définis dans $\mathbb{Z}^n$, alors que pour les ensembles de particules, ils sont définis dans $\mathbb{R}^n$.

Afin d'illustrer notre approche, nous allons étudier dans cette section l'exemple de la figure 8. Considérons un domaine D dans un espace à 2 dimensions. Dans cet exemple, le processeur local P_i possède deux blocs $B_{i,0}^A$ et $B_{i,1}^A$; et le processeur distant Q_j possède également deux blocs $B_{j,0}^B$ et $B_{j,1}^B$.

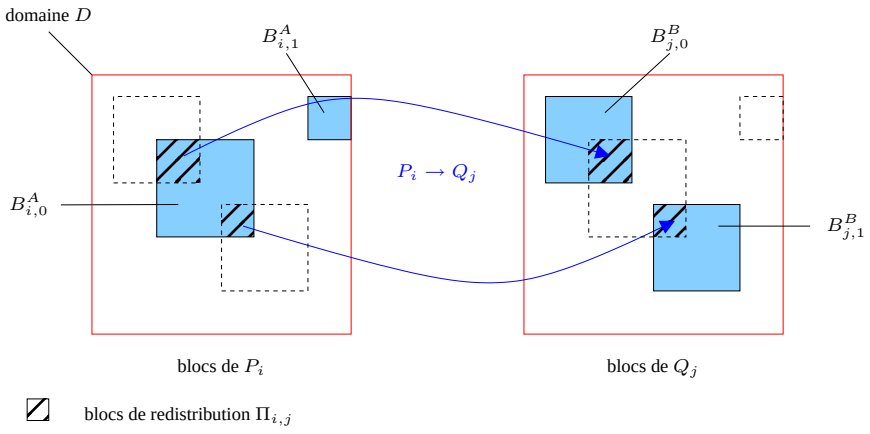

Figure 8. *Intersection géométrique des blocs de P_i et Q_j*

L'algorithme que nous proposons pour la génération des messages symboliques (figure 9) se décompose en trois étapes : premièrement, le calcul des blocs de redistribution $\Pi_{i,j,k}$ par intersection géométrique des blocs locaux $B_{i,r}^A$ et distants $B_{j,l}^B$ (ligne 4) ; puis le calcul du masque d'extraction $\mathcal{M}$ relatif au bloc de redistribution $\Pi_{i,j,k}$ grâce à l'opération *find* (ligne 6) ; et finalement la réorganisation des sous-messages symboliques dans un ordre canonique grâce à l'opération *sort* (ligne 12). Les éléments appartenant au message $m_{i,j}$ se trouvent à *l'intersection géométrique* des blocs de P_i et de Q_j dans l'espace $\mathbb{Z}^n$ ou $\mathbb{R}^n$ (figure 8). L'intersection entre le bloc local $B_{i,r}^A$ (en trait plein) et le bloc distant $B_{j,l}^B$ (en pointillé) forme ce qu'on appelle un *bloc de redistribution*, noté $\Pi_{i,j,k} = B_{i,r}^A \cap B_{j,l}^B$. On note $\Pi_{i,j}$ l'ensemble des blocs de redistribution non vides ($|\Pi_{i,j}| \leq |B_i^A| \times |B_j^B|$). A chaque bloc de redistri-

bution $\Pi_{i,j,k}$ non vide correspond un sous-message symbolique $\hat{m}_{i,j,k} = (r, l, \mathcal{M})$, dont il faut déterminer le masque $\mathcal{M}$. Ce masque désigne les éléments du bloc de redistribution $\Pi_{i,j,k}$ qu'il faut extraire du bloc local $B^A_{i,r}$.

Algorithme 2 (*redistribute*$(\hat{\mathcal{O}}^A_i, \hat{\mathcal{O}}^B_j)$)
1 $k \leftarrow 0$
2 Pour chaque bloc local $B^A_{i,r}$ de $\hat{\mathcal{O}}^A_i$ faire
3 Pour chaque bloc distant $B^B_{j,l}$ de $\hat{\mathcal{O}}^B_j$ faire
4 $\Pi_{i,j,k} \leftarrow B^A_{i,r} \cap B^B_{j,l}$
5 Si $\Pi_{i,j,k} \neq \emptyset$ alors
6 $\mathcal{M} \leftarrow$ *find*$(\Pi_{i,j,k}, B^A_{i,r})$
7 $\hat{m}_{i,j,k} \leftarrow (r, l, \mathcal{M})$
8 $k++$
9 Fin si
10 Fin pour
11 Fin pour
12 $\hat{m}_{i,j} \leftarrow$ *sort*$(\hat{m}_{i,j})$

Figure 9. *Approche spatiale : génération du message symbolique* $\hat{m}_{i,j}$

La recherche des éléments du bloc de redistribution $\Pi_{i,j,k}$ repose sur une opération abstraite ***find***, dont le rôle est de calculer le masque d'extraction $\mathcal{M}$. Ce masque définit un ordre sur les éléments au sein d'un sous-message, qui se base dans notre approche sur l'ordre local (naturel) de la région source, ce qui revient simplement à construire un masque strictement croissant. L'opération ***find*** est dite abstraite car sa définition va dépendre de la classe de l'objet considéré. D'une manière générale, on peut distinguer trois cas :

1) si l'intersection entre les blocs est nulle ($B^A_{i,r} \cap B^B_{j,l} = \emptyset$) alors le masque est vide ; cela veut dire qu'il n'y a aucun élément à échanger entre les régions correspondantes ;

2) si les deux blocs sont égaux ($B^A_{i,r} = B^B_{j,l}$), alors le masque est plein ; cela signifie que tous les éléments du bloc $B^A_{i,r}$ doivent être envoyés vers le bloc $B^B_{j,l}$ et réciproquement ;

3) si l'intersection entre les blocs est *partielle*, alors le calcul du masque va dépendre de la classe de l'objet considéré comme nous allons le voir (figures 10 et 11).

Dans le cas des grilles, la construction du masque dépend uniquement de la position relative du bloc de redistribution $\Pi_{i,j,k} = B^A_{i,r} \cap B^B_{j,l}$ par rapport au bloc local $B^A_{i,r}$ qui le contient (le *bloc source*). Soit $(l_0, l_1, \ldots, l_{n-1})$ les dimensions du bloc de redistribution. Au pire, le masque va donc contenir $l_1 \times \ldots \times l_{n-1}$ intervalles de taille l_0, un nombre qui est d'un ordre de grandeur inférieur au nombre total d'éléments dans la région ($l_0 \times l_1 \times \ldots \times l_{n-1}$). Au mieux, le masque contiendra un seul intervalle de taille $l_0 \times l_1 \times \ldots \times l_{n-1}$ (masque plein). Dans l'exemple de la figure 10, le masque permettant d'extraire le bloc de redistribution $B^A_{i,0} \cap B^B_{j,0} = ((6, 6), (10, 10))$ est formé

par un ensemble de $l_1 = 5$ intervalles $([0,4],[10,14],[20,24],[30,34],[40,44])$ de taille $l_0 = 5$.

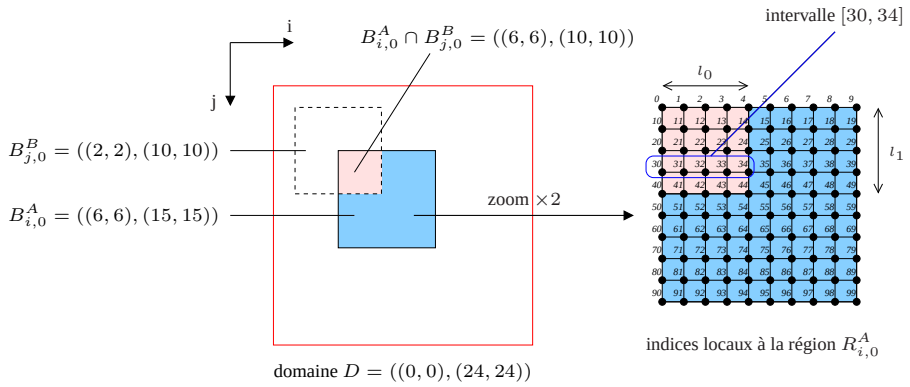

Figure 10. *Calcul du masque d'extraction associé au bloc de redistribution $B^A_{i,0} \cap B^B_{j,0}$ dans le cas des grilles structurées*

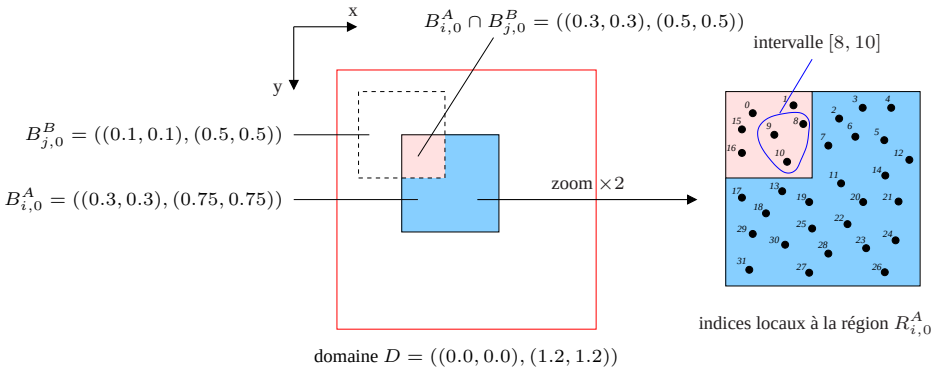

Figure 11. *Calcul du masque d'extraction associé au bloc de redistribution $B^A_{i,0} \cap B^B_{j,0}$ dans le cas des boîtes d'atomes*

Dans le cas des boîtes d'atomes, la construction du masque pour des blocs de redistribution résultant d'une intersection partielle est plus complexe. Elle nécessite de parcourir tous les éléments e du bloc local $B^A_{i,r}$ et de tester une à une leurs coordonnées pour déterminer lesquelles se trouvent à l'intérieur du bloc de redistribution : $\varphi(e) \in \Pi_{i,j,k}$. Dans l'exemple de la figure 11, le masque associé au bloc de redistribution $B^A_{i,0} \cap B^B_{j,0}$ est formé d'un ensemble de 3 intervalles $\{[0,1],[8,10],[15,16]\}$. Contrairement aux grilles structurées, il n'est pas possible de déterminer analytiquement le nombre d'intervalles dans le masque, ni la taille de ces intervalles. On suppose seulement que la taille des intervalles d'éléments continus ne sera pas trop petite si la position des atomes dans la boîte est plus ou moins corrélée à leurs indices dans la région. Au pire, le nombre d'intervalles est égal au nombre d'atomes dans le bloc de

redistribution (*i.e.* intervalles de taille 1). Notons que lorsque les distributions considérées sont à grain suffisamment fin (cas d'un pavage de l'espace), les intersections partielles entre blocs locaux et distants sont plus rares. Dans ce cas, le calcul des masques devient trivial, dans le sens où il suffit de transférer des boîtes d'atomes entières, sans qu'il soit nécessaire de trier les atomes à l'intérieur d'une boîte (masque formé d'un seul intervalle englobant tous les éléments).

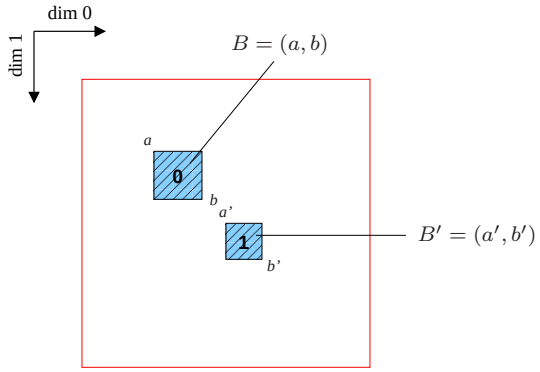

Figure 12. *Opération* sort *: tri des blocs de redistributions B et B'*

Afin d'obtenir un message canonique, il faut encore ordonner les sous-messages symboliques entre eux dans un ordre identique pour P_i et Q_j. C'est précisément le rôle de l'opération *sort* dans notre algorithme. Ce tri des sous-messages peut s'effectuer très simplement si l'on dispose d'un relation d'ordre sur les blocs de redistribution associés à chaque sous-message. La définition d'une relation d'ordre sur les blocs de $\mathbb{Z}^n$ (resp. $\mathbb{R}^n$) peut s'effectuer simplement en comparant la position des points extrêmes relatifs aux blocs[1]. Considérons par exemple deux blocs $B = (a, b)$ et $B' = (a', b')$. Par définition, on a $B < B'$ si et seulement si : $a < a'$ ou si $a = a'$ alors $b < b'$. La figure 12 illustre ce propos : la coordonnée de a sur la dimension d'indice 1 (dimension de poids fort) est strictement inférieure à celle de b. Par conséquent, le point a est strictement inférieur à b, et donc le bloc B est strictement inférieur au bloc B'. Ainsi, il est possible de trier les blocs de redistribution dans un ordre canonique, c'est-à-dire identique pour P_i et Q_j et donc de constituer un message symbolique bien ordonné, prêt pour la sérialisation.

Du point de vue de la complexité algorithmique, l'étape prépondérante est le calcul des masques par l'opération *find*. Le nombre de masques qu'il faut effectivement calculer est égal au nombre de blocs de redistribution non vides $|\Pi_{i,j}|$. Ce nombre peut être majoré pour chaque P_i par le produit du nombre de blocs locaux et du nombre total de blocs distants, c'est-à-dire par $|B_i^A| \times \sum_{j=0}^{N} |B_j^B|$. La complexité de cette opération dépend de la classe de l'objet considéré. Dans la cas des grilles structurées, ce

1. On considère la relation d'ordre total suivante sur les points de l'espace $\mathbb{Z}^n$ (resp. $\mathbb{R}^n$). Soient $p = (p_0, \ldots, p_{n-1})$ et $p' = (p'_0, \ldots, p'_{n-1})$ deux points de $\mathbb{Z}^n$ (resp. $\mathbb{R}^n$). Par définition, on a $p < p'$ si et seulement si il existe une dimension $c < n$ telle que $p_c < p'_c$ et $\forall i < c,\ p_i = p'_i$.

calcul est simplement linéaire par rapport au nombre d'intervalles dans le masque, un nombre qui au pire est d'un ordre de grandeur inférieur au nombre d'éléments dans la région (*i.e.* complexité au pire sous-linéaire par rapport au nombre d'éléments). Dans le cas des boîtes d'atomes, ce calcul a une complexité au pire linéaire par rapport au nombre d'éléments dans la région. En pratique, cette complexité sera d'autant plus petite que les distributions des deux codes seront proches (présence de blocs identiques entre ces codes).

4.3. *Approche placement de la redistribution*

Dans cette section, nous proposons une nouvelle approche de la redistribution, baptisée *l'approche placement*, s'appliquant plus spécifiquement dans le contexte du pilotage $M \times N$. On supposera alors que le code A joue le rôle d'une simulation parallèle sur M processeurs et B celui d'un programme de visualisation parallèle sur N processeurs (en général, $M \gg N$). Dans le cadre du pilotage $M \times N$, seul le code de simulation (code A) possède initialement un objet complexe $\mathcal{O}^A$ distribué sur P. Côté visualisation, le code B n'a *a priori* aucune information sur l'objet distant qu'il souhaite visualiser. L'objet $\mathcal{O}^B$ est ainsi « vierge » de toute distribution. Il faut alors choisir cette distribution à l'initialisation du couplage, à partir des informations relatives à la distribution de l'objet distant $\mathcal{O}^A$. En pratique, l'acquisition de ces informations implique une étape de communication préliminaire entre chaque paire de processeurs (P_i, Q_j). On peut alors formuler le problème de la redistribution comme un problème plus simple s'apparentant à un *problème de placement* des éléments d'un code vers l'autre. Dans ce cas, la distribution des éléments pour l'objet vierge peut être choisie « au mieux » afin de faciliter le couplage et réduire le coût des communications inter-codes[2]. Nous avons choisi d'appliquer cette approche à des objets complexes non structurés, comme les ensembles de particules ou les maillages non structurés. Nous proposons deux variantes de cette approche : l'approche *placement sans découpage* des régions et l'approche *placement avec découpage*. Pour chacune de ces deux approches, le problème de la redistribution va principalement consister à calculer les messages symboliques $\hat{m}_{i,j}$ de P_i vers Q_j, et plus précisément à construire les masques d'extraction sur les « régions sources ».

4.3.1. *Placement sans découpage*

Cette stratégie de redistribution très simple consiste à placer des régions entières (une ou plusieurs) du processeur P_i vers le processeur Q_j. Le message symbolique $\hat{m}_{i,j}$ se compose d'autant de sous-messages qu'il y de régions à échanger entre P_i et Q_j. Tous les éléments de la région $R^A_{i,r}$ appartiennent au sous-message

2. Il est important de noter que les algorithmes parallèles de visualisation/rendu que nous visons (*i.e.* sort-last) fonctionnent quelle que soit la distribution des données. Dans cette approche, un processeur Q_j peut alors contribuer à n'importe quelle partie de l'image (contrairement aux algorithmes de type sort-first). Cependant, une bonne distribution – même si cela reste difficile à formaliser – doit être équilibrée et favorisée une certaine localité.

$\hat{m}_{i,j,k} = (r, l, \mathcal{M})$ destiné à la région $R^B_{j,l}$ avec le masque $\mathcal{M} = ([0, |R^A_{i,r}| - 1])$. La construction d'un message symbolique canonique nécessite encore de trier les sous-messages $\hat{m}_{i,j,k}$ dans un ordre identique pour P_i et Q_j. Pour ce faire, nous utilisons simplement l'ordre croissant sur les numéros de région dans le code A.

Le placement des régions du code A sur le code B peut être fixé explicitement par l'utilisateur à partir de la description de $\mathcal{O}^A$ ou être calculé automatiquement par *une fonction de placement*. Nous supposons que le nombre d'éléments dans les régions de $\mathcal{O}^A$ est globalement équilibré, ce qui est généralement le cas pour la plupart des codes de simulations numériques. Sous cette hypothèse, il est possible de définir une fonction de placement très simple ne dépendant que de M, de N et du nombre de régions $|R^A_i|$ associé à chaque P_i. Soient $|R^A|$ le nombre total de régions et z le reste de la division entière de $|R^A|$ par N. Le nombre de régions $|R^B_j|$ affectés à Q_j est $\lceil |R^A|/N \rceil$ si $j < z$ et $\lfloor |R^A|/N \rfloor$ sinon. Si $|R^A|$ est divisible par N, tous les Q_j possèdent exactement le même nombre de régions, sinon il diffère au pire de 1. Dans l'exemple de la figure 13, le code A est distribué sur $M = 4$ processeurs avec 2 régions par processeur, et B est distribué sur $N = 3$ processeurs. Dans ce cas, nous choisissons de placer les $|R^A| = 8$ régions de la façon suivante : 3 régions sur Q_0 , 3 régions sur Q_1 et 2 régions sur Q_2.

En conclusion, il faut souligner que cette stratégie de redistribution est très simple à mettre en œuvre et peut s'appliquer de manière très générale à tout type d'objet complexe, et pas uniquement aux ensembles de particules ou aux maillages non structurés. En revanche, cette approche peut conduire à un mauvais équilibrage de la charge du code B si les régions du code A ne sont pas à grain suffisamment fin par rapport au nombre total d'éléments.

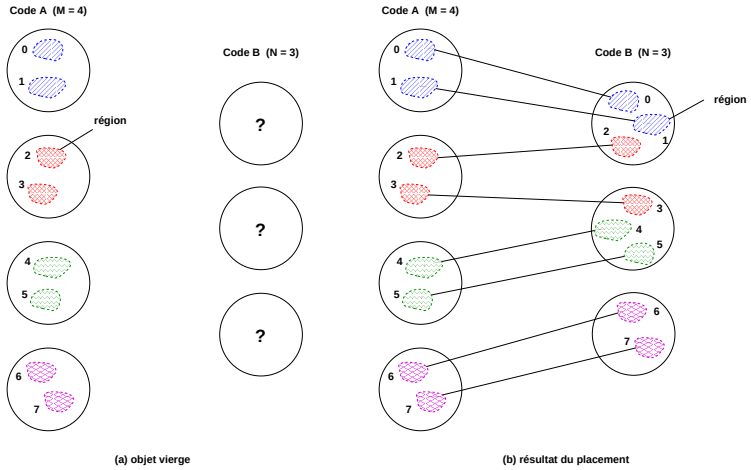

Figure 13. *Exemple de placement de A vers B sans découpage des régions*

4.3.2. *Placement avec découpage*

Afin de surmonter les difficultés de la méthode précédente, on autorise le découpage des régions pour ajuster la taille des messages échangés entre A et B. Nous nous plaçons dans le cas où il n'y a qu'une région par processeur notée $R^A_{i,0}$ sur P_i (resp. $R^B_{j,0}$ sur Q_j). De plus, nous supposerons que le nombre d'éléments dans chaque région de P_i est identique et égal à m. Notons qu'il est possible de généraliser cet algorithme sans trop de difficultés au cas où le nombre de régions par processeur est supérieur à 1 et au cas où les tailles des régions sont différentes.

Cet algorithme repose sur la définition de l'opération *split*, une opération abstraite qui permet de découper une région en n parties de poids $(p_0, p_1, \ldots, p_{n-1})$, tels que $p_k > 0$ et $\sum_{k=0}^{n-1} p_k = 1$. La k-ème partie est définie par un masque d'extraction $\mathcal{M}_k$ sur la région, désignant un ensemble de $p_k.m$ éléments. L'opération *split* permet de calculer les masques d'extraction servant à générer les messages symboliques. Dans le cas des ensembles de particules, il est possible de considérer un découpage en n intervalles d'éléments successifs (selon l'ordre local de la région). En revanche, le découpage d'un maillage non structuré en n parties de poids différents nécessite d'utiliser un algorithme de partitionnement, comme ceux implantés dans Scotch (Pellegrini, 1997) ou Metis (Karypis *et al.*, 1995). Notons que le partitionnement des cellules d'un maillage induit implicitement un partitionnement des nœuds portées par les cellules. Par conséquent, nous manipulerons en plus du masque classique sur les éléments (*i.e.* les cellules), un masque sur les nœuds simplement calculé à partir de ce dernier.

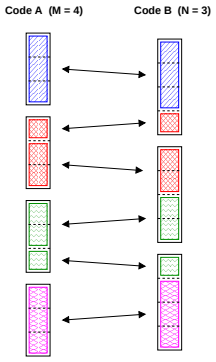

Figure 14. *Exemple de placement de A vers B avec découpage*

Le calcul d'un placement des éléments de A vers B se base sur un découpage en $M \times N$ unités logiques de l'objet $\mathcal{O}^A$ et de l'objet distant $\mathcal{O}^B$. Les unités logiques sont simplement une abstraction qui va nous permettre de calculer plus facilement les poids servant au découpage des régions. Chaque région locale $R^A_{i,0}$ est décomposée

en N unités logiques et chaque région distante $R^B_{j,0}$ est décomposée en M unités logiques. On note $u^A_{i,k}$ la k-ème unité logique de la région locale $R^A_{i,0}$ ($0 \leq k < N$) et $u^B_{j,k}$ la k-ème unité logique de la région distante $R^B_{j,0}$ ($0 \leq k < M$). La figure 14 illustre un découpage logique en 12 unités logiques ($M = 4$ et $N = 3$) et leur placement sur les processeurs distants selon notre algorithme. Les unités logiques sont numérotées par ordre de processeurs croissants de 0 à $M.N - 1$ et chaque unité locale $u^A_{i,k}$ d'indice global $i \times N + k$ est associée à l'unité distante $u^B_{j,k'}$ de même indice, c'est-à-dire telle que $i \times N + k = j \times M + k'$. Plus précisément, si l'on note $U^A_i = [i.N, (i+1).N - 1]$ l'intervalle des indices globaux relatifs aux unités logiques de P_i et de même pour l'intervalle $U^B_j = [j.M, (j+1).M - 1]$ sur Q_j, alors le message symbolique $\hat{m}_{i,j}$ est constitué des unités logiques dont l'indice global est à l'intersection de ces deux intervalles : $U^A_i \cap U^B_j$. Lorsque l'intersection est nulle, il n'y a aucun message échangé entre P_i et Q_j. Il est facile de voir que chaque P_i envoie au moins 1 message aux processeurs de Q et au plus 2 messages. On peut démontrer que le nombre total de messages échangés est alors de $M + N - \text{PGCD}(M, N)$. Notons que dans le cas particulier où M et N sont des multiples (en supposant $M \geq N$) alors chaque P_i envoie exactement 1 message de N unités logiques consécutives (*i.e.* pas de découpage) et chaque Q_j en reçoit exactement M/N. Dans l'exemple de la figure 14, cette stratégie de placement génère un total de 6 messages pour $M = 4$ et $N = 3$, tels que $U^A_0 \cap U^B_0 = [0, 2], U^A_1 \cap U^B_0 = [3, 3], U^A_1 \cap U^B_1 = [4, 5], U^A_2 \cap U^B_1 = [6, 7], U^A_2 \cap U^B_2 = [8, 8]$ et $U^A_3 \cap U^B_2 = [9, 11]$. Les poids relatifs au découpage des régions des processeurs de P sont alors respectivement $(3/3)$ pour P_0, $(1/3, 2/3)$ pour P_1, $(2/3, 1/3)$ pour P_2 et $(3/3)$ pour P_3.

D'une manière générale, la réception des messages pour Q_j se base sur l'opération abstraite *merge*, dont le rôle est de fusionner plusieurs messages à destination d'une même région $R^A_{j,0}$. Dans le cas des ensembles de particules, cette opération correspond simplement à une accumulation des données reçues, message par message, dans un ordre bien déterminé pour conserver les éléments de la région $R^B_{j,0}$ dans un ordre identique d'une réception à l'autre. Dans le cas des maillages non structurés, nous avons dû adapté ce mécanisme pour prendre également en compte la série des connectivités. En effet, les indices des nœuds dans la série des connectivités font référence aux indices locaux dans la région $R^A_{i,0}$. Par conséquent, ces indices doivent être convertis en une nouvelle numérotation locale à la région $R^B_{j,0}$. Cette conversion s'effectue en deux étapes, une première qui consiste à convertir les indices des nœuds reçus en une numérotation interne au message de rang k s'étendant de 0 à $n_k - 1$, avec n_k le nombre total de nœuds utilisés dans ce message. Notons que cette nouvelle numérotation doit respecter l'ordre croissant des indices de nœuds définis dans la région source. La deuxième étape de la conversion consiste simplement à décaler les indices des nœuds de $d = \sum_{k'=0}^{k-1} n_{k'}$ pour prendre en compte l'accumulation des nœuds dans la région $R^B_{j,0}$ relative aux $k - 1$ messages précédents.

5. Bibliothèque RedGRID

Afin de valider notre approche, nous avons développé un environnement logiciel pour le couplage de codes parallèles et plus précisément pour la redistribution des objets complexes. Cette bibliothèque se décompose en deux couches logicielles, appelées RedSYM et RedCORBA[3]. La bibliothèque RedGRID est programmé en C++ et dispose également d'une API C et Fortran, des langages très répandus dans le domaine du calcul scientifique.

5.1. *RedSYM : la couche de redistribution symbolique*

La bibliothèque RedSYM implante le modèle de description des données et les algorithmes présentés dans ce papier. Plus précisément, RedSYM calcule (en parallèle) la matrice de communication à partir de la description des objets complexes. Cette bibliothèque est dite *symbolique* dans le sens où son interface est clairement indépendante d'une couche de communication particulière. Dans RedSYM, les messages générés sont uniquement stockés de manière symbolique sous la forme d'une séquence d'intervalles d'indices désignant la liste ordonnée des éléments formant un message. Associé au modèle de stockage d'une série de données, le message symbolique permet de reconstruire efficacement la séquence des adresses mémoire constituant le message physique.

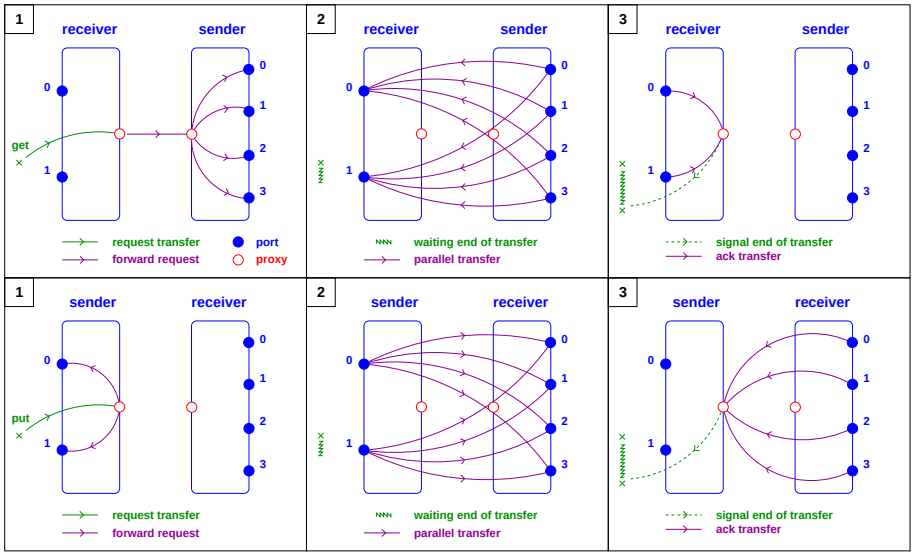

Figure 15. *RedCORBA : schéma des requêtes* get *(en haut) et* put *(en bas)*

3. Ces travaux ont été réalisés dans le cadre de l'ACI GRID EPSN et de l'ARC RedGRID.

5.2. *RedCORBA : la couche de communication*

La bibliothèque RedCORBA implante une couche de communication au-dessus de RedSYM, basée sur la technologie CORBA. Elle permet de coupler dynamiquement plusieurs codes parallèles (e.g. MPI, PVM, etc.) sur un réseau hétérogène. Elle prend en charge l'échange des descriptions entre les codes couplés, et le transfert parallèle des données d'un code (*sender*) à l'autre (*receiver*). RedCORBA utilise une approche composant, comparable à celle mise en œuvre dans PAWS (Beckman *et al.*, 1998), basée sur la notion de *ports* de communication et de *data channel*. Par ailleurs, un *proxy* permet de coordonner les transferts via un système de requêtes *get/put*, représenté sur la figure 15. Ce système implique une gestion centralisée des requêtes et des acquittements, alors que le transfert s'effectue toujours en point-à-point entre les ports. Afin de traiter plus efficacement les transferts successifs, nous stockons la matrice de communication symbolique calculée par RedSYM, matrice qui est factorisée pour toutes les séries de données. Les messages CORBA générés par RedCORBA sont également stockés sous la forme d'une séquence d'adresses mémoire. Ainsi la matrice de communication est calculée une seule fois, lors de l'étape de connexion des codes couplés et elle est ensuite conservée durant toutes les étapes de communication. Cette matrice devra être remise à jour uniquement dans le cas où la distribution des données venait à changer.

Le transfert des messages repose sur une stratégie zéro-copie n'imposant pas de copie des données à l'envoi. Cependant, une copie des données s'avère nécessaire dans RedCORBA au moment de la réception. En effet, le paradigme de communication utilisé par CORBA ne permet pas d'effectuer une réception « sur place » des données qui sont passées comme les arguments d'un appel de méthode à distance, ce qui nous impose une recopie mémoire dans RedCORBA. Cette limitation ne remet pas en cause le choix de la technologie CORBA qui s'avère à notre sens le middleware de prédilection pour le couplage de codes assurant à la fois la portabilité, l'interopérabilité et la dynamicité requise[4] pour ce type d'application, ce que MPI-1 par exemple ne pourrait pas permettre.

6. Résultats

Nous allons maintenant consacrer une section à l'évaluation des performances de la bibliothèque RedGRID et plus précisément de la couche de communication RedCORBA. Pour cette étude, nous supposons que le couplage est statique, dans le sens où le choix des distributions ne sont pas remis en cause au cours du temps. Dans ce contexte, la génération des messages est une opération ponctuelle survenant à l'initialisation du couplage, alors que les transferts sont répétés continuellement, le plus souvent à chaque itération du code. C'est pourquoi nous allons principalement étudier les temps de transfert et plus particulièrement les débits entre les codes couplés. Nous réservons pour plus tard l'étude des performances de l'algorithme de redistribution à

4. Connexion client/serveur des codes couplés notamment dans le contexte du pilotage.

proprement parler, étude devenant primordiale dès lors que la distribution des données est susceptible de changer fréquemment (couplage dynamique).

Dans ces expériences, nous allons évaluer le transfert pour différents objets complexes, pour différentes configurations $M \times N$, selon les deux approches de la redistribution que nous avons présentées à la section précédente. Nous mesurons la durée moyenne du transfert d'une série de données de type *double* entre deux codes parallèles (LAM/MPI) couplés avec RedCORBA, l'un jouant le rôle d'émetteur (M processeurs) et l'autre de récepteur (N processeurs). Plus précisément, nous mesurons le débit cumulé moyen (taille total des données / temps total) obtenu après répétition de plusieurs requêtes *put* pour des données de taille totale variant de quelques kilo-octets à plus de 100 Mo. Nous effectuons principalement des mesures de débits cumulés (en Mo/s), car elles permettent de mettre en évidence l'agrégation possible de la bande passante lors des flux parallèles de communication.

Les mesures de performance ont été entièrement réalisées sur un cluster *Grid'5000* situé à Bordeaux, composé de 50 bi-Opterons (AMD 64 bits avec 2 Go de RAM) et interconnecté via un switch à un réseau Giga-Ethernet (bande-passante théorique à 128 Mo/s). Nous utilisons OmniORB4, un ORB réputé pour ses bonnes performances : débit maximal mesuré sur ce réseau à 110 Mo/s (latence $107\ \mu s$) en point-à-point sur ce réseau contre 112 Mo/s pour LAM/MPI (latence $103\ \mu s$). On peut ainsi remarquer que OmniORB4 est pratiquement aussi performant que LAM/MPI sur ce réseau, ce qui vient du fait qu'il utilise une stratégie zéro-copie à l'envoi qui le rend très efficace. Notons toutefois que le débit maximal atteint par RedCORBA en point-à-point n'est que de 97.5 Mo/s, soit une baisse de $13\ \%$ par rapport à CORBA/OmniORB, qui est due à une recopie des données à la réception dans RedCORBA.

6.1. *Résultats pour l'approche spatiale*

6.1.1. *Cas des grilles structurées*

Nous avons choisi d'étudier le problème de la redistribution pour une grille structurée régulière (carré) 2D avec une série de données réelles associées aux nœuds de la grille. Nous considérons deux « motifs » de redistribution : *col2col* et *col2row*. Pour le premier motif, *col2col*, les deux codes possèdent une distribution bloc-colonne 1D. Si M et N sont égaux alors les distributions sont identiques et il n'y aura donc pas de redistribution à faire (1 seul message envoyé pour chaque émetteur à son vis-à-vis). Dans le second motif, *col2row*, le code émetteur possède une distribution bloc-colonne 1D et le code récepteur une distribution bloc-ligne 1D. Nous avons choisi ce dernier motif, car il implique un schéma de communication total entre les deux codes et donc un découpage des données dans la mémoire de chaque émetteur en N messages.

Les résultats obtenus (figure 16) pour les grilles structurées sont globalement satisfaisants, car ils montrent nettement l'agrégation de la bande-passante obtenue avec RedCORBA pour les cas 4×4, 8×8 et 16×16. Ces résultats sont d'autant plus satisfaisants que ces mesures prennent en compte le temps de gestion des requêtes et

des acquittements de RedCORBA, ainsi que la copie à la réception des données. Il faut noter que l'écart de débit entre RedCORBA et MPI reste toujours du même ordre de grandeur que celui mesuré en 1×1, c'est-à-dire d'environ -13%. Toutefois, de manière assez surprenante on observe que RedCORBA avec un motif du type *col2row* obtient le plus souvent de meilleures performances qu'avec le motif *a priori* idéal *col2col*, où un seul « gros » message est reçu par chaque processeur. Notre explication est que la réception de multiples « petits » messages en *col2row* s'effectue de manière concurrente dans RedCORBA (pool de threads CORBA) et ordonnée (ordonnancement de type round-robin), ce qui favorise la mise en place d'un pipeline entre la réception des données et la recopie des données précédentes.

Concernant le temps nécessaire pour générer les messages avec RedSYM & RedCORBA (incluant l'échange des descripteur), nous constatons que celui-ci dépend du nombre total de blocs de redistribution, résultant de l'intersection des blocs appartenant aux deux distributions (généralement corrélés à M et N). A titre d'exemple, si l'on considère la configuration 8×8 pour la grille 400×400, le nombre de blocs de redistribution sera de 8 dans le cas *col2col* contre 64 blocs le cas *col2row*, avec un temps de redistribution de 13.0 ms dans le premier cas contre 15.0 ms dans le second.

6.1.2. *Cas des boîtes d'atomes*

Nous présentons quelques résultats expérimentaux également obtenus avec l'approche spatiale pour les boîtes d'atomes (ensembles de particules), qui sont des objets complexes relativement proches. La principale différence tient au fait que les boîtes d'atomes sont décrites à l'aide d'une distribution des éléments par bloc dans $\mathbb{R}^n$. Comme le montre la figure 17, les performances dépendent directement du motif de redistribution, ce motif résultant de l'intersection des distributions choisies pour les deux codes couplés. Nous étudions les performances pour plusieurs types de distributions : bloc-colonne 1D (*col*), bloc-ligne 1D (*row*) et un pavage de l'espace en blocs 2D (*blk*). Plus précisément, la distribution *blk* correspond à une décomposition du domaine global 2D en $M \times M$ blocs (côté émetteur), encore raffiné en 2×2 sous-blocs, ce qui nous donne un pavage relativement fin du domaine : en tout $4.M^2$ blocs côté émetteur et $4.N^2$ blocs côté récepteur. Lorsque les distributions sont « bien ajustées » (*col2col* ou *blk2blk* en 8×8, les performances sont aussi bonnes que pour les grilles structurées. En revanche, lorsque les distributions sont « pathologiques » (*col2row* ou *blk2blk* en 8×7 et *col2row* en 8×8), les atomes à envoyer vers un certain destinataire doivent être extraits des blocs-source en fonction de leurs coordonnées spatiales (cf. section 4.2), ce qui induit un découpage important des données en mémoire et l'utilisation d'un cache à l'envoi dans RedCORBA (*i.e.* copie à l'envoi). Ce comportement est d'autant plus critique lorsque le nombre d'atomes augmente, entraînant une chute importante des performances au delà d'un certain seuil. Dans le contexte du pilotage, nous préférerons toujours utiliser la stratégie de placement des ensembles de particules plutôt que l'approche spatiale avec des boîtes d'atomes, car elle est plus simple à mettre en œuvre et offre de meilleures performances.

Le temps nécessaire pour générer les messages CORBA avec RedSYM et RedCORBA ne dépend que du nombre de processeurs M et N dans les cas simples. En revanche dans les cas pathologiques, la génération des messages devient linéaire par rapport au nombre d'atomes (parcours des atomes pour le calcul des différents masques d'extraction). A titre d'exemple, ce temps est d'environ 200 ms pour 100000 atomes en *col2row* 8×8 ou 8×7, de 50 ms en *blk2blk* 8×7 et de 10 ms *col2col* 8×8.

6.2. *Résultats pour l'approche placement*

Dans ces expériences, nous examinons le problème de la redistribution pour les ensembles de particules et des maillages non structurés en considérant différentes configurations $M \times N$, en comparant les stratégies de placement avec ou sans découpage.

6.2.1. *Cas des ensembles de particules*

Nous allons maintenant présenter quelques résultats expérimentaux dans le cas des ensembles de particules pour les stratégies de placement avec ou sans découpage, dans les configurations 8×8 et 8×7 (figure 18). Les particules sont initialement distribuées de manière équilibrée entre tous les processeurs émetteurs. La stratégie de placement avec découpage donne des résultats identiques à la stratégie sans découpage pour des cas triviaux (8×8), mais offre de bien meilleures performances lorsque M et N sont premiers entre eux. Dans le cas 8×7, le débit cumulé est même doublé, ce qui s'explique par le fait que, lors du placement sans découpage, le premier processeur côté réception se voit attribuer deux régions contre une seule pour les autres récepteurs, alors que dans la stratégie avec découpage, chaque processeur côté réception se voit attribuer la même proportion de données, d'où les meilleurs résultats observés.

Le temps nécessaire pour générer les messages CORBA avec RedSYM et RedCORBA est indépendant de la taille des données dans le cas des ensembles de particules et ne dépend que du nombre de processeurs M et N. Ce temps est de l'ordre de 10 ms dans les expériences précédentes.

6.2.2. *Cas des maillages non structurés*

Nous reproduisons des expériences similaires dans le cas des maillages non structurés. Il s'agit de comparer le comportement des redistributions de maillages en 8×8 et en 8×7 pour les stratégies de placement avec ou sans découpage (figure 19). Le maillage considéré est une grille de quadrilatères définie comme un maillage non structuré (*i.e.* liste des connectivités explicite), initialement découpée en colonne sur chaque processeur côté émetteur. Dans le cas trivial 8×8, le placement calculé est identique pour les deux stratégies et l'on pourrait s'attendre à obtenir d'aussi bonnes performances avec découpage que sans découpage. La différence de performance tient au fait que nous utilisons un cache à l'envoi pour la stratégie avec découpage, car cette stratégie produit des messages naturellement très décousus, sauf dans certains cas triviaux comme celui-ci. Le cas 8×7 est là pour mettre en évidence l'intérêt de la stratégie avec découpage pour équilibrer les flux de communication et obtenir de meilleures

performances. En effet, on constate que le débit est multiplié par 1.5 par rapport à la stratégie sans découpage. Notons que le débit cumulé n'est pas tout à fait multiplié par 2, à cause de l'utilisation d'un cache à l'envoi pour construire un message contigu.

Le temps nécessaire pour générer ces messages est très supérieur pour la stratégie avec découpage où nous avons recours à un algorithme plus complexe de partitionnement des maillages actuellement basé sur Metis, de l'ordre de 250 ms pour un maillage de 160 000 quadrilatères, contre 10 ms dans l'autre cas. D'une manière générale, le temps de génération des messages est constant (dépendant de M et N uniquement) dans la stratégie sans découpage alors qu'il est sur-linéaire par rapport au nombre de cellules du maillage dans la stratégie avec découpage.

7. Conclusion

Dans cet article, nous nous sommes écartés des modèles de redistribution classiques en bloc-cyclique pour nous intéresser à une approche plus générique pour la redistribution d'objets complexes, tels que les grilles structurées, les ensembles de particules ou les maillages non structurés, des objets très fréquents dans les codes de simulations numériques. Ce modèle de description permet de former une vision cohérente des données telles qu'elles sont distribuées sur chaque code et de construire des algorithmes de redistribution de « haut-niveau ». Sur la base de ce modèle, nous avons présenté deux approches de la redistribution : l'approche spatiale et l'approche placement de la redistribution. Si l'approche spatiale s'applique typiquement au contexte du couplage de codes, l'approche placement (avec ou sans découpage) se trouve particulièrement bien adaptée dans le contexte du pilotage $M \times N$. Les résultats expérimentaux ont permis de valider nos algorithmes en mettant en évidence qu'il était possible d'agréger la bande-passante même pour des schémas de communication complexes afin de réduire le temps total de transfert. Pour conclure, notons que ces algorithmes sont déjà utilisés dans l'environnement de pilotage EPSN, grâce à la bibliothèque RedGRID, pour visualiser en temps réel et en parallèle les résultats intermédiaires d'une simulation numérique elle-même parallèle (`www.labri.fr/epsn`).

Nos perspectives de recherche s'orientent actuellement selon trois axes principaux : (1) tout d'abord, l'étude des temps de génération des messages, étude devenant primordiale dès lors que la distribution des données est susceptible de changer dynamiquement ; (2) ensuite, la prise en compte de structures de données hiérarchiques (grilles multiniveaux, *Adaptative Mesh Refinment*) ; (3) et finalement, le développement d'une nouvelle couche de communication plus performante basée sur les possibilités de MPI-2 (Message-Passing Interface Forum, 1997).

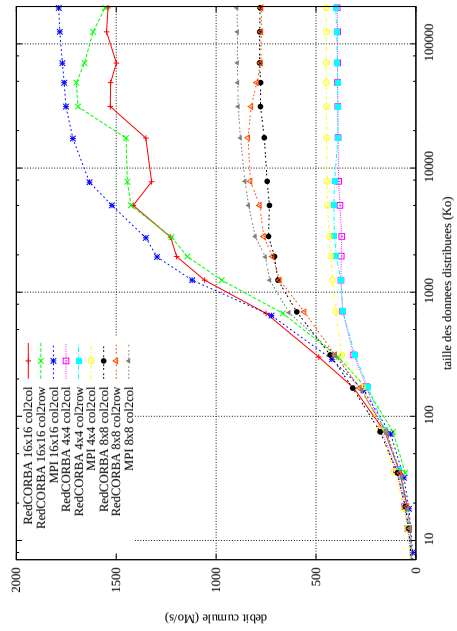

Figure 16. *Approche spatiale : cas des grilles 2D*

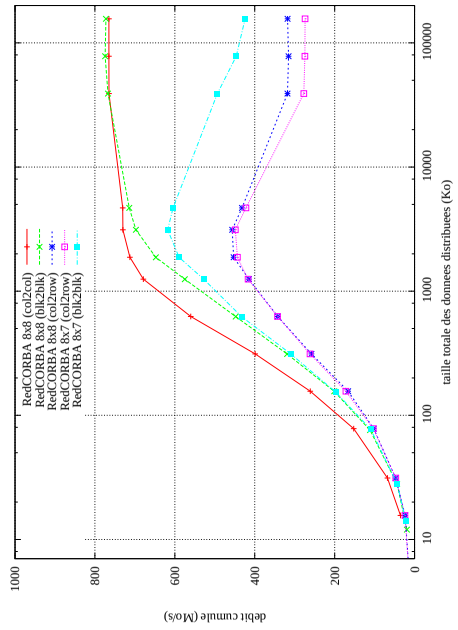

Figure 17. *Approche spatiale : cas des boîtes d'atomes dans* $\mathbb{R}^2$

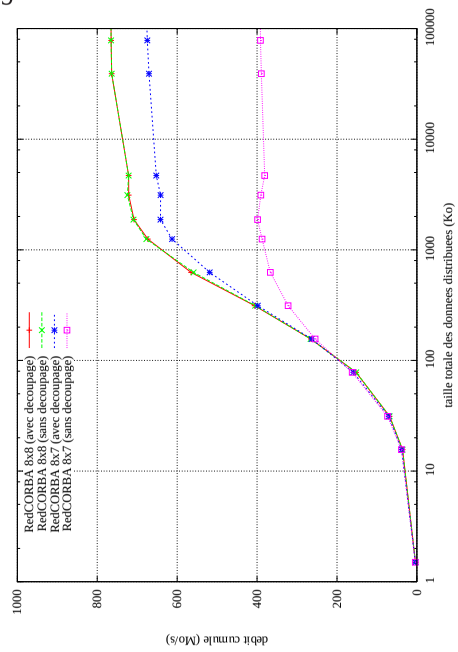

Figure 18. *Approche placement : cas des ensembles de particules*

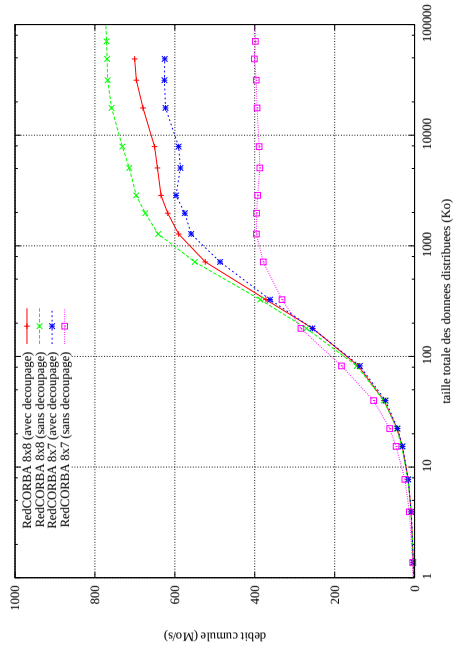

Figure 19. *Approche placement : cas des maillages non structurés*

8. Bibliographie

Beckman P. H., Fasel P. K., Humphrey W. F., Mniszewski S. M., "Efficient Coupling of Parallel Applications Using PAWS", *The Tenth IEEE International Symposium on High Performance Distributed Computing (HPDC)*, 1998.

Bertrand F., Bramley R., Damevski K. B., Kohl J. A., Bernholdt D. E., Larson J. W., Sussman A., "Data Redistribution and Remote Method Invocation in Parallel Component Architectures", *Proceedings of the 19th International Parallel and Distributed Processing Symposium : IPDPS 2005*, 2005.

Bertrand F., Yuan Y., Chiu K., Bramley R., "An Approach to Parallel MxN Communication", *12th High Performance Distributed Computing (HPDC)*, Seattle, WA, June, 2003.

Blackford L., Choi J., Cleary A., D'Azevado E., Demmel J., Dhillon I., Dongarra J., Hammarling S., Henry G., Petitet A., Stanley K., Walker D., Whaley R., "ScaLAPACK Users'Guide", , Technical Report, SIAM, 1997.

Esnard A., Dussere M., Coulaud O., "A Time-Coherent Model for the Steering of Parallel Simulations", *Euro-Par 2004 Parallel Processing*, Springer-Verlag, Pisa, Italy, p. 90-97, 2004.

Esnard A., Richart N., Coulaud O., "A Steering Environment for Online Parallel Visualization of Legacy Parallel Simulations", *Proceedings of the 10th International Symposium on Distributed Simulation and Real-Time Applications (DS-RT 2006)*, IEEE Press, Torremolinos, Malaga, Spain, p. 7-14, October, 2006.

Jeannot E., Wagner F., "Two Fast and Efficient Message Scheduling Algorithms for Data Redistribution through a Backbone", *IPDPS*, 2004.

Joppich W., Kürschner M., "MpCCI : a Tool for the Simulation of Coupled Applications", *Concurrency and Computation : Practice and Experience*, vol. 18, n° 2, p. 183-192, 2006.

Kalé L. V., Bhandarkar M., Brunner R., "Load Balancing in Parallel Molecular Dynamics", *Fifth International Symposium on Solving Irregularly Structured Problems in Parallel*, vol. 1457 of *Lecture Notes in Computer Science*, p. 251-261, 1998.

Karypis G., Kumar V., A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs, Technical Report n° 95-035, University of Minnesota, june, 1995.

Kerbyson D. J., Jones P. W., "A Performance Model of the Parallel Ocean Program", *International Journal of High Performance Computing Applications*, vol. 19, n° 3, p. 261-276, 2005.

Koebel C., Loveman D., Schreiber R., Steele G., Zosel M., *The High Performance Fortran Handbook*, MIT Press, Cambridge, 1994.

Kohl J. A., Papadopoulos P. M., "CUMULVS : Providing fault-tolerance, Visualization, and Steering of Parallel Applications", *International Journal of Supercomputer Applications*, vol. 11, p. 224-235, 1997.

Lee J.-Y., Sussman A., Efficient Communication Between Parallel Programs with InterComm, Technical Report n° CS-TR-4557 and UMIACS-TR-2004-0, University of Maryland, Department of Computer Science and UMIACS, January, 2004.

Message-Passing Interface Forum, *MPI-2.0 : Extensions to the Message-Passing Interface*, MPI Forum, june, 1997.

Pellegrini F., "Graph Partitioning based Methods and Tools for Scientific Computing", *Parallel Computing ETPSC'3*, vol. 23, p. 153-164, 1997.

Plimpton S., Hendrickson B., "Parallel Molecular Dynamics Algorithms for Simulation of Molecular Systems", *Parallel Computing in Computational Chemistry*, 592, American Chemical Society, p. 114-135, 1995.

Prylli L., Tourancheau B., "Efficient Block Cyclic Data Redistribution", *Euro-Par*, vol. 1, LNCS Springer Verlag, p. 155-164, August, 1996.

Ranganathan M., Acharya A., Edjlali G., Sussman A., Saltz J., "Runtime Coupling of Data-Parallel Programs", *ICS '96 : Proceedings of the 10th international conference on Supercomputing*, ACM Press, New York, NY, USA, p. 229-236, 1996.

Schroeder W., Martin K., Lorensen B., *The Visualisation ToolKit*, Kitware, 2002.

Thakur R., Choudhary A., Ramanujam J., "Efficient Algorithms for Array Redistribution", *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, n° 6, p. 587-594, 1996.

Upson C., Faulhaber T., Kamis D., Schlegel D., Laidlaw D., Vroom F., Gurwitz R., vanDam A., "The Application Visualization System : A Computational Environment for Scientific Visualization", *IEEE Computer Graphics and Applications*, vol. 9, p. 30-42, 1989.

Walker D. W., Otto S. W., "Redistribution of Block-Cyclic Data Distributions using MPI", *Concurrency : Practice and Experience*, vol. 8, n° 9, p. 707-728, 1996.

Article reçu le 23 janvier 2007
Accepté après révisions le 25 octobre 2007

Aurélien Esnard *est maître de conférence à l'université de Bordeaux I, membre de l'équipe-projet ScAlApplix de l'INRIA Futurs et du laboratoire bordelais de recherche en informatique (LaBRI). Dans le contexte de la simulation numérique haute performance, ses travaux portent sur la thématique du couplage de codes et plus particulièrement sur la problématique du pilotage interactif des simulations au travers des environnements de visualisation scientifique.*

ANNEXE POUR LE SERVICE FABRICATION

A FOURNIR PAR LES AUTEURS AVEC UN EXEMPLAIRE PAPIER
DE LEUR ARTICLE ET LE COPYRIGHT SIGNE PAR COURRIER
LE FICHIER PDF CORRESPONDANT SERA ENVOYE PAR E-MAIL

1. ARTICLE POUR LA REVUE :
   RSTI - TSI – 27/2008. RenPar'06

2. AUTEURS :
   Aurélien Esnard[*]

3. TITRE DE L'ARTICLE :
   RedGRID : un environnement pour la redistribution d'objets complexes

4. TITRE ABRÉGÉ POUR LE HAUT DE PAGE MOINS DE 40 SIGNES :
   RedGRID

5. DATE DE CETTE VERSION :
   2 avril 2008

6. COORDONNÉES DES AUTEURS :

   – adresse postale :

   [*] Université de Bordeaux I, LaBRI et INRIA Futurs (projet ScAlApplix).
   351, cours de la Libération, 33405 Talence

   aurelien.esnard@labri.fr

   – téléphone : 05 40 00 38 79

   – télécopie :

   – e-mail : aurelien.esnard@labri.fr

7. LOGICIEL UTILISÉ POUR LA PRÉPARATION DE CET ARTICLE :
   LaTeX, avec le fichier de style `article-hermes.cls`,
   version 1.23 du 17/11/2005.

8. FORMULAIRE DE COPYRIGHT :
   Retourner le formulaire de copyright signé par les auteurs, téléchargé sur :
   `http://www.revuesonline.com`

SERVICE ÉDITORIAL – HERMES-LAVOISIER
14 rue de Provigny, F-94236 Cachan cedex
Tél. : 01-47-40-67-67
E-mail : revues@lavoisier.fr
Serveur web : http://www.revuesonline.com