RENPAR'15 / CFSE'3 / SympAAA'2003
La Colle sur Loup, France, 15 au 17 octobre 2003

Placement et équilibrage de charge pour calculs itératifs sur grappes hétérogènes

Arnaud Legrand, Hélène Renard, Yves Robert, Frédéric Vivien
LIP, UMR CNRS-INRIA-UCBL 5668, École normale supérieure de Lyon, France
{Arnaud.Legrand,Helene.Renard,Yves.Robert,Frederic.Vivien}@ens-lyon.fr

Résumé

Cet article est consacré à la mise en œuvre d'algorithmes itératifs sur plate-formes hétérogènes. Les données sont réparties sur l'ensemble des processeurs, qui sont organisés en anneau virtuel. À chaque itération, des calculs indépendants sont effectués en parallèle et des communications ont lieu entre les processeurs consécutifs de l'anneau. Le problème est de déterminer comment partitionner les données et comment les répartir pour que le temps total d'exécution soit minimal. Une difficulté majeure est d'inclure un anneau dans un réseau qui n'est pas forcément un graphe complet, de telle sorte que certains liens de communication soient partagés par plusieurs couples de ressources. Nous avons démontré un résultat de complexité qui établit la difficulté de ce problème, et nous proposons une heuristique polynomiale qui fournit des schémas efficaces d'allocation, de routage et de distribution de données.

1. Introduction

Nous nous intéressons à la mise en œuvre d'algorithmes itératifs sur des grappes hétérogènes. Ces algorithmes fonctionnent avec un volume important de données, qui sera réparti sur l'ensemble des processeurs. À chaque itération, des calculs indépendants seront effectués en parallèle et certaines communications auront lieu. Nous énonçons le problème comme suit : l'algorithme itératif fonctionne répétitivement sur une matrice rectangulaire de données qui est divisée en tranches verticales allouées aux processeurs. À chaque étape de l'algorithme, les tranches sont mises à jour localement et les informations frontières sont échangées entre tranches consécutives. Cette contrainte géométrique implique que les processeurs soient organisés en anneau virtuel. Chaque processeur communiquera seulement deux fois, une fois avec son prédécesseur (virtuel) dans l'anneau et une fois avec son successeur. Il n'existe pas de raison *a priori* de réduire le partitionnement des données à une unique dimension et de ne l'appliquer que sur un anneau de processeurs unidimensionnel. Cependant, un tel partitionnement est très naturel et nous montrerons que trouver l'optimal est déjà très difficile.

Nous considérons une grappe totalement hétérogène, composée de processeurs de vitesses de calculs différentes, communicant par des liens de bandes passantes différentes. Du point de vue architectural, le problème se décompose en deux parties : (i) sélectionner les processeurs participant à la solution et décider de leur position dans l'anneau ; (ii) définir les routes pour aller d'un processeur à son successeur. Une difficulté majeure est que certaines routes partageront des liens physiques, les réseaux de communication de grappes hétérogènes n'étant pas totalement connectés. Si plusieurs routes partagent le même lien physique, nous décidons quelle fraction de bande passante sera attribuée à chaque route.

Une fois l'anneau et le routage décidés, il reste à déterminer le meilleur partitionnement des données. La qualité de la solution finale dépend d'un grand nombre de paramètres de l'application ainsi que de l'architecture, et il faut s'attendre à ce que le problème d'optimisation soit difficile à résoudre.

Pour évaluer l'impact du partage de la bande passante des liens, nous travaillons aussi avec la version simple du problème où nous voyons le réseau comme un graphe complet : entre chaque paire de nœuds, le routage est fixé (plus court chemin en terme de bande passante), et la bande passante est définie par le lien le plus lent dans le chemin routé. Ce modèle simplifié n'est pas très réaliste car aucun conflit de lien n'est pris en compte, mais il mènera à un anneau solution qui pourra être comparé à celui obtenu en tenant compte du partage des liens, fournissant de ce fait une manière commode d'évaluer l'impact des différentes hypothèses faites sur les communications.

La section suivante (Section 2) est consacrée aux spécifications précises et formelles du problème d'optimisation précédent, lequel est noté SHAREDRING ; nous discutons de la version simplifiée du problème,

sans partage des liens, que nous notons SLICERING. Ensuite, dans la section 3, nous établissons un résultat de complexité : nous montrons que le problème associé à SHAREDRING est NP-complet. Après la preuve de ce résultat, la section 4 traite de la conception d'heuristiques en temps polynomial pour résoudre le problème d'optimisation SHAREDRING. La section 5 est similaire pour le problème SLICERING. Nous présentons nos résultats expérimentaux dans la section 6. Nous examinons dans la section 7 les travaux déjà publiés sur ce sujet. Finalement, nous concluons à la section 8.

2. Cadre de travail

Nous présentons ici le modèle de plate-forme utilisé et nous énonçons formellement le problème d'optimisation à résoudre, puis nous introduisons une version simplifiée sans partage de liens.

2.1. Présentation de la plate-forme

Coûts de calcul

La plate-forme de calcul ciblée est représentée par un graphe orienté $G = (P, E)$. Chaque nœud P_i du graphe, $1 \leqslant i \leqslant |P| = p$, représente une ressource de calcul, et est pondéré par son temps de cycle w_i : P_i nécessite w_i unités de temps pour effectuer une tâche unitaire.

Coûts de communication

Les arêtes du graphe représentent les liens de communication et sont étiquetées avec les bandes passantes disponibles. Si $e \in E$ est un lien orienté de P_i à P_j, notons b_e la bande passante du lien. Nous aurons besoin de L/b_e unités de temps pour transférer un message de taille L de P_i à P_j en utilisant le lien e. Lorsque plusieurs messages partagent le même lien, chacun reçoit une fraction de la bande passante disponible. Si deux messages partagent le même lien e et si le premier message utilise deux tiers de la bande passante, i.e. $2b_e/3$, alors le second message ne pourra utiliser qu'au plus $b_e/3$. Les fractions de bandes passantes qui sont allouées aux messages peuvent être déterminées par l'utilisateur à la seule condition que la somme de ces fractions ne dépasse pas la bande passante totale du lien. En pratique, un protocole tel que celui décrit dans [7] nous offre une telle liberté pour la stratégie de routage.

Routage

Supposons que nous pouvons décider comment les messages sont routés d'un processeur à un autre et que nous voulons router un message de taille L de P_i à P_j, en passant par k arêtes $e_1, e_2, \ldots, e_k$. Pour chaque arête e_m, le message aura une fraction f_m de la bande passante b_{e_m}. La vitesse globale de communication le long du chemin sera limitée par la plus petite bande passante disponible : nous avons besoin de L/b unités de temps pour router le message, où $b = \min_{1 \leqslant m \leqslant k} f_m$: c'est comme si nous avions un lien direct dédié au routage de ce message mais de bande passante réduite b.

Paramètres de l'application : calculs

Soit W la taille totale du travail qui doit être accompli à chaque pas de l'algorithme. Le processeur P_i effectuera une partie $\alpha_i.W$ de ce travail, où $\alpha_i \geqslant 0$ pour $1 \leqslant i \leqslant p$ et $\sum_{i=1}^{p} \alpha_i = 1$. Notons que $\alpha_j = 0$, pour un certain j, signifie que le processeur P_j ne participe pas au calcul. En effet, il n'y a aucune raison *a priori* que toutes les ressources soient utilisées, surtout lorsque le travail n'est pas très important : les communications supplémentaires encourues en ajoutant plus de processeurs peuvent ralentir l'ensemble du processus en dépit de la vitesse de traitement cumulée.

Paramètres de l'application : communications dans l'anneau

Les processeurs sont organisés le long d'un anneau (qui n'est pas encore déterminé). Après avoir mis à jour ses données de taille $\alpha_i W$, chaque processeur P_i envoie un message de longueur H fixée (typiquement, les données frontières) à son successeur. Pour illustrer la relation entre W et H, nous pouvons voir la matrice de données comme un rectangle composé de W colonnes de hauteur H, ainsi une seule colonne est échangée entre paire de processeurs consécutifs dans l'anneau (le paramètre H peut être représenté comme un volume fixe de communication).
Soit succ(i) et pred(i) le successeur et le prédécesseur de P_i dans l'anneau virtuel. Il existe un chemin

de communication $\mathcal{S}_i$ ($\mathcal{S}$ pour « successeur ») de P_i à $P_{\text{succ}(i)}$ dans le réseau : soit $s_{i,m}$ la fraction de la bande passante b_{e_m} du lien physique e_m qui a été allouée pour le chemin $\mathcal{S}_i$. Évidemment, si un lien e_r n'est pas utilisé dans le chemin, alors $s_{i,r} = 0$. Soit $c_{i,\text{succ}(i)} = \frac{1}{\min_{e_m \in \mathcal{S}_i} s_{i,m}}$: alors P_i a besoin de $H.c_{i,\text{succ}(i)}$ unités de temps pour envoyer un message de taille H à son successeur $P_{\text{succ}(i)}$. De même, nous définissons le chemin de communication $\mathcal{P}_i$ ($\mathcal{P}$ pour « prédécesseur ») de P_i à $P_{\text{pred}(i)}$ dans le réseau ; $p_{i,m}$ est la fraction de la bande passante b_{e_m} du lien physique e_m qui a été allouée pour le chemin $\mathcal{P}_i$, et $c_{i,\text{pred}(i)} = \frac{1}{\min_{e_m \in \mathcal{P}_i} p_{i,m}}$. Alors P_i a besoin de $H.c_{i,\text{pred}(i)}$ unités de temps pour envoyer un message de taille H à son prédécesseur $P_{\text{pred}(i)}$.

Fonction objective

Le coût total d'une seule étape de l'algorithme itératif est le maximum, parmi tous les processeurs impliqués, du temps passé à calculer et à communiquer :

$$T_{\text{step}} = \max_{1 \leqslant i \leqslant p} \mathbb{I}\{i\}[\alpha_i.W.w_i + H.(c_{i,\text{pred}(i)} + c_{i,\text{succ}(i)})]$$

où $\mathbb{I}\{i\}[x] = x$ si P_i participe au calcul et 0 sinon. En résumé, le but est de déterminer la meilleure méthode pour sélectionner q processeurs parmi p disponibles, de leur assigner des charges de travail, de les arranger le long d'un anneau et de partager la bande passante du réseau de telle sorte que le temps total d'exécution pas-à-pas soit minimal.
Notre cadre de travail peut être étendu, par exemple pour prendre en compte des liens bidirectionnels, des liens multiples ainsi que des « backbones » (cf [8, 9]).

2.2. Le problème d'optimisation SHAREDRING

Nous énonçons le problème d'optimisation de la manière suivante :

Définition 1 (SHAREDRING(p,w_i,E,b_{e_m},W,H)). *Étant donnés p processeurs P_i de temps de cycle w_i et $|E|$ liens de communication e_m de bande passante b_{e_m}, étant donnés W la charge de travail totale et H le volume de communication à chaque étape, minimiser*

$$T_{step} = \min_{1 \leqslant q \leqslant p} \min_{\substack{\sigma \in \Theta_{q,p} \\ \sum_{i=1}^{q} \alpha_{\sigma(i)} = 1}} \max_{1 \leqslant i \leqslant q} \left(\alpha_{\sigma(i)}.W.w_{\sigma(i)} + H.(c_{\sigma(i),\sigma(i-1 \bmod q)} + c_{\sigma(i),\sigma(i+1 \bmod q)})\right) \quad (1)$$

Dans l'équation 1, $\Theta_{q,p}$ symbolise l'ensemble des fonctions injectives $\sigma : [1..q] \to [1..p]$, lesquelles indexent les q processeurs sélectionnés pour l'anneau, pour tout q compris entre 1 et p. Pour chaque anneau candidat représenté par une telle fonction σ, il existe des contraintes cachées par l'introduction des quantités $c_{\sigma(i),\sigma(i-1 \bmod q)}$ et $c_{\sigma(i),\sigma(i+1 \bmod q)}$: il existe $2q$ chemins de communication, le chemin $\mathcal{S}_i$ de $P_{\sigma(i)}$ à son successeur $P_{\text{succ}(\sigma(i))} = P_{\sigma(i+1 \bmod q)}$ et le chemin $\mathcal{P}_i$ de $P_{\sigma(i)}$ à son prédécesseur $P_{\text{pred}(\sigma(i))} = P_{\sigma(i-1 \bmod q)}$, pour tout $1 \leqslant i \leqslant q$. Pour chaque lien e_m dans le réseau interconnecté, soit $s_{\sigma(i),m}$ (resp. $p_{\sigma(i),m}$) la fraction de la bande passante b_{e_m} qui est allouée au chemin $\mathcal{S}_{\sigma(i)}$ (resp. $\mathcal{P}_{\sigma(i)}$). Nous obtenons les équations suivantes :

$$\begin{cases} s_{\sigma(i),m} \geqslant 0,\ p_{\sigma(i),m} \geqslant 0, & 1 \leqslant i \leqslant q, \quad 1 \leqslant m \leqslant E, \\ c_{\sigma(i),\text{succ}(\sigma(i))} = \frac{1}{\min_{e_m \in \mathcal{S}_{\sigma(i)}} s_{\sigma(i),m}} & 1 \leqslant i \leqslant q, \end{cases} \qquad \begin{cases} \sum_{i=1}^{q} (s_{\sigma(i),m} + p_{\sigma(i),m}) \leqslant b_{e_m} & 1 \leqslant m \leqslant E \\ c_{\sigma(i),\text{pred}(\sigma(i))} = \frac{1}{\min_{e_m \in \mathcal{P}_{\sigma(i)}} p_{\sigma(i),m}} & 1 \leqslant i \leqslant q \end{cases}$$

La deuxième équation vient du fait que la bande passante du lien e_m ne doit pas être dépassée. Puisque chaque chemin de communication $\mathcal{S}_{\sigma(i)}$ ou $\mathcal{P}_{\sigma(i)}$ n'impliquera que peu d'arêtes, la plupart des quantités $s_{\sigma(i),m}$ et $p_{\sigma(i),m}$ seront égales à zéro. En fait, nous avons écrit $e_m \in \mathcal{S}_{\sigma(i)}$ si l'arête e_m est utilisée dans le chemin $\mathcal{S}_{\sigma(i)}$, i.e. si $s_{i,m}$ ne vaut pas zéro (de même $e_m \in \mathcal{P}_{\sigma(i)}$ si $p_{i,m}$ ne vaut pas zéro).
À partir de l'équation 1, nous pouvons voir que plus le ratio $\frac{W}{H}$ sera grand, plus le nombre de processeurs augmentera dans la solution optimale : dans ce cas, l'impact des communications devient petit devant le coût des calculs, et ces calculs doivent être répartis sur l'ensemble des ressources. Mais même dans ce cas, nous devons encore décider comment organiser les processeurs en anneau, construire les routes de communication, assigner les fractions de bandes passantes et finalement allouer les données. Extraire le « meilleur » anneau semble être un problème combinatoire difficile.

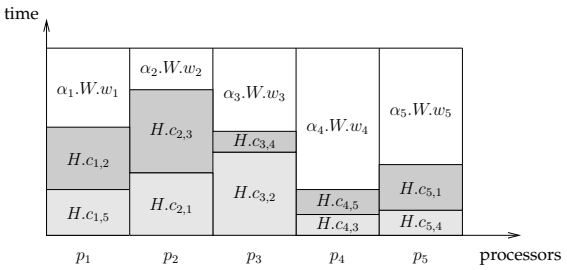

FIG. 1 – Résumé des temps de calcul et de communication avec $q = 5$ processeurs.

2.3. Le problème d'optimisation SLICERING

Dans cette section, nous trait ons d'une version simplifiée du problème sans lien partagé, que nous appelons SLICERING. En fait, le problème d'optimisation SLICERING est exactement donné par l'équation 1, mais il y a une simplification importante concernant les chemins de routage et les coûts de communication : le chemin de routage entre toute paire de processeurs est fixe, ainsi que sa bande passante. Cela conduit à supposer un réseau totalement interconnecté, où la bande passante entre P_i et P_j est constante. Étant donné un réseau « réel », construisons le problème simplifié de la manière suivante : nous prenons le réseau en entrée et calculons les plus courts chemins (en terme de bande passante) entre toutes les paires de processeurs. Si un même lien est partagé par plusieurs chemins, la bande passante disponible pour chaque chemin est surévaluée. Cela conduit à un (faux) réseau totalement connecté.

3. Complexité

Le résultat suivant établit la difficulté intrinsèque du problème SHAREDRING :

Theorème 1. *Le problème de décision associé au problème d'optimisation* SHAREDRING *est NP-complet.*

Par manque de place, nous renvoyons à [8] pour la preuve. Malgré la simplification, le problème de décision associé au problème d'optimisation SLICERING est également NP-complet [9].

4. Heuristique pour le problème SHAREDRING

Nous décrivons maintenant une heuristique en temps polynomial pour résoudre le problème d'optimisation SHAREDRING et construire un anneau solution. Nous décrivons l'heuristique en trois étapes : (i) l'algorithme glouton utilisé pour construire un anneau solution ; (ii) la stratégie mise en œuvre pour assigner des fractions de bande passante pendant la construction ; et (iii) des optimisations finales.

4.1. Construction de l'anneau

Nous considérons un anneau solution impliquant q processeurs, numérotés de P_1 à P_q. Idéalement, tous ces processeurs devraient nécessiter la même quantité de temps pour calculer et communiquer : autrement, nous devrions diminuer légèrement la charge de calcul du dernier processeur pour satisfaire le fait que les calculs soient suivis des communications, et assigner le travail supplémentaire à un autre[1]. Nous avons donc (voir Figure 1 pour une illustration) : $T_{\text{step}} = \alpha_i.W.w_i + H.(c_{i,i-1} + c_{i,i+1})$ pour tous les i (les indices des coûts de communication sont pris modulo q). Puisque $\sum_{i=1}^{q} \alpha_i = 1$, il en découle que $\sum_{i=1}^{q} \frac{T_{\text{step}} - H.(c_{i,i-1}+c_{i,i+1})}{W.w_i} = 1$. En définissant $w_{\text{cumul}} = \frac{1}{\sum_{i=1}^{q} \frac{1}{w_i}}$, nous obtenons :

$$T_{\text{step}} = W.w_{\text{cumul}} \left(1 + \frac{H}{W} \sum_{i=1}^{q} \frac{c_{i,i-1} + c_{i,i+1}}{w_i} \right) \tag{2}$$

Nous utiliserons l'équation 2 comme base pour un algorithme glouton pour générer un anneau solution itérativement. L'algorithme glouton commence par sélectionner la meilleure paire de processeurs. En-

[1] Ici nous supposons implicitement que la charge totale de travail peut être arbitrairement partitionnée. De ce fait nous utilisons le cadre de travail de la « charge divisible ». Voir [8] pour des références d'articles sur la théorie de la charge divisible.

suite, il inclut itérativement un nouveau processeur dans l'anneau solution actuel. Supposons que nous avons déjà sélectionné un anneau de r processeurs. Pour chaque processeur restant P_i, nous cherchons où l'insérer dans l'anneau actuel.

Calculons le coût d'insertion de P_k entre P_i et P_j. Nous nous appuyons sur une autre heuristique qui construit les chemins de communication et alloue des fractions de bande passante (explication en section 4.2), dans le but de calculer les nouveaux coûts $c_{k,j}$ (chemin de P_k à son successeur P_j), $c_{j,k}$ (le chemin inverse), $c_{k,i}$ (chemin de P_k à son prédécesseur P_i), et $c_{i,k}$ (le chemin inverse). Une fois que nous avons ces coûts, nous calculons la nouvelle valeur de T_{step} comme suit :

- Nous mettons à jour w_{cumul} en ajoutant le nouveau processeur P_k, ce qui diminue w_{cumul}.
- Dans la somme $\sum_{s=1}^{r} \frac{c_{\sigma(s),\sigma(s-1)}+c_{\sigma(s),\sigma(s+1)}}{w_{\sigma(s)}}$, nous supprimons les deux termes correspondant aux deux chemins entre P_i et P_j (par hypothèse nous avions $i = \sigma(s)$ et $j = \sigma(s+1)$ pour un s donné), et nous insérons les nouveaux termes $\frac{c_{k,j}+c_{k,i}}{w_k}$, $\frac{c_{j,k}}{w_j}$ et $\frac{c_{i,k}}{w_i}$.

Cette étape de l'heuristique a une complexité proportionnelle à $(p-r).r$ fois le coût pour calculer quatre chemins de communication. Pour finir, nous agrandissons l'anneau jusqu'à avoir p processeurs. Nous retournons la valeur minimale obtenue pour T_{step}. La complexité totale est $\sum_{r=1}^{p}(p-r)rC = O(p^3)C$, où C est le coût de calcul de quatre chemins dans le réseau. Notons qu'il est important de tester toutes les valeurs de r, car T_{step} peut ne pas varier de façon monotone avec r.

4.2. Allocation de bande passante

Dans cette section, nous supposons que nous avons déjà un anneau de r processeurs, une paire (P_i, P_j) de processeurs successifs dans l'anneau et un nouveau processeur P_k à insérer entre P_i et P_j. Parallèlement à l'anneau, nous avons construit $2r$ chemins de communication et une certaine fraction de la bande passante initiale a été allouée à chacun de ces chemins. Pour construire les quatre nouveaux chemins impliquant P_k, nous raisonnons sur le graphe $G = (V, E, b)$ où chaque arête est étiquetée par la bande passante disponible restante : maintenant $b(e_m)$ n'est plus la bande passante initiale de l'arête e_m, mais ce qui a été laissé par les $2r$ chemins. La première chose à faire est de réinjecter dans le réseau les fractions de bande passante utilisées par les deux chemins de communication entre P_i et P_j. Nous utilisons un algorithme simple de calcul du plus court chemin (en termes de bande passante) pour déterminer les quatre chemins, de P_k à P_i et P_j et vice-versa. Il y a ici une subtilité, car ces quatre chemins peuvent partager certains liens. La stratégie que nous utilisons est la suivante :

- Nous calculons indépendamment quatre chemins de bande passante maximale, à l'aide d'un algorithme standard de calcul du plus court chemin [5] sur G.
- Si certains chemins partagent des liens, nous ne changeons pas les chemins, mais nous employons une méthode (analytique) brutale pour calculer les fractions de bande passante minimisant l'équation 2 devant être allouées à chaque chemin, et nous mettons à jour les coûts des chemins en conséquence.

Maintenant que nous avons les chemins et leurs coûts, nous calculons la nouvelle valeur de T_{step} comme expliqué précédemment. Le coût C de calcul des quatre chemins dans le réseau est $O(p + E)$.

4.3. Optimisations

Un moyen concis de décrire l'heuristique est le suivant : nous construisons un anneau de façon gloutonne en fractionnant les bandes passantes pour insérer de nouveaux processeurs. Pour diminuer le coût de l'heuristique, nous ne recalculons jamais les fractions de bande passante ayant été assignées aux précédents chemins de communication. Quand l'heuristique se termine, nous avons un anneau de q processeurs, q charges de travail, $2q$ chemins de communication, $2q$ fractions de bande passante, $2q$ coûts de communication pour ces chemins, et une valeur réaliste de T_{step}. Nous avons implémenté deux variantes destinées à affiner cette solution. L'idée des deux variantes est de tout conserver, sauf les fractions de bande passante (et les charges de travail), et de recalculer celles-ci chaque fois que l'on a inséré un nouveau processeur dans l'anneau. Puisque nous connaissons les $2q$ chemins, nous pouvons ré-évaluer les fractions de bande passante, en utilisant une approche globale (Voir [8] pour les détails) :

Méthode 1 : Max-min fairness. C'est l'algorithme traditionnel de partage de bande passante [2], conçu pour maximiser la bande passante minimum allouée à un chemin.

Méthode 2 : résolution quadratique à l'aide du logiciel KINSOL. Une fois obtenu un anneau ainsi que tous les chemins de communication, le programme pour minimiser T_{step} est quadratique en les inconnues α_i, $s_{i,j}$ et $p_{i,j}$. Nous utilisons la bibliothèque KINSOL [11] pour le résoudre.

5. Heuristique pour le problème SLICERING

L'heuristique gloutonne pour le problème SLICERING est similaire à la précédente. Elle commence par sélectionner le processeur (ou la paire de processeurs) le plus rapide. Ensuite, elle inclut itérativement un nouveau processeur dans l'anneau solution actuel : pour chaque processeur restant P_i, elle cherche où l'insérer dans l'anneau courant et elle retient le processeur et la paire de processeurs qui minimisent le coût d'insertion et nous stockons la valeur de T_{step}. Cette étape de l'heuristique a une complexité proportionnelle à $(p-r).r$. Nous augmentons l'anneau jusqu'à avoir p processeurs et nous retournons la valeur minimale obtenue pour T_{step}. La complexité totale est $\sum_{r=1}^{p}(p-r)r = O(p^3)$. Il est important de tester toutes les valeurs de r, puisque T_{step} peut ne pas varier conjointement à r (Voir [9] pour plus de détails). Pour finir notons que l'heuristique pour le problème SHAREDRING (Section 4) retournerait la même solution que l'heuristique décrite ici, si un graphe totalement connecté lui était fourni en entrée.

6. Résultats expérimentaux

6.1. Description des plate-formes

Nous avons expérimenté nos heuristiques sur deux plate-formes, l'une située à l'ENS de Lyon et l'autre à l'université de Strasbourg. La figure 2 montre la plate-forme de Lyon, composée de 14 unités de calcul et 3 routeurs. Une vue abstraite de la plate-forme de Lyon est représentée sur la figure 3. Sur cette figure les nœuds 0 à 13, en forme de cercle, sont les processeurs, alors que les nœuds 14 à 16, en forme de losange, sont les routeurs. Les arêtes sont étiquetées avec les bandes passantes des liens. Les temps de cycle des processeurs (en secondes par megaflop) sont rassemblés dans le tableau 1. De façon similaire, la plate-forme de Strasbourg est composée de 13 unités de calcul et de 6 routeurs. Les temps de cycle des processeurs sont rassemblés dans le tableau 2.

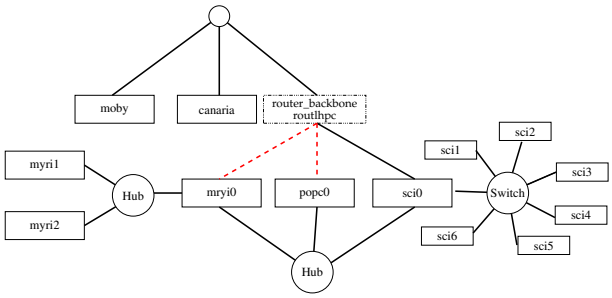

FIG. 2 – Topologie de la plate-forme de Lyon.

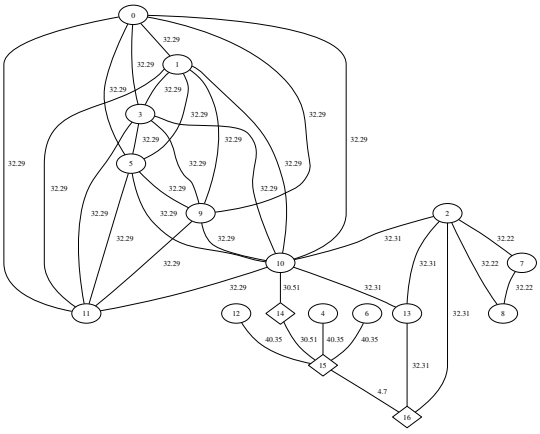

FIG. 3 – Vue abstraite de la plate-forme de Lyon.

P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
0.0206	0.0206	0.0206	0.0206	0.0291	0.0206	0.0087	0.0206	0.0206

P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}
0.0206	0.0206	0.0206	0.0291	0.0451	0	0	0

TAB. 1 – Temps de cycle des processeurs pour la plate-forme de Lyon.

P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9
0.0087	0.0072	0.0087	0.0131	0.016	0.0058	0.0087	0.0262	0.0102	0.0131

P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}	P_{17}	P_{18}
0.0072	0.0058	0.0072	0	0	0	0	0	0

TAB. 2 – Temps de cycle des processeurs pour la plate-forme de Strasbourg.

6.2. Résultats

Pour les deux topologies, nous évaluons l'impact du partage de liens de la manière suivante. Dans la première heuristique, nous construisons l'anneau solution sans prendre en compte le partage de liens. Utilisant le graphe abstrait de la figure 3, nous exécutons un algorithme de plus court chemin pour déterminer la bande passante entre chaque paire de processeurs, simulant par ce biais un réseau totalement interconnecté. Puis nous retournons l'anneau solution calculé par l'heuristique gloutonne pour le problème SLICERING, comme décrit en Section 5. La valeur de T_{step} obtenue par l'heuristique peut

très bien ne pas être réalisable du fait que le réseau réel n'est pas totalement connecté. De ce fait, nous conservons l'anneau et les chemins de communication entre les processeurs adjacents dans cet anneau et nous les évaluons en utilisant l'optimisation de programmation quadratique.

La seconde heuristique est l'heuristique gloutonne conçue en Section 4 pour le problème SHAREDRING, en utilisant l'optimisation de programmation quadratique. La différence majeure entre les deux heuristiques est que la dernière prend en compte la contention des liens lors de la construction de l'anneau.

Pour comparer la valeur de $\frac{T_{step}}{W}$ retournée par les deux algorithmes nous utilisons différents rapports des coûts de communication et de calcul (H/W). Les tableaux 3 et 4 montrent ces valeurs pour chaque plate-forme. Les conclusions pouvant être tirées de ces expérimentations sont les suivantes :

– Lorsque l'impact du coût de communication est bas, le but principal est d'équilibrer les calculs et les deux heuristiques sont équivalentes.
– Lorsque le rapport H/W devient plus important, l'effet de la contention des liens devient évident et la solution retournée par la seconde heuristique est bien meilleure.

Pour conclure, nous mettons en avant le fait qu'une modélisation précise des communications a un impact important sur la performance des stratégies d'équilibrage de charge.

Ratio H/W	H1 : slice-ring	H2 : shared-ring	Amélioration
0.1	3.17	3.15	0.63%
1	3.46	3.22	6.94%
10	10.39	3.9	62.46%

TAB. 3 – T_{step}/W pour chaque heuristique sur la plate-forme de Lyon.

Ratio H/W	H1 : slice-ring	H2 : shared-ring	Amélioration
0.1	7.32	7.26	0.82%
1	9.65	7.53	21.97%
10	19.24	10.26	46.67%

TAB. 4 – T_{step}/W pour chaque heuristique sur la plate-forme de Strasbourg.

7. Travaux antérieurs

Les stratégies d'équilibrage de charge ont été largement étudiées, à la fois pour des plate-formes homogènes [10] et hétérogènes (chapitre 25 de [4]). La distribution des calculs (ainsi que des données associées) peut être effectuée soit dynamiquement, soit statiquement, soit par un mélange des deux.

Une grande part de la littérature traite de stratégies dynamiques, qui nécessitent des phases périodiques de ré-allocation pour remédier au déséquilibre observé. Même si nous visons des schémas statiques, nous traitons brièvement de quelques références importantes dans le domaine des approches dynamiques. Des paradigmes simples sont basés sur l'idée *« utiliser le passé pour prédire le futur »*, i.e. utiliser la vitesse de calcul de chaque machine actuellement observée pour décider de la prochaine répartition du travail [1]. Plusieurs auteurs [12] proposent une politique d'allocation qui minimise dynamiquement la dégradation du système (ainsi que le coût de ré-allocation) pour chaque étape de calcul. D'autres articles [13, 6] plébiscitent des schémas locaux où les données sont échangées uniquement entre processeurs voisins. De manière générale, il y a une difficulté certaine à déterminer un compromis entre les paramètres de distribution des données et les politiques de génération de processus et de migration. Des calculs redondants peuvent être nécessaires pour utiliser une plate-forme au meilleur de ses capacités.

Dans le contexte d'une approche orientée bibliothèque logicielle, des stratégies dynamiques sont difficiles à introduire car elles impliquent une gestion compliquée de la mémoire. Des stratégies statiques sont plus spécifiques mais se révèlent utiles si suffisamment de connaissance peut être injectée dans le processus décisionnel d'ordonnancement et d'application. En d'autres termes, si les caractéristiques de la plate-forme cible (vitesses des processeurs et capacités des liens) et de l'application cible (coût des calculs et des communications associés à chaque partition de données) sont connues avec suffisamment de précision, alors un excellent niveau de performance peut être atteint par le biais de stratégies statiques. Toutefois, des schémas sophistiqués de distribution de données (tels ceux présentés dans ce papier) sont indispensables pour atteindre ce niveau de performance.

Plusieurs auteurs ont traité de l'implémentation statique de noyaux d'algèbre linéaire sur des plate-formes hétérogènes (voir [8] pour des références). Les principales conclusions de ces papiers s'appliquent pour trois types de problèmes :

– La distribution de tâches indépendantes sur un réseau unidimensionnel (linéaire) de processeurs hétérogènes est facile.
– La distribution de tâches indépendantes sur une grille bidimensionnelle de processeurs est difficile.

Nous devons rechercher la meilleure distribution du travail sur ladite grille pour chaque arrangement de processeurs, le nombre de tels arrangements étant exponentiel en la taille de la grille.
– Ne pas prendre en compte les contraintes géométriques induites par les deux dimensions de la grille conduit à des partitionnements irréguliers qui permettent un bon équilibrage de charge mais sont bien plus difficiles à mettre en œuvre.

Dans cette perspective, nous montrons que la distribution de tâches indépendantes sur un réseau unidimensionnel de processeurs n'est plus facile lorsque l'on tient en plus compte des communications.
Des travaux antérieurs incluent également l'importante littérature traitant de charges divisibles [3] : comme dans cet article, un travail important peut être arbitrairement divisé en plusieurs tâches, ces tâches étant assignées aux processeurs de telle sorte que le temps total d'exécution, i.e. la somme des temps de communication et de calculs, soit minimisé.

8. Conclusion

La principale limitation de la programmation sur plate-formes hétérogènes vient de la difficulté supplémentaire à équilibrer la charge. Les données et les calculs ne sont pas répartis équitablement entre les processeurs. Minimiser les surcoûts de communication devient alors une tâche ardue. Dans cet article, l'accent a été mis sur une modélisation réaliste de communications concurrentes dans des réseaux de grappes. Un résultat majeur est la NP-complétude du problème SHAREDRING. Plus que la preuve, le résultat lui-même est intéressant puisqu'il fournit une nouvelle preuve de la difficulté intrinsèque de la conception d'algorithmes hétérogènes. Une autre avancée importante de cet article est la conception d'une heuristique efficace, qui fournit une ligne de conduite pragmatique au concepteur de calculs scientifiques itératifs. L'importance d'une modélisation précise des communications, qui prend pleinement en compte les contentions, est clairement mise en évidence par les résultats expérimentaux. Notre heuristique rend possible l'implémentation efficace de calculs itératifs sur des grappes constituées de plusieurs ressources hétérogènes, ce qui constitue une alternative prometteuse à l'usage de super-ordinateurs coûteux.

Bibliographie

1. F. Berman. High-performance schedulers. In I. Foster and C. Kesselman, editors, *The Grid : Blueprint for a New Computing Infrastructure*, pages 279–309. Morgan-Kaufmann, 1999.
2. D. Bertsekas and R. Gallager. *Data Networks*. Prentice Hall, 1987.
3. V. Bharadwaj, D. Ghose, V. Mani, and T. Robertazzi. *Scheduling Divisible Loads in Parallel and Distributed Systems*. IEEE Computer Society Press, 1996.
4. R. Buyya. *High Performance Cluster Computing. Volume* 1 *: Architecture and Systems*. Prentice Hall PTR, Upper Saddle River, NJ, 1999.
5. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.
6. E. Deelman and B. Szymanski. Dynamic load balancing in parallel discrete event simulation for spatially explicit problems. In *PADS'98*, pages 46–53. IEEE Computer Society Press, 1998.
7. D. Katabi, M. Handley, and C. Rohrs. Congestion control for high bandwidth-delay product networks. In *Proceedings of ACM SIGCOMM 2002*, pages 89–102. ACM Press, 2002.
8. A. Legrand, H. Renard, Y. Robert, and F. Vivien. Load-balancing iterative computations in heterogeneous clusters with shared communication links. Research Report RR-2003-23, LIP, ENS Lyon, France, Apr. 2003.
9. H. Renard, Y. Robert, and F. Vivien. Static load-balancing techniques for iterative computations on heterogeneous clusters. Research Report RR-2003-12, LIP, ENS Lyon, France, Feb. 2003.
10. B. A. Shirazi, A. R. Hurson, and K. M. Kavi. *Scheduling and load balancing in parallel and distributed systems*. IEEE Computer Science Press, 1995.
11. A. Taylor and A. Hindmarsh. User documentation for KINSOL, a nonlinear solver for sequential and parallel computers. Tech. Rep. UCRL-ID-131185, Lawrence Livermore Nat. Lab., July 1998.
12. J. Watts and S. Taylor. A practical approach to dynamic load balancing. *IEEE Trans. Parallel and Distributed Systems*, 9(93) :235–248, 1998.
13. M.-Y. Wu. On runtime parallel scheduling for processor load balancing. *IEEE Trans. Parallel and Distributed Systems*, 8(2) :173–186, 1997.