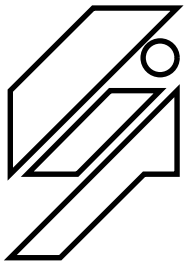
Compilation of data-parallel program for a network of workstations

Guy Cuvillier, Gil Utard



Laboratoire de l'Informatique du Parallélisme

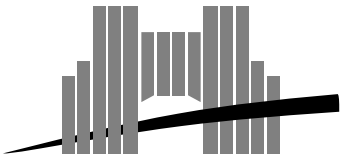
Ecole Normale Supérieure de Lyon
Unité de recherche associée au CNRS n°1398

Compilation de programmes data-parallèles pour un réseau de stations de travail

Guy Cuvillier et Gil Utard

November 1994

Research Report N^{O} 94-31



Ecole Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : (+33) 72.72.80.00 Télécopieur : (+33) 72.72.80.80
Adresse électronique : lip@lip.ens−lyon.fr

Compilation de programmes data-parallèles pour un réseau de stations de travail

Guy Cuvillier et Gil Utard

November 1994

Abstract

We describe the compilation and execution of data-parallel languages for networks of workstations. Executions of distributed programs on cluster of workstations must take into account the loads changing dynamically due to other users. We show that data-parallel programming model is a good candidate to include dynamics load balancing features. We present an implementation for a network of workstations with the data-parallel language C* and PVM.

Keywords: Data-parallelism; Compilation; Network of workstations; Dynamic load balancing

Résumé

Nous décrivons un schéma de compilation et d'exécution de programmes data-parallèles pour les réseaux de stations de travail. L'exécution de programmes distribués sur un ensemble de stations nécessite de prendre en compte leur variation de charge. Nous montrons que le modèle de programmation data-parallèle offre un cadre favorable au rééquilibrage dynamique de la charge. Nous présentons une implémentation de ces techniques pour un réseau de stations sous PVM pour le langage data-parallèle C*.

Mots-clés: Parallélisme de données ; compilation ; réseau de stations ; rééquilibrage de la charge

Compilation de programmes data-parallèles pour un réseau de stations de travail

Guy Cuvillier et Gil Utard*
LIP, URA CNRS 1398
ENS Lyon
46 Allée d'Italie
F-69364 Lyon Cedex 07 France

Novembre 1994

Table des matières

*. Ce travail a été soutenu par le Projet de Recherche Coordonnée du CNRS "Coopération, Concurrence, Communication" C^3, le contrat PRC/MRE ParaDigme ainsi que le contrat DRET 91/1180.

1 Introduction

Le ratio prix/performances des stations de travail ne cesse de diminuer et des technologies récentes (l'*alpha* de DEC ou le HP-PA) permettent de construire des machines de puissance inimaginable auparavant. Par ailleurs, nombre d'institutions se dotent de réseaux locaux dont les possibilités d'extension seront décuplées dans un proche avenir, grâce aux innovations telles que le FDDI.

Dans la pratique on observe que chaque station d'un réseau est utilisée la plupart du temps pour des applications interactives. Une expérience de l'université d'Ohio qui consistait à observer pendant trois mois un réseau constitué de 200 stations sans disque et de 18 serveurs, a montré que le taux d'inoccupation des ressources en calculs était de 90% avec une variance de 2,5% [12] ! La puissance de calcul et la capacité mémoire de tels réseaux sont pourtant conséquentes et offrent pour le calcul intensif une alternative économique séduisante aux super-calculateurs.

L'*hypercomputing* [4] est l'exploitation des ressources inutilisées par la machine parallèle virtuelle que constitue un réseau de stations. Des outils logiciels tels que PVM [1] et Linda [19] visent cet objectif.

Mais le problème récurrent du parallélisme demeure : quel modèle de programmation employer pour développer des applications ? Le modèle de programmation offert par PVM est essentiellement celui du passage de messages. Les inconvénients de ce modèle sont maintenant bien connus : absence de structures de données, mise au point délicate due au non-déterminisme, extensibilité limitée des applications. Le modèle proposé par Linda nécessite un système d'exécution sophistiqué [14]. Nous allons nous intéresser à une autre approche : le *parallélisme de données* [2].

Ce modèle dispose d'un certain *savoir-faire* hérité de la programmation des machines SIMD et du modèle algorithmique théorique PRAM. Autre avantage, une application data-parallèle s'étend de manière naturelle en fonction de la taille des données à traiter.

La compilation d'un langage data-parallèle pour un réseau de stations peut largement employer les techniques développées pour les machines MIMD à mémoire distribuée [8, 18]. Il existe d'ailleurs une implémentation de Hyper C pour réseaux de stations [9]. Dans les machines MIMD, les nœuds sont le plus souvent homogènes et entièrement dédiés à l'application. Pour un réseau de stations, ils sont de capacité variable, et chacun est généralement dédié à un utilisateur. Le contexte d'exécution est ici très particulier. Exécuter efficacement un programme parallèle sans spolier les utilisateurs nécessite alors d'introduire un mécanisme de redistribution automatique de la charge [4].

Nous rappellerons la technique de compilation par virtualisation des langages data-parallèles pour machines MIMD et montrerons qu'elle offre un cadre favorable à la redistribution dynamique de la charge. Nous présenterons une implémentation d'un compilateur C* pour PVM avec redistribution dynamique de charge et les premiers résultats obtenus.

2 Un schéma de compilation data-parallèle : la virtualisation

Dans le modèle de programmation data-parallèle, un problème est représenté par un domaine, typiquement des tableaux multidimensionnels. Ce sont les `collection` en Hyper C, ou les `shape` en C*. Par exemple, en calcul par éléments finis, un domaine correspond à un maillage de l'espace. On définit différentes variables parallèles sur ce domaine. Les composantes de ces variables sont accessibles en parallèle. Un programme est une composition séquentielle d'opérations *élément à élément* et d'opérations de *réarrangement*. Chaque opération est associée à un ensemble de composantes où elle sera appliquée : ces composantes sont dites *actives*.

Généralement les langages data-parallèles sont associés à un langage scalaire pour le calcul de la partie non parallèle de l'application. Par exemple C*, Hyper C et MPL sont des extensions data-parallèles de C ; MP-FORTRAN et CM-FORTRAN sont des extensions de FORTRAN.

Classiquement, le modèle d'exécution sous-jacent associe une machine SIMD *virtuelle* à chaque domaine, telle que la CM2 de TMC ou la Maspar de DEC. C'est une machine composée d'un processeur scalaire et d'une grille de processeurs élémentaires à mémoire distribuée correspondant au domaine. L'unité de contrôle diffuse de manière synchrone les instructions parallèles du programme à l'ensemble des processeurs élémentaires qui les exécutent ou non selon l'activité de la composante qu'il représente. Cette activité est représentée par le *contexte* des processeurs virtuels. Les opérations de réarrangement génèrent des communications entre les processeurs élémentaires. Le code scalaire est exécuté par le processeur du même nom.

La virtualisation est une technique de compilation courante des programmes data-parallèles. C'est une *simulation* de la machine SIMD virtuelle associée au programme sur une machine à topologie fixée [18]. En pratique, la taille du domaine est nettement supérieure à la taille de la machine physique. Chaque processeur physique traite plusieurs processeurs élémentaires (dit *virtuels*) de la machine émulée. Chaque processeur virtuel est représenté par son image mémoire et son contexte, i.e. son état au sens de la sémantique opérationnelle. L'ensemble des processeurs virtuels alloués à un processeur physique est déduit du code source et d'une fonction de *distribution* des domaines du programme sur la machine physique. Pour faciliter le calcul d'adresse des opérations de réarrangement, la fonction de distribution est généralement régulière (e.g. de type bloc). Sur une machine MIMD à mémoire distribuée, le code scalaire est généralement dupliqué sur l'ensemble des processeurs physiques pour des raisons d'efficacité.

Le code généré par la virtualisation est de type SPMD. L'exécution consiste en une succession de phases de calculs et de communications. Les phases de calculs correspondent aux opérations élément à élément du programme source, et sont implémentées par des *boucles de virtualisation*. Les phases de communications correspondent aux opérations de réarrangement du programme source, et sont implémentées par l'appel à des fonctions de communication spécifiques. De par la sémantique synchrone du modèle d'exécution, aucun processeur phy-

sique ne doit commencer une nouvelle phase de calcul tant qu'un autre processeur détenant des données nécessaires n'a pas encore terminé la phase de calcul précédente. Ceci est assuré dans le cas de machine MIMD à mémoire distribuée, d'une part par un flôt identique d'appel aux routines de communications spécifiques, indépendamment du flôt de données parallèle, et d'autre part par une implémentation des routines de communication qui conserve l'ordre d'émission/réception des messages.

Lors de l'exécution d'un programme obtenue par cette technique de compilation, on observe un couplage fort entre les processeurs physiques. La vitesse de l'application est cadencée par le plus lent. Avec cette technique de compilation, le langage data-parallèle doit être considéré comme un langage évolué (car parallèle) de bas niveau (la qualité de l'exécution est extrêmement sensible à l'écriture du programme).

Du fait de la sémantique séquentielle des programmes data-parallèles, l'analyse du flot des données est grandement facilitée, et les optimisations possibles sont basées sur le réarrangement de l'ordre des instructions parallèles. Ainsi il est possible d'augmenter la granularité des programmes obtenus par fusion des boucles de virtualisation, et de recourir au recouvrement des communications par les calculs. Le lecteur intéressé peut se référer à [18, 8, 5].

Les facteurs de déséquilibre interne à l'application sont dus en général à une non homogénéité des calculs sur l'ensemble du domaine et/ou à une mauvaise distribution du domaine sur l'ensemble des processeurs physiques. Elles sont de la responsabilité du programmeur, ou du compilateur amont qui a généré le code data-parallèle à partir d'un langage dit évolué.

3 Ordonnancement d'applications parallèles sur réseau de stations

L'ordonnancement d'applications parallèles sur les réseaux de stations se heurte d'une part à l'hétérogénéité du parc de machines disponibles, et d'autre part au contexte multi-utilisateur. Comme la charge des stations évolue selon des facteurs externes, seul l'ordonnancement dynamique peut-être efficace. De tels ordonnanceurs dynamiques sont aujourd'hui disponibles. Du fait des latences élevées des communications, la granularité des applications doit être élevée. Deux grands types de granularité sont actuellement traités :

- une grosse granularité au niveau « job », avec la sélection de site distant pour l'exécution de commande shell tel que `lsf` d'Utopia ou certain `make` parallèles ;
- une granularité moyenne au niveau processus ou sous-programme, avec le placement d'une application parallèle décrite par un graphe de tâches.

La granularité des applications étant élevée et non divisible, les algorithmes d'ordonnancement dynamique ne cherchent pas à atteindre un équilibrage «parfait» de la charge, mais plutôt à partager celle-ci sur l'ensemble des stations. L'objectif est de ne pas surcharger ou laisser inoccupées des stations.

Le cas des programmes data-parallèles

Un programme data-parallèle est considéré comme étant de granularité fine. La technique de virtualisation permet d'obtenir dans la plupart des cas des programmes SPMD de granularité moyenne. La granularité escomptée est fonction du rapport entre le nombre d'opérations de réarrangement et celui des opérations élément à élément, de la taille des domaines, et du nombre de processeurs physiques.

L'hypothèse principale est que la charge de calcul d'un processeur physique est proportionnelle au nombre de processeurs virtuels émulés. On est alors dans un cadre idéal pour le rééquilibrage dynamique de l'exécution du programme data-parallèle. Redistribuer la charge est équivalent à redistribuer les données, i.e. les processeurs virtuels, entre les processeurs physiques. Par rapport aux applications parallèles décrites par un graphe de tâches indivisibles, on a ici un plus grand degré de finesse dans l'équilibrage de la charge des processeurs.

Les phases de communications sont un moment propice pour effectuer le rééquilibrage, elles correspondent à la transition entre deux phases de calcul sur les processeurs physiques. L'état des processeurs virtuels étant parfaitement connu, ils peuvent changer de processeur physique hôte.

Un processeur virtuel est représenté par sa mémoire et son contexte dans le processeur physique. À chaque redistribution, ces informations d'état sont rassemblées pour chaque processeur virtuel migrateur.

4 Implémentation

Le support utilisé pour cette réalisation est le compilateur C*de l'Université du New Hampshire (UNH-C*). Ce compilateur est délivré avec des bibliothèques de communication pour le Delta d'Intel, et comporte également une bibliothèque de simulation pour station de travail.

Ce compilateur a tout d'abord été porté sur un réseau de stations de travail sous PVM. La bibliothèque de communication écrites pour le Delta a été détournée : les appels système de communication du Delta ont été remplacés par des appels équivalents aux primitives de communication PVM.

Pour simplifier la mise en œuvre, une gestion centralisée a été adoptée. Ce choix permet de plus de superviser l'exécution des programmes data parallèles. Un programme maître baptisé «lanceur» engendre autant de tâches PVM esclaves que demandé par l'utilisateur. Chacune de ces tâches peut être affectée à une station propre.

Une fois les tâches esclaves lancées, le maître se met en attente des messages de résultats, qui attestent que le programme s'est bien déroulé. Si une erreur se produit, le signal est récupéré dans la tâche esclave et un label de message spécial avertit le maître. Celui-ci prend alors en charge la suppression de l'ensemble des tâches esclaves et l'arrêt total du programme.

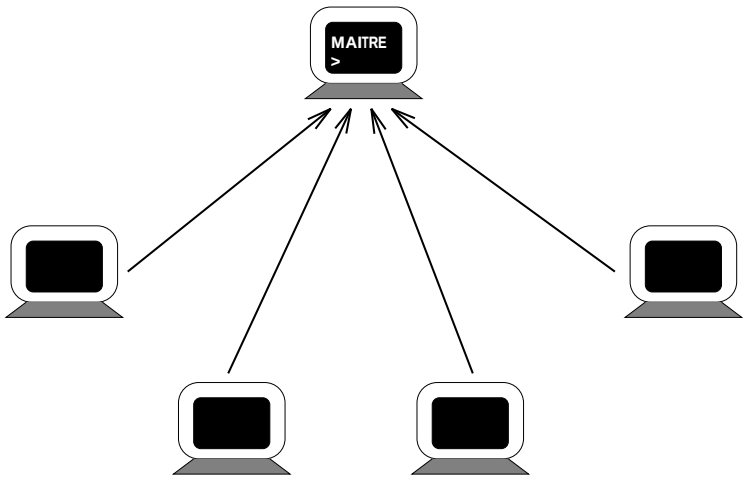


FIG. 1 - *À chaque communication, collecte des informations d'efficacité.*

4.1 Description générale

Le fonctionnement des tâches esclaves est le suivant.

- À chaque passage dans une fonction de communication les esclaves envoient au maître leur mesure d'efficacité (figure 1). Notons que l'envoi de ces informations n'est ici systématique que parce que nous tenons à assurer une veille assidue sur le déroulement du programme ; ceci pour permettre une bonne observation de l'exécution. L'implémentation réalisée permet de fixer une fréquence d'émission des informations des esclaves.
- À une phase de communication P convenue ils attendent au contraire la décision du maître concernant une éventuelle procédure d'adaptation de la charge (figure 2).

Le maître réalise les opérations suivantes.

- Il concentre les informations d'efficacité délivrées par les esclaves (figure 1).
- À la phase de communication P convenue, il évalue une nouvelle distribution de processeurs virtuels au vu des informations d'efficacité collectées. Il décide du rééquilibrage et transmet sa décision (figure 2).
  - Dans l'affirmative, il le signale aux esclaves en attente de sa réponse, en associant à son message des consignes pour réaliser la nouvelle répartition. Ceux-ci réalisent alors les échanges de données nécessaires (figure 3).
  - Si en revanche une redistribution n'est pas rentable, il confirme aux esclaves qu'ils peuvent continuer à travailler avec l'ancienne distribution.

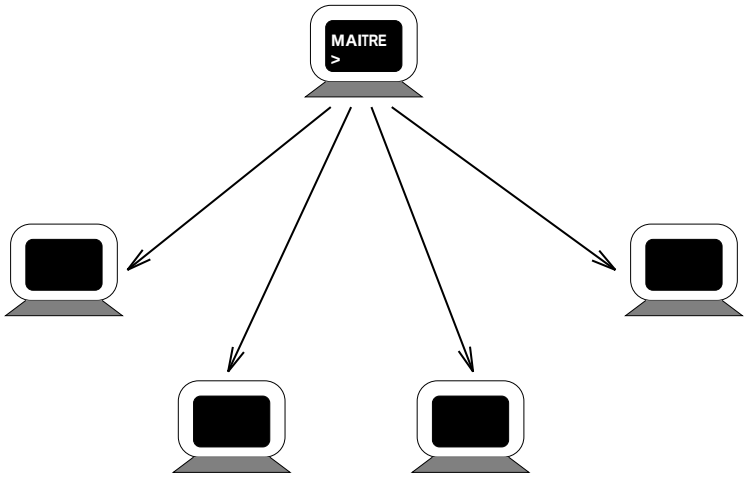


FIG. 2 - *À une phase de communication convenue, propagation de la décision du maître.*

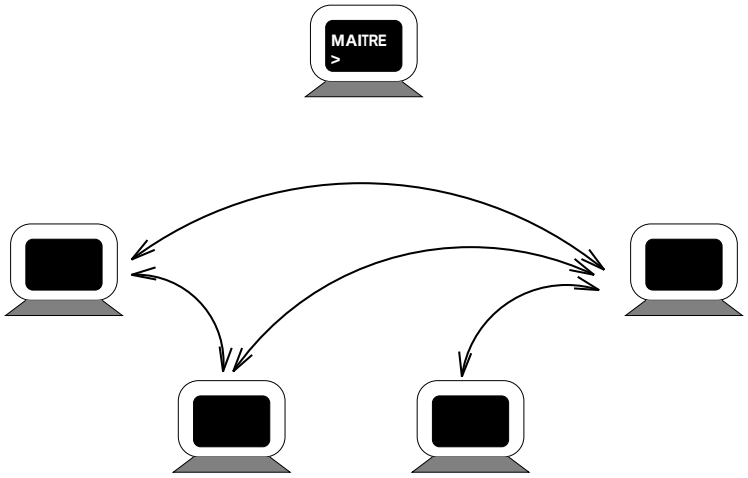


FIG. 3 - *Échanges de données entre les esclaves pour réaliser l'équilibrage demandé.*

4.2 Distribution des processeurs virtuels

La distribution est équivalente à un partitionnement du domaine. Dans la pratique, les partitionnements sont de type bloc. Deux types de distribution par blocs des domaines sur les stations de travail sont considérés :

- une dimension : le domaine est décomposé en bandes ;
- deux dimensions : le domaine est décomposé en rectangle *recti-linéaire*, deux pavés adjacents en ligne (resp. en colonne) ont le même nombre de colonnes (resp. de lignes).

Le partitionnement en deux dimensions permet d'exploiter l'organisation des réseaux en «grappes». Chaque grappe possédant son propre bus de communications on augmente ainsi la bande passante totale. Mais ce type d'organisation n'est complètement exploitable que s'il existe un axe de communication privilégié dans le domaine de l'application (figure 4). Sinon, les communications étant égales dans les deux dimensions, les passerelles constituent un goulot d'étranglement dans l'échange des données entre les grappes. Le partitionnement

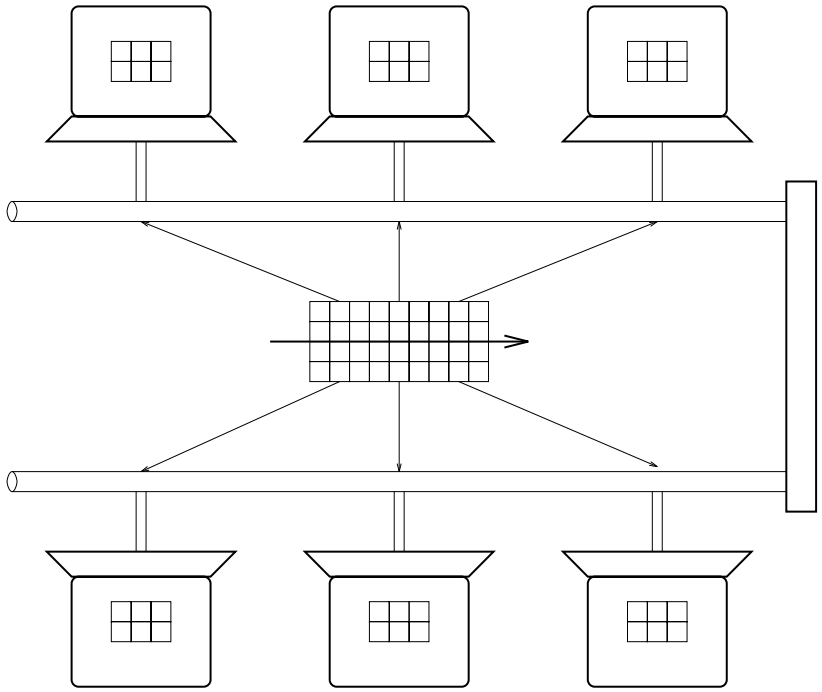


FIG. 4 - *Exploitation d'un réseau en grappes. Il existe dans le domaine de l'application un axe de communication privilégié (la flèche). Le domaine est distribué pour exploiter au mieux la bande passante du réseau.*

en n-dimension peut de la même manière être envisagé pour exploiter une arborescence de grappes de stations. Une arborescence de plus de deux niveaux de grappes de stations ne

rentre plus dans le cadre des réseaux locaux. Il s'agit plutôt des réseaux hiérarchiques de grande taille (interurbains notamment). Dans ce cas on a une organisation de type «thin-tree» du réseau de communications, et le type d'application envisageable est limité [3]. Elle nécessite d'introduire des mécanismes de tolérance aux pannes et demande une coordination de l'administration des systèmes [7].

En fonction d'une distribution initiale des processeurs virtuels, chaque station réserve à l'initialisation l'espace mémoire nécessaire pour les accueillir, plus une fraction supplémentaire pour permettre la réalisation des rééquilibrages futurs.

4.3 Calcul et réalisation d'une redistribution

Le calcul d'un rééquilibrage est basé sur une connaissance globale du système. En général, les réseaux disposent d'un médium de type bus qui permet d'obtenir cet état à un faible coût en terme de communication. L'avantage d'une connaissance globale est qu'une redistribution optimale peut rapidement être calculée. Sous l'hypothèse que la charge de travail est proportionnelle au nombre de processeurs virtuels émulés, une distribution au prorata des vitesses de chaque station est optimale.

La migration des processeurs virtuels se fait de telle sorte que la structure du découpage en blocs soit conservée. Le mouvement des processeurs virtuels s'effectue par bandes. Cette restriction diminue le degré de finesse dans le rééquilibrage de la charge mais si la structure régulière de la répartition n'est pas conservée, alors l'adressage de processeurs virtuels se complexifie. Ici l'adressage est obtenu simplement par le maintien d'une table d'adressage indiquant les bornes d'adresse de chaque station.

Le rééquilibrage s'effectue pour la `shape` C* courante : ce sont toutes les variables parallèles liées à la shape qui devront migrer. Ces variables sont de deux sortes : les variables utilisateur et les variables temporaires générées à la compilation.

Le compilateur a été modifié pour conserver trace de toutes les variables parallèles liées à chaque shape. De plus, le contexte d'activité était géré classiquement par des tableaux de booléens. Un changement de contexte revenait à empiler un nouveau tableau décrivant le nouveau contexte ou à dépiler le tableau courant. La taille de cette information était donc variable. Les travaux de Levaire ont montré que la gestion de contexte par piles de tableaux pouvait être remplacée par l'emploi d'un tableau de compteurs unique [13]. Afin de réduire la taille des informations de contexte à traiter, nous avons introduit une représentation des contextes par compteurs dans C*.

4.3.1 Détermination de la vitesse d'une station de travail

Classiquement, la détermination de la vitesse d'une machine est une fonction complexe de la *puissance* et de la *charge* de chaque machine.

- La puissance doit être définie par une liste de critères qualitatifs telles que le type de processeur, la fréquence d'horloge de ce dernier, la quantité de mémoire.
- Les critères de mesure de la charge de machine sont délicats à appréhender. La méthode la plus classique est d'observer les différentes files d'attente de processus du système d'exploitation. Une machine chargée a une file d'attente de processus prêts a être exécutés longue. L'inconvénient de cette méthode est qu'elle ne prend pas en compte la nature des processus qui vont être exécutés. Une machine avec une longue file d'attente de processus légers est-elle moins chargée qu'une machine avec une file d'attente courte de processus lourds ?

C'est une méthode de mesure plus simple qui est implémentée. Elle consiste à relever le temps réel écoulé dans les phases de calculs de l'application data-parallèle. L'avantage d'une telle mesure est qu'elle intègre les paramètres de la charge de la machine et de sa puissance intrinsèque. Seuls les temps des phases de calculs sont considérés. Si l'on intègre aussi les temps des phases de communication, on obtiendra des temps quasiment identiques pour toutes les stations, du fait du fort couplage des processeurs physiques.

La vitesse d'une station de travail est le nombre de processeurs virtuels émulés par unité de temps. C'est le rapport entre le nombre de processeurs virtuels émulés et le temps «réel» écoulé dans les phases de calculs. Seuls les processeurs virtuels actifs sont comptabilisés. Prendre en compte les processeurs virtuels inactifs intègre le déséquilibre possible dû à l'application, ce qui n'est pas le propos de cette étude.

4.3.2 Rééquilibrage dans le cas mono-dimensionnel.

La largeur de la bande de processeurs virtuels détenue par chaque station est déterminée au prorata de leur vitesse. Le calcul de la distribution doit aussi tenir compte des contraintes au niveau de la mémoire des stations. Si la distribution précédente est telle qu'une ou plusieurs stations voient leur capacité dépassée, alors le reliquat sera distribué de la même manière sur les stations restantes (et ainsi de suite si de nouvelles stations sont saturées).

La connaissance de la nouvelle et de l'ancienne distribution étant globale, chaque station peut aisément déterminer les communications nécessaires pour atteindre la nouvelle distribution. Les primitives d'envoi non-bloquant de messages et celle de réception bloquante disponibles dans les protocoles de communication sont utilisées. Ainsi, il n'y a pas de risque d'*étreintes fatales* dans le protocole de rééquilibrage.

Plus précisément, chaque station effectue les étapes suivantes.

- Au vu de la nouvelle distribution, elle détermine l'intervalle de processeurs physiques

désormais propriétaires de la section de données qu'elle possédait. Les envois nécessaires des sous-sections sont effectués.

- La partie des données restantes est décalée en mémoire.
- L'ancienne distribution procure l'intervalle de processeurs physiques qui doivent fournir les nouvelles données dont elle a la responsabilité. Elle effectue les réceptions successives des nouvelles sous-sections.

Pour réduire le coût des décalages mémoires, ceux-ci sont recouverts par le temps de communication.

À l'issue des échanges de données, il ne reste qu'à actualiser la table d'adressage pour rendre compte de la nouvelle distribution, ainsi qu'à «configurer» le processeur physique pour qu'il émule le nouveau nombre de processeurs virtuels résultant de la redistribution.

4.3.3 Rééquilibrage dans le cas bi-dimensionnel.

Le partitionnement devant-être rectangulaire *recti-lineaire*, une distribution «exacte» au prorata de la vitesse des stations n'est plus possible. Les blocs adjacents ayant le même nombre de lignes et de colonnes, l'attribution entre les stations est inter-dépendante.

Considérons la répartition d'un domaine bi-dimensionnel de L lignes et C colonnes, i.e. $N = L \times C$ processeurs virtuels, en n tranches de lignes et p tranches de colonnes. Soit $v_{i,j}$ avec $1 \leq i \leq n$ et $1 \leq j \leq p$ la vitesse (en nombre de processeurs virtuels émulés par unité de temps) de la station détenant le bloc (i,j). Soit $n = (n_{i,j})$ le nombre de processeurs virtuels du bloc (i,j) d'une distribution arbitraire. Le temps de calcul t_n d'une distribution n est égal à :

$$t_n \quad = \quad max_{i,j}\ (n_{i,j}/v_{i,j})$$

La distribution qui minimise ce temps de calcul est celle où chaque station détient un nombre de processeurs virtuels au prorata de sa vitesse, i.e. quand $n_{i,j} = \frac{N \times v_{i,j}}{\sum_{k,l} v_{k,l}}$. Le temps de calcul optimal t_o est alors égal à $\frac{N}{\sum_{k,l} v_{k,l}}$.

Le problème est de trouver une partition n *recti-linéaire* du domaine tel que le temps de calcul soit minimal. La distribution n est décrite par deux vecteurs $\vec{l} = (l_i)_{1 \leq i \leq n}$ et $\vec{c} = (c_j)_{1 \leq j \leq p}$ tel que $\sum_i l_i = L$ et $\sum_j c_j = C$, représentant le partitionnement des lignes et des colonnes. Le nombre de processeurs virtuels du bloc (i,j) est égal à $n_{i,j} = (l_i \times c_j)$.

Il s'avère que le découpage optimal est difficile à obtenir. Il n'est pas égal à un découpage des lignes (resp. des colonnes) tel que la part relative de chaque tranche de lignes (resp. de colonnes) soit la somme des vitesses relatives des stations de la tranche. Considérons l'exemple suivant d'un découpage d'un domaine 30×30 en 3 lignes et 3 colonnes, sur des

stations dont les vitesses sont données par la matrice suivante :

$$\begin{pmatrix} 2 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Une distribution telle que $\vec{l} = (12\ 9\ 9)$ et $\vec{c} = (12\ 9\ 9)$, i.e. au prorata de la somme des vitesses des stations pour chaque ligne et colonne, donnerait comme matrice de temps de calcul :

$$\begin{pmatrix} 74 & 108 & 108 \\ 108 & 81 & 81 \\ 108 & 81 & 81 \end{pmatrix}$$

soit un temps maximum de 108. Alors qu'une equi-distribution avec $\vec{l} = (10\ 10\ 10)$ et $\vec{c} = (10\ 10\ 10)$ donnerait comme matrice de temps de calcul :

$$\begin{pmatrix} 50 & 100 & 100 \\ 100 & 100 & 100 \\ 100 & 100 & 100 \end{pmatrix}$$

soit un temps maximun de 100.

La distribution est calculée par une méthode itérative. Elle consiste à fixer une distribution dans une dimension et à déterminer la distribution optimale dans l'autre dimension, puis de calculer la meilleure distribution dans l'autre dimension avec la nouvelle distribution obtenue. L'algorithme est itéré jusqu'à stabilisation.

Plus formellement, si on part d'une distribution des lignes arbitraire $\vec{l}^{(0)}$, la distribution successive des colonnes $\vec{c}^{(k)}$ et des lignes $\vec{l}^{(k)}$ est donnée par les algorithmes suivants.

$$\vec{c}^{(k)} \begin{cases} 1 \leq j \leq p : \alpha_j^{(k)} = min_{1 \leq i \leq n}(\frac{v_{i,j}}{l_i^{(k-1)}}) & \textit{Nombre de colonnes de p.v. traitées sur la colonne } j \textit{ par unité de temps.} \\ 1 \leq j \leq p : c_j^{(k)} = \frac{C \times \alpha_j^{(k)}}{\sum_{1 \leq l \leq p} \alpha_l^{(k)}} & \textit{Distribution des colonnes de p.v au prorata des vitesses de chaque colonne.} \end{cases}$$

$$\vec{l}^{(k)} \begin{cases} 1 \leq i \leq n : \beta_i^{(k)} = min_{1 \leq j \leq p}(\frac{v_{i,j}}{c_j^{(k)}}) & \textit{Nombre de lignes de p.v. traitées sur la ligne } j \textit{ par unité de temps.} \\ 1 \leq i \leq n : l_i^{(k)} = \frac{L \times \beta_i^{(k)}}{\sum_{1 \leq l \leq n} \beta_l^{(k)}} & \textit{Distribution des lignes de p.v au prorata des vitesses de chaque lignes.} \end{cases}$$

La suite des temps de calcul obtenue à chaque itération est monotone décroissante (la distribution au prorata des vitesses assure que le temps de calcul ne peut pas augmenter). L'algorithme converge car l'espace de recherche est fini. La solution donnée est un optimum local. Dans la pratique l'algorithme converge rapidement.

Pour réaliser une nouvelle distribution donnée, le mouvement des processeurs virtuels entre les stations peut devenir très complexe. La surface d'interaction d'une station pour un échange de processeurs virtuels est définie comme étant le nombre de stations qui seront impliquées dans les communications. Une variation forte de la distribution peut entraîner une augmentation forte de cette surface, entraînant une augmentation du temps de communication et compliquant la gestion de la mémoire. La solution adoptée est de choisir, dans la nouvelle distribution calculée, une seule dimension (ligne ou colonne) où sera effectuée un rééquilibrage (celle qui minimise le plus le temps de calcul). Le mouvement des processeurs virtuels est effectué de la même manière que dans le cas mono-dimensionnel (on raisonne en termes de lignes ou colonnes de processeurs virtuels).

4.4 Déclenchement d'une redistribution

La prise de décision d'un rééquilibrage est décidée à des phases de communications convenues à l'avance. À chacune de ces phases, un rééquilibrage est décidé ou non, et la prochaine phase de prise de décision est déterminée. Les phases sont choisies de telle manière que le temps entre chaque prise de décision soit constant.

Le gain d'une redistribution ne doit pas être absorbé par son coût. Ce dernier est la somme du coût de la détermination d'une nouvelle distribution, qui peut être réduit par un mécanisme d'anticipation, et de celui de la migration des processeurs virtuels.

4.4.1 Détermination de la nouvelle phase de prise de décision

Une constante de temps k représente le temps voulu entre deux étapes de rééquilibrage. Le maître reçoit régulièrement les messages de charge des esclaves. Il calcule le temps moyen entre deux messages émanant d'un même processeur et se sert de cette valeur pour calculer la prochaine phase de rendez-vous P, i.e. le nombre de messages charges que devra envoyer chaque processeur esclave avant de ce mettre en attente des informations de rééquilibrage envoyées par le maître.

$$P = \frac{k}{\text{Temps moyen inter-messages}}$$

Cette valeur sera communiquée aux esclaves en même temps que la décision de rééquilibrage.

4.4.2 Réduction du coût de la détermination d'une distribution

Dans le coût de la détermination d'une distribution on a celui de la constitution de l'état global du système et celui du calcul de la distribution. Le calcul de la distribution est peu coûteux car les algorithmes sont simples. La constitution de l'état global nécessite la diffusion d'informations par chaque station, i.e. un coût de communication, et d'attendre que toutes les stations aient diffusé leur information, i.e. un coût de déséquilibre.

Le déséquilibre implique une attente des stations lentes par les plus rapides. Une anticipation de la diffusion de la décision du maître, i.e. dans une phase de communication précédente, permet d'une part de recouvrir le coût de communication par du calcul utile, et d'autre part d'entamer, dans la mesure du possible, le processus de migration des processeurs virtuels entre les stations les plus rapides.

Ainsi, le maître ne calculera et ne diffusera pas la décision d'un rééquilibrage à la phase de communication P calculée auparavant, mais au moins dans une phase de communication précédente.

Pour savoir si l'anticipation précédente était suffisante, il suffit d'observer s'il n'y a pas eu d'attente de décision du maître par un esclave. Les esclaves communiquent au maître leur temps d'attente éventuel. Le calcul du nombre de phases à anticiper s'exprime donc par le quotient du temps maximum d'attente des esclaves par ce temps moyen inter-phase de communication.

$$anticipation = \frac{\text{Temps d'attente maximum}}{\text{Temps moyen inter-messages}}$$

4.4.3 Critère de décision d'un rééquilibrage

Le rééquilibrage doit être déclenché quand le coût temporel de la migration des processeurs virtuels est inférieur au gain de temps de calcul escompté. Le coût temporel de la migration des processeurs virtuels peut être évalué à partir de la vitesse de communication. Celle-ci est évaluée lors des phases d'équilibrage précédentes, à l'instar de la vitesse de calcul :

$$vitesse_{comm} = \frac{\text{nombre de processeurs migrés}}{\text{temps écoulé pour l'équilibrage}}$$

La valeur de cette vitesse est réactualisée à chaque équilibrage. Le coût temporel de l'équilibrage de charge à venir peut être approximé par :

$$tempsEquilibrage = \frac{\text{nombre de processeurs virtuels migrés}}{vitesse_{comm}}$$

Le gain temporel de calcul escompté correspond à la différence entre les temps maximaux d'exécution pour la nouvelle distribution (d_i) et l'ancienne distribution (d'_i). Pour chaque station i, f_i est sa vitesse :

$$gainTemporelCalculs = \max_i(\frac{d_i}{f_i}) - \max_i(\frac{d'_i}{f_i})$$

Un rééquilibrage aura lieu si : $tempsEquilibrage < gainTemporelCalculs$

5 Expérimentation

Comme nous sommes partis d'une version *alpha* du compilateur C*de l'université du New Hampshire, nous n'avons pas cherché à obtenir des performances de calculs. Les expériences

que nous avons menées avaient pour but de valider deux principales idées : l'exploitation de la bande passante offerte par une organisation en grappes du réseau, et le mécanisme d'adaption à la variation de charge des stations.

5.1 Exécution dans un réseau en grappe

Les expériences se placent dans une situation de communications intensives. Le programme C* consiste à effectuer un grand nombre de décalages dans une `shape` de dimension deux, dont une dimension est privilégiée en communication. Le programme effectue alternativement r décalages dans la dimension privilégiée, puis un décalage dans l'autre dimension. La taille de la `shape` est égale au nombre de stations : chaque station ne gère qu'un processeur virtuel, ceci afin de réduire le coût des boucles de virtualisation. Chaque processeur virtuel détient un tableau d'une taille fixé : on a une variable parallèle de tableaux. Chaque opération de décalage génère, pour chaque station, un envoi de tableau vers la station détenant le processeur virtuel voisin sur la `shape`.

Les expériences ont été exécutées alternativement sur six stations d'une même grappe, puis sur deux grappes de trois stations. La `shape` est distribuée de telle manière que les communications dans la dimension privilégiée restent dans la même grappe.

La première expérience a consisté à observer les différents débits (sur une grappe et deux grappes) selon la taille des messages. Toutes les communications se font dans la dimension privilégiée. Les débits obtenus sont présentés sur la figure 5 pour des messages de tailles de 1Ko, 2Ko, 4Ko et 6Ko. Ceci nous permet d'évaluer la bonne taille des messages pour masquer «l'effet paquet» dû au différentes couches de communication PVM/TCP-IP. On voit que de bons débits peuvent être obtenus avec des messages de 4Ko.

La seconde expérience a consisté à observer l'augmentation du débit selon le rapport de communication entre les deux dimensions. La taille des messages est fixé à 4Ko. Les résultats sont présentés dans la figure 6. Pour un rapport de 1/1, la passerelle constitue un goulot d'étranglement. Mais dès le rapport 2/1 le débit maximun théorique d'une grappe est dépassé.

5.2 Adaptation aux variations de charges

Dans un premier cadre d'expérimentation nous nous sommes placés dans un réseau de stations isolées où nous ajustions les charges à volonté. Nous avons adopté une stratégie de rééquilibrage gloutonne, une version restreinte du mécanisme de prise de décision présenté en 4.4.3 : dès qu'une variation de charge dépasse un certain seuil (au moins 5% de processeurs virtuels migrent), la redistribution est effectuée. Ceci afin de ne pas masquer le comportement du réseau par une politique trop raffinée. Les tests portent sur la multiplication de matrices ligne par ligne. Pour une matrice d'ordre N, le domaine est un tableau mono-dimensionnel de N composantes comportant chacune un tableau de taille N. Pour p processeurs, la complexité de l'algorithme est en N^3/p et génère $N \times p$ communications de taille N. La granularité est

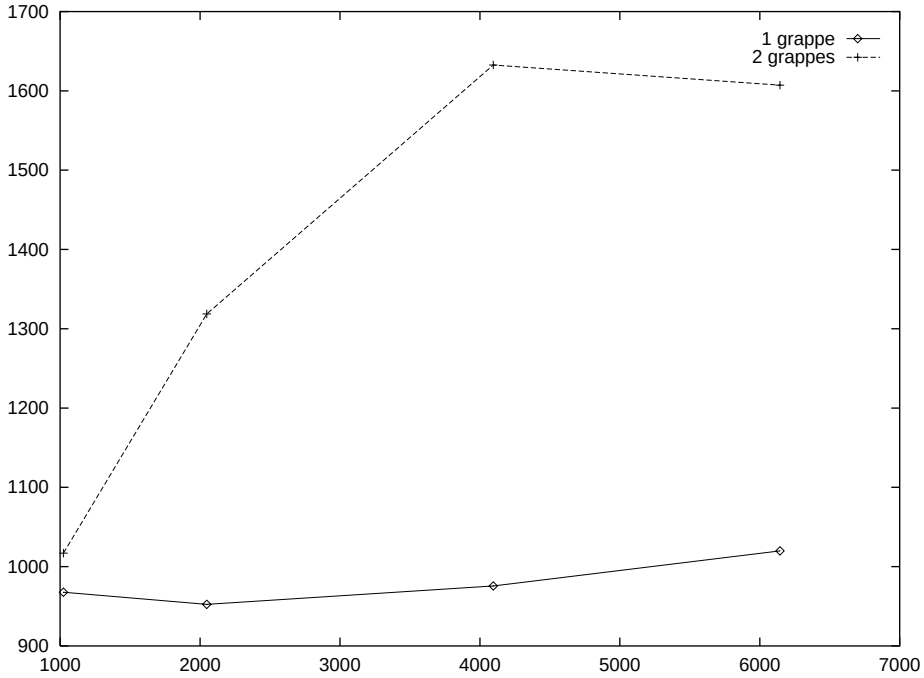


FIG. 5 - *Débit (Ko/s) selon la taille des messages sur une ou deux grappes de stations*

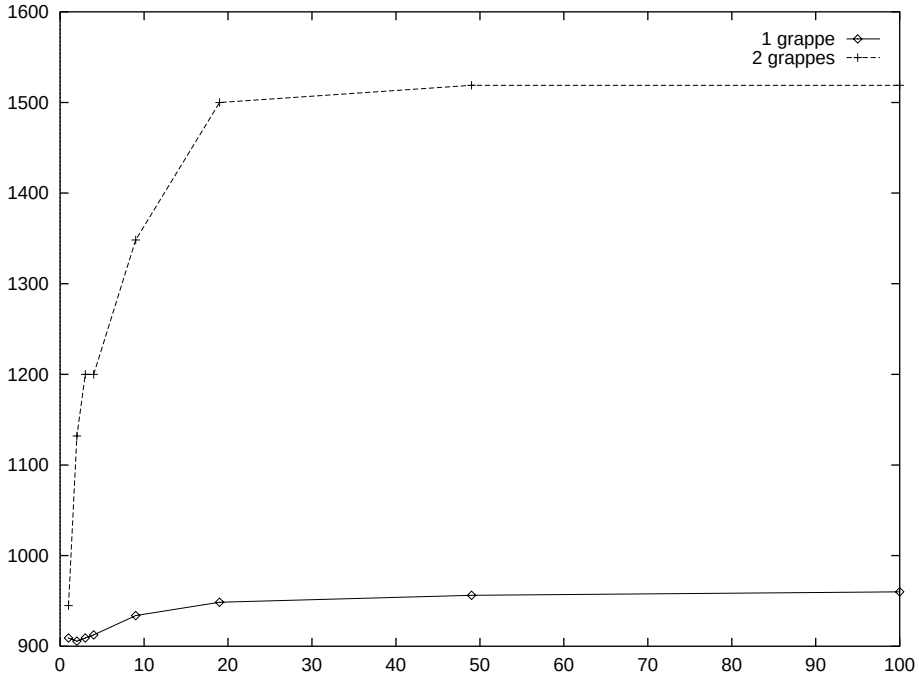


FIG. 6 - *Débit (Ko/s) selon le rapport de communication entre la dimension privilégiée et la non privilégiée en communication*

assez forte pour réduire l'influence des communications.

La première expérience a consisté à exécuter le programme sur deux stations non perturbées de puissances différentes. Le gain obtenu par le rééquilibrage dynamique correspond alors à un rééquilibrage statique au prorata des performances des machines. Notre méthode permet de prendre en compte l'*hétérogénéité* des ressource de calcul disponibles sur un réseau.

L'expérience suivante (tableau 1) consistait à observer le comportement dans un réseau de stations perturbées. Le premier test devait apprécier le comportement de notre dispositif dans un milieu de type interactif ; les perturbations générées aléatoirement étaient de courte durée (30 secondes au maximum) et fréquentes. Le deuxième cadre d'essais devait jauger notre outil face à des perturbations régulières et de longue durée dues au lancement de traitements par lot.

TAB. 1 - *Multiplication de matrices avec et sans rééquilibrage (minutes:secondes)*

	Environnement interactif						Environnement batch					
	Matrices 400×400			Matrices 800×800			Matrices 400×400			Matrices 800×800		
p	**Sans**	**Avec**	Δ	**Sans**	**Avec**	Δ	**Sans**	**Avec**	Δ	**Sans**	**Avec**	Δ
2	3:11	3:30	-9%	27:48	30:14	-8%	4:07	2:52	30%	38:32	38:07	1%
4	2:09	2:12	-2%	14:06	13:00	8%	2:11	1:30	31%	17:00	11:39	31%
8	1:07	1:04	4%	07:53	07:21	7%	1:11	0:50	29%	08:40	05:30	36%

On constate que les gains obtenus lors du premier test sont faibles. Dans le cas des deux nœuds, on observe même une augmentation du temps d'exécution. Le protocole induit dans cette situation une augmentation du couplage des machines. Pour des calculs long (800×800), les rééquilibrages sont fréquents. Ce phénomène est dû aux perturbations courtes et nombreuses. En moyenne cependant, chaque station subit une même somme de perturbations et le rééquilibrage n'apporte rien. On constate alors que le protocole de rééquilibrage n'est pas trop coûteux, mais qu'il est nécessaire d'intégrer un mécanisme d'évaluation de l'opportunité de rééquilibrage, comme presenté dans la section 4.4.3. Le protocole consomme inutilement les ressources en communication du réseau.

Dans notre deuxième expérience, seule la moitié des machines était perturbée. Les tâches parasites étaient longues mais l'instant de déclenchement était propre à chaque site. Les gains deviennent ici appréciables. La perturbation étant plus régulière, le nombre de rééquilibrages est en général moindre. Le cas du produit de matrices 800 par 800 pour 2 stations a révélé un phénomène de «ping-pong». Ceci viendrait à notre sens d'une variation de priorité des processus induite par UNIX dans sa stratégie d'ordonnancement ; sur une machine perturbée, l'ordonnanceur local donne alternativement la priorité au perturbateur puis à notre application. La distribution des données qui devrait en théorie se stabiliser varie alors suivant deux états.

6 Conclusion et travaux concomitants

Dans cette étude nous avons présenté une technique d'exécution de programmes data-parallèles sur réseau de stations. Le langage source est impératif et orienté collection. la technique de compilation est la virtualisation. L'exécution prend en compte les variations de charge des différentes stations par une redistribution dynamique des données. Une exploitation possible de l'organisation des réseaux en grappes, augmentant la bande passante de communication, a été présentée et expérimentée.

Les programmes data-parallèles offrent un cadre favorable au rééquilibrage de la charge : les charges sont définies par la distribution des données et les calculs sont identiques sur l'ensemble des données. On a un *rééquilibrage dynamique de la charge dirigé par les données* de mise en œuvre plus aisée et plus fine que le rééquilibrage classique basé sur les graphes de processus. Cette idée de rééquilibrer par les données a déjà été émise par Dekeyser, Fonlupt et Marquet qui proposent d'intégrer des mécanismes de redistribution des données au niveau de l'application même [6]. De Keyser et D. Roose avaient déjà remarqué que l'on pouvait exploiter les phases de communication des programmes data-parallèles pour opérer un rééquilibrage par redistribution des données [11]. Ils appellent cette technique le rééquilibrage *quasi-dynamique*, et proposent différents algorithmes de calcul à partir d'un graphe de dépendance des équilibrages à effectuer dans les différentes communications. Ce rééquilibrage est donc lié au déséquilibre de l'application.

D'autres travaux existent sur l'utilisation de langages data-parallèle sur réseau de stations. En terme de performances on peut citer les travaux de Mosberger, Turner et Peterson sur la compilation de C^* (version de TMC) sur un cluster de stations HPPA connectées par un réseau FDDI [5]. Leur compilateur est capable de réordonner la suite d'opération parallèles du programme source et de générer du code où les communications recouvrent les calculs. Ils obtiennent une meilleur efficacité comparativement au code généré pour une CM5. Il y a aussi les travaux de Nedeljković et Quinn sur la compilation de Data-Parallel C qui intègrent les premiers l'équilibrage de la charge par migration des processeurs virtuels [15]. Seuls les distributions en une dimension étaient considérées, et n'exploitaient pas les possibilités offertes par une organisation en grappe des stations.

L'algorithme de partitionnement rectilineaire en deux dimensions est inspiré des travaux de Nicol [16]. Ce dernier a de plus montré que le même problème en trois dimensions est NP. Nous aurions pu utiliser les partitionnements recursifs des domaines proposés par Kaddoura, Rank et Wang qui permettent de mieux exploiter l'hétérogénéité des réseaux [10]. En fait nous avons considéré les partitionnements rectilinéaires pour modifier au minimun le compilateur de l'Université du New Hampshire, conçu initialement pour les architectures homogènes.

L'équilibrage de charge que nous avons décrit est fonction de facteurs *externes* à l'application : disparité des puissances des machines et variations de leur charge. L'idée principale est d'affranchir l'utilisateur de l'hétérogénéité d'un réseau en lui présentant des machines SIMD virtuelles (ou *templates* pour reprendre la terminologie d'HPF) homogènes. On pourrait ainsi facilement employer les techniques de Dekeyser, Fonlupt et Marquet précédemment citées pour intégrer les facteurs internes de déséquilibre de l'application.

Seul l'aspect calculatoire est pris en compte dans notre mécanisme. Il serait intéressant d'intégrer aussi l'aspect communication, qui est souvent le goulot d'étranglement des réseaux de stations. L'annexe de ce rapport présente une première formulation du problème.

Remerciements. Nous remercions Phil Hatcher d'avoir mis à notre disposition le compilateur UNH-C*.

Références

[1] A. Beguelin, J. Dongarra, G.A. geist, R. Manchek, and V.S. Sunderam. A user's guide to PVM parallel virtual machine. TN, Oak Ridge National Laboratory, June 1992.

[2] L. Bougé. Modèle de programmation à parallélisme de données : une perspective sémantique. *Technique et science informatiques*, 12(5):541–562, 1993.

[3] C.H. Cap. Massive Parallelism with Workstation Clusters – Challenge or Nonsense? Technical Report IFI-TR 94.01, Dept. of Comp. Scs., University of Zurich, December 1993.

[4] C.H. Cap and V. Strumpen. Efficient parallel computing in distributed workstation environments. *Parallel Computing*, 19:1221–1234, 1993.

[5] C.J.Turner, D. Mosberger, and L.L. Peterson. Cluster-C*: Understanting the Performance Limits. In *Scalable High-Performance Computing Conference*, pages 229–238, Knoxville, May 1994. IEEE, IEE Computer Society Press.

[6] J.-L. Dekeyser, C. Fonlupt, and P. Marquet. Dynamic Load Balancing on SIMD Data-Parallel Computers. Technical Report AS-149, LIFL, Univ. des Sciences et Technoloigies de Lille, April 1994.

[7] G.A. Geist. PVM 3 Beyond network Computing. In Jens Volkert, editor, *Parallel Computation*, number 734 in LNCS, pages 194–203. Springer-Verlag, October 1993.

[8] P.J. Hatcher and M.J. Quinn. *Data-Parallel Programming on MIMD Computers*. Scientific and Engineering Computation. The MIT Press, 1991.

[9] Hyperparallel Technologies. *Hyper C*, 1993.

[10] M. Kaddoura and S. Ranka adn A. Wang. Array Decompositinos for Clusters of Machines with Non-Uniform Computational Power. Technical report, School of Computer Science, Syracuse University, June 1994.

[11] J. De Keyser and D. Roose. Load balancing data parallel programs on distributed memory computers. *Parallel Computing*, 19:1190–1219, 1993.

[12] P. Krueger and R. Chawla. The Stealth distributed scheduler. In *The 11th Int. Conf on Distributed Computing Systems*, pages 336–343, Alington, Texas, USA, May 1991. IEEE, IEEE computer society press.

[13] J.-L. Levaire. *Contribution à l'étude sémantique des langages à parallélisme de données ; application à la compilation.* PhD thesis, Université de Paris 7, LIP, ENS Lyon, February 1993.

[14] A. Matrone, P. Schiano, and V. Puoti. LINDA and PVM: A comparison between two environments for parallel programming. *Parallel Computing*, 19:949–957, 1993.

[15] Nenad Nedeljković and Michael J. Quinn. Data-parallel programming on a network of heterogeneous workstations. *Concurrency: Practice and Experience*, 5(4):257–268, June 1993.

[16] D.M. Nicol. Rectilinear Partitioning of Irregular Data Parallel Computations. *J. of Parallel and Distributing Computing*, to appear 1994.

[17] D.M. Nicol and P.F. Reynold. Optimal Dynamic Remapping of Data Parallel Computations. *IEEE Tran. on Computers*, 39(2):206–219, February 1990.

[18] N. Paris. Compilation du flot de contrôle pour le parallélisme de données. *Technique et science informatiques*, 12(6):745–773, 1993.

[19] Scient. Comp. Ass., New Haven. *C-LINDA reference manual*, 1991.

Annexe

Influence des communications sur l'efficacité

Il existe déjà une longue littérature sur l'opportunité d'utiliser les réseaux de station de travail dans le cadre du calcul intensif. Il est clair que dans l'état actuel des technologies le principal frein est le faible débit des réseaux par rapport aux puissances des stations de travail. La technologie des processeurs croissant nettement plus rapidement que celle des réseaux, cet écart devrait se creuser de plus en plus.

La virtualisation des programmes data-parallèles conduit à des programmes SPMD composés de successions de phases calcul/communication. On peut reprendre l'analyse de Hatcher et Quinn pour évaluer l'efficacité de la virtualisation d'un programme data-parallèle en fonction de sa granularité [8]. Soit n la taille du domaine, p le nombre de stations, $f(n)$ le volume de calculs élémentaires dans une phase de calcul, i.e. la *granularité du programme*, $g(n,p)$ le volume de données dans la phase de communication, μ le temps d'un calcul élémentaire, τ le temps de communication d'une donnée. On se place dans le cas où les communications sont sérialisées et qu'il n'y a pas de recouvrement calcul/communication. L'efficacité

est donnée par la formule suivante :

$$e = \frac{\mu f(n)}{\mu f(n) + \tau g(n,p)} \quad (1)$$

Le modèle de communication est simplifié et ne considère pas un temps d'initialisation. On l'admet car on s'interesse au cas où le volume de communication est assez grand pour absorber le coût d'initialisation.

Typiquement le volume de calcul de chaque phase est de l'ordre de la taille du domaine, i.e. il y a une séquence de k opérations élément à élément avant chaque opération de réarrangement : $f(n) = k \times n$. L'efficacité (1) est alors donné par :

$$e = \frac{1}{1 + \frac{g(n,p)}{n} \times \frac{1}{k} \times \frac{\tau}{\mu}} \quad (2)$$

La fraction $\frac{\tau}{\mu}$ représente le rapport entre la vitesse des stations et le débit du réseau, i.e. la *granularité du réseau*.

La régularité est un critère de classification des opérations de communications.

Communications irrégulières

Dans le cas de communications irrégulières, on fait l'hypothèse d'une communication aléatoire et uniforme sur le domaine. Chaque processeur virtuel va communiquer avec un autre processeur virtuel choisi aléatoirement selon une loi uniforme. Le domaine est partitionné en p blocs de taille $n_i \geq 0$ ($\sum_{i=1}^{i=p} n_i = n$). Pour un processeur virtuel d'une station i, la probabilité que la communication soit interne (i.e. ne sorte pas de la station) est égale n_i/n. Pour une station i, l'espérence mathématique des communications internes est égale à n_i^2/n. Pour le réseau en entier, l'espérence mathématique des communications internes est égale à $\sum_{i=1}^{i=p} n_i^2/n$. Donc l'espérence mathématique du volume de communications sur le réseau (les communications externes) est égale à :

$$n - \sum_{i=1}^{i=p} n_i^2/n = n - n\sum_{i=1}^{i=p} (n_i/n)^2 \leq n - n/p = n((p-1)/p)$$

En posant $g(n,p) = n((p-1)/p)$ dans l'équation (2), l'efficacité dans le cas de communications irrégulières est :

$$e_i = \frac{1}{1 + \frac{p-1}{p} \times \frac{1}{k} \times \frac{\tau}{\mu}} \quad (3)$$

C'est une efficacité dans le pire des cas et on remarque qu'elle ne dépend pas de la taille du domaine, le volume de communications étant du même ordre que celui des calculs. L'expression $u = \frac{1}{k} \times \frac{\tau}{\mu}$ représente le rapport entre la granularité du réseau et la granularité du programme original. La courbe d'efficacité en fonction de ce rapport et du nombre de stations est représentés sur la figure 7. La granularité des programmes data-parallèle étant généralement plus faible que celle du réseau, on atteint très vite une mauvaise efficacité.

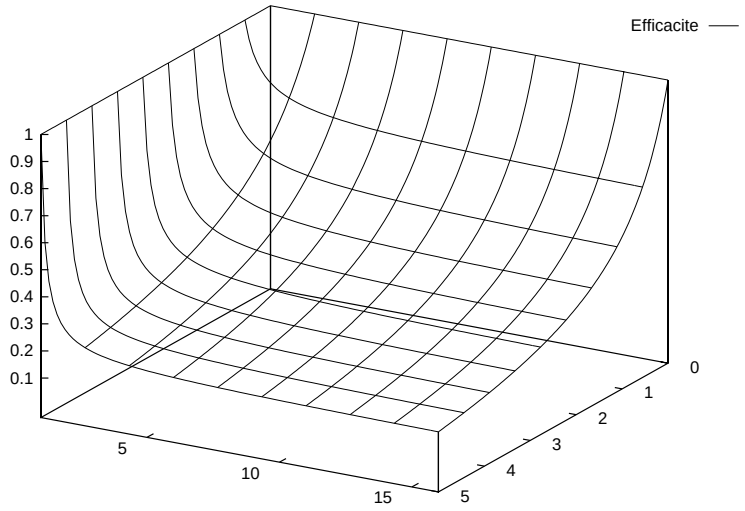


FIG. 7 - *Efficacité dans le cas de communications irrégulières, en fonction du ratio granularité réseau/granularité programme et du nombre de stations.*

Communications régulières

Le représentant typique de cette classe de communications est le décalage sur une grille. Si on ce place dans le cas d'une grille 2D avec un décalage dans une seule dimension, le nombre de données transitant sur le réseau est proportionnel au nombre de stations comme le résume la figure 8. On a $g(n,p) = p\sqrt{n}$. L'efficacité initiale (2) devient :

$$e_r \quad = \quad \frac{1}{1 + \frac{p}{\sqrt{n}} \times \frac{1}{k} \times \frac{\tau}{\mu}} \tag{4}$$

Ici l'efficacité dépend de la taille du domaine. Les courbes d'efficacités pour un nombre de stations fixé en fonction du rapport granularité réseau/granularité programme et de la taille du domaine est représenté sur la figure 4. L'efficacité est nettement supérieure à celle des communications irrégulières pour un nombre raisonnable de stations.

Reduction du coût des communications

Les performances peuvent brutalement chuter par une prééminence des coûts de communications. Ce coût peut provenir d'un encombrement du réseau et/ou d'une augmentation du volume de communication.

L'etude de l'efficacité précédente indique que l'on peut reduire naturellement le coût des communications en réduisant le nombre de stations. Nous avons déjà implémenté cette idée

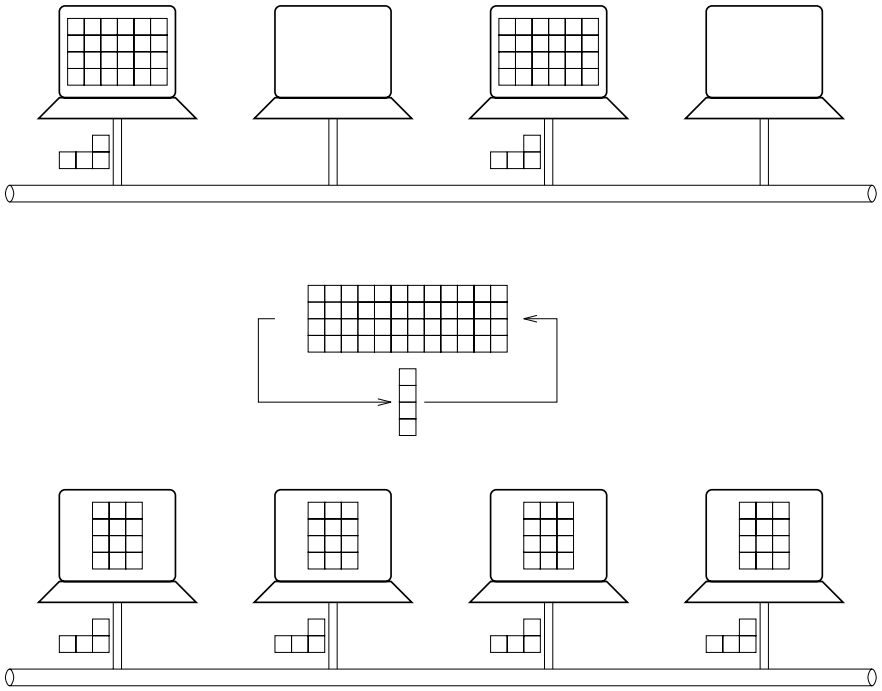


FIG. 8 - *Proportionalité du volume de communications par rapport au nombre de stations. Le domaine est un tore 2D et l'opération de communication est un décalage du domaine dans une dimension. Quand le domaine est distribué sur deux stations, deux colonnes transitent sur le bus, tandis qu'avec un découpage sur quatre stations, ce sont quatre colonnes qui transitent.*

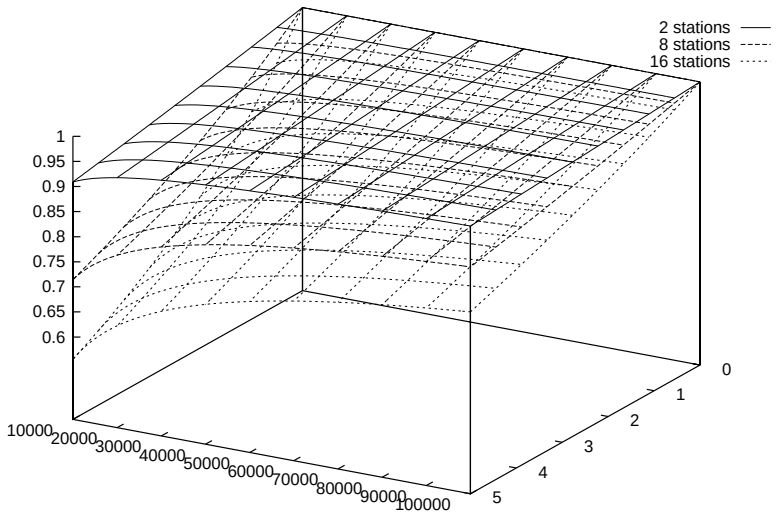


FIG. 9 - *Efficacités dans le cas de communications régulieres en fonction du ratio granularité réseau/granularité programme et de la taille du domaine.*

dans le compilateur C*, mais le surcoût de gestion des communications de la librairie C* est tel que les bénéfices de la réduction du volume des communication sont toujours absorbés. D'autre part, dans un environement UNIX, il est très difficile d'instrumenter précisement le coût de communications.

L'opération de réduction du nombre de stations s'avère coûteuse de par le volume de données à déplacer. Il est nécessaire d'introduire un mécanisme complexe qui détermine l'oportunité d'une telle opération. Sans une connaissance a priori de l'évolution de l'exécution de l'application, seule une approche probabiliste peut être envisagée. Nicol et Reynolds ont proposé une méthode intéressante basée sur les chaînes de Markov dont on pourrait s'inspirer [17].

Dans le cas des communications irrégulières on remarque que par rapport à l'entropie de la distribution, le volume de communication est d'autant plus grand que l'équilibrage spatial est parfait. Par exemple avec deux stations, si les n processeurs virtuels sont partagés équitablement on a une espérence mathématique de $n \times 1/2$ pour les communications externes. Par contre, avec une distribution en $9/10$ et $1/10$ elle n'est plus que de $n \times 9/50 \leq n \times 1/5$. En fait la courbe d'efficacité de la figure 7 est justement celle d'une distribution spatiale homogène. Une autre idée serait de choisir les stations qui vont maximiser l'entropie lors de la réduction de leur nombre, i.e. profiter du déséquilibre spatial induit par le déséquilibre de charge. On peut de même renforcer le déséquilibre pour introduire un déphasage entre les stations (i.e. certaines émettront leurs messages avant les autres) qui permettrait de réduire les contentions du réseau. C'est ce que nous nous proposons d'implémenter ultérieurement dans une version améliorée de la librarie C*.