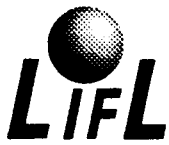
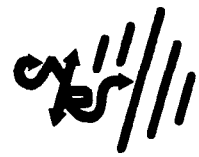
NUMÉRO D'ORDRE : 1453 ANNÉE : 1994



Laboratoire d'Informatique Fondamentale de Lille



THÈSE

présentée à

L'UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE

pour obtenir le titre de

DOCTEUR en INFORMATIQUE

par

Cyrille FONLUPT

Distribution dynamique de données sur machine SIMD

Thèse soutenue le 16 décembre 1994, devant la commission d'examen :

Président :	Vincent CORDONNIER	LIFL, Université de Lille 1
Directeur de thèse :	Jean-Luc DEKEYSER	LIFL, Université de Lille 1
Rapporteurs :	Isaac SCHERSON	University of California, Irvine, USA
	Denis TRYSTRAM	LMC-IMAG, Grenoble
Examinateurs :	Philippe MARQUET	LIFL, Université de Lille 1
	Serge MIGUET	LIP, École normale supérieure de Lyon

UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE
U.F.R. d'I.E.E.A. Bât. M3 - 59655 VILLENEUVE D'ASCQ CEDEX
Tél. (33) 20 43 47 24 Télécopie (33) 20 43 65 66

Table des matières

Introduction

À chaque accroissement des performances des ordinateurs, les communautés scientifiques diverses ont la possibilité de résoudre plus rapidement et plus précisément leurs problèmes, leur ouvrant ainsi de nouvelles perspectives qui réclament encore plus de capacités de calcul. Tel Sisyphe, les informaticiens ont le devoir de pousser plus avant leurs travaux pour atteindre un maximum de performances sans cesse plus éloigné.

Ainsi les architectures séquentiels permettant de traiter plusieurs millions d'opérations par secondes, deviennent trop « lentes » pour certaines applications scientifiques. Les limites technologiques d'intégration sont pratiquement atteintes et ne permettent pas d'espérer un accroissement aussi rapide que celui connu durant ces dernières années.

La méthode la plus prometteuse pour dépasser ces critères physiques est d'utiliser le parallélisme. Le parallélisme correspond à l'utilisation simultanée de plusieurs processeurs sur le même programme. Le but est de multiplier la puissance de calcul du système par le nombre de processeurs. Si un processeur de la CM-5 a une puissance de 128 MégaFlops, l'utilisation conjointe de 1024 processeurs offrira une puissance *théorique* de 131.000 MégaFlops. Cependant, la réalité est toute autre puisque la meilleure performance mesurée est de 59.700 MégaFlops [GMS94]. Cette différence importante entre la puissance théorique et la puissance mesurée qui augmente avec le nombre de processeurs s'explique en partie par la difficulté de programmation d'une machine parallèle.

Un programme parallèle comporte un ensemble de composants, chacun devant être placé sur un nœud en vue de son exécution ou de son utilisation. Si le placement des composants d'une application parallèle n'est pas optimisé, certains nœuds du système se trouveront surchargés tandis que d'autres se trouveront inoccupés. L'ordonnancement de ces composants est donc un problème central du parallélisme et un des points essentiels pour l'utilisation efficace d'une machine. Cet ordonnancement peut être réalisé de manière statique pendant la phase de compilation ou dynamiquement durant l'exécution du programme. Cette thèse ne s'intéresse qu'à cette dernière classe, on parle dans ce cas d'équilibrage ou rééquilibrage dynamique.

Le problème de l'équilibrage de charge dynamique est historiquement lié au développement des machines MIMD. En effet, même si les systèmes multi-processeurs ont montré leur puissance dans la résolution de problèmes qui pouvaient être partitionnés en sous-tâches de taille plus réduite, il existe un grand nombre de problèmes pour lesquels les ressources nécessaires ne sont connues qu'au cours de l'exécution. Des techniques d'équilibrage dynamiques sont alors nécessaires afin d'éviter que certains processeurs possèdent tout le travail et que d'autres attendent inoccupés.

Dans les architectures parallèles à gros grains, l'entité à placer est appelée processus ou tâche, chaque processus contient son propre code et ses données, il peut s'exécuter indépendamment des autres processus.

Dans les ordinateurs parallèles à grains fins, l'entité à placer est appelée donnée ou point. L'ensemble des Processeurs Élémentaires exécute les instructions émises par un séquenceur et travaille sur leurs données locales. Une question se pose, l'approche SIMD peut elle bénéficier de techniques d'équilibrage dynamiques similaires aux machines asynchrones?

Nous qualifions d'*algorithme à piles* tout algorithme où le traitement sur une donnée peut être exécuté indépendamment des autres et sur n'importe quel processeur élémentaire. La pile est constituée d'objets ; chaque objet est traité suivant le même algorithme. Un élément d'une pile distribuée peut éventuellement engendrer des successeurs. Le modèle data-parallèle est bien adapté à la résolution des algorithmes à piles, chaque processeur élémentaire devenant responsable d'une pile distribuée et exécutant la même instruction ou séquence d'instructions sur tous les objets.

Les méthodes de simulation d'un flux de particules dans un détecteur sont une illustration parfaite de ce genre d'algorithmes [Dek86]. Chaque particule est suivie pas à pas dans le détecteur jusqu'à son arrêt, lors de chaque pas elle peut produire des particules secondaires. Celles-ci sont empilées, leur traitement proprement dit est retardé. L'algorithme « tracking » peut être facilement parallélisé par distribution de la pile sur l'ensemble des processeurs élémentaires. Cette parallélisation entraîne malheureusement un déséquilibre de la charge sur chacun des processeurs élémentaires.

De nombreux algorithmes distribués de parcours d'arbres ont été implémentés avec succès sur des machines de type MIMD [KR87, RK87]. Powley et *al* [PFK91, PFK93] ont introduit la possibilité d'effectuer de l'équilibrage dynamique pour un algorithme à pile ciblé pour le parcours d'arbre en parallèle. Il suffit d'avoir un nœud racine et une fonction permettant de générer les successeurs. Les successeurs sont générés en parallèle et empilés de manière distribuée. Le traitement d'un sous-arbre peut conduire à une impasse, tandis que l'exploration d'autres nœuds amènera à la création de nouveaux sous-arbres.

Le but essentiel d'une technique d'équilibrage dynamique sur un algorithme à pile est

d'assurer une distribution équilibrée des éléments de la pile sur l'ensemble des processeurs élémentaires afin de conserver un taux de parallélisme maximal.

Le but de cette thèse est de proposer de nouvelles techniques d'équilibrage de charge pour les algorithmes à pile et vérifier leur adéquation au modèle d'exécution SIMD. En tenant compte des caractéristiques des algorithmes à pile, nous nous sommes posés les questions suivantes :

- comment caractériser une technique d'équilibrage dynamique pour un algorithme à pile?

  Des éléments de réponse à cette question sont donnés dans le chapitre I. L'existant des méthodes d'équilibrage sur machine MIMD est étudié ainsi que ses éventuelles implications sur le modèle SIMD ;

- comment analyser une technique d'équilibrage?

  En étudiant diverses méthodes mathématiques d'analyse d'algorithmes nous avons examiné diverses techniques et leur transposition dans le modèle SIMD.

Cette comparaison nous a amené à définir un modèle de réarrangement dynamique des données dont les deux principales caractéristiques sont la politique de déclenchement et la technique de redistribution des données. Le chapitre III présente ce modèle SIMD. Les algorithmes d'équilibrage pour les algorithmes à pile sont présentés par l'utilisation d'un langage spécifique au data-parallélisme. Les diverses politiques de déclenchement sont également présentées.

À partir de la définition des techniques de redistribution dynamiques, les qualités et désavantages des algorithmes ont été étudiés de deux manières :

- la modélisation des algorithmes permet d'estimer certains paramètres importants des techniques d'équilibrage comme la convergence effective. Cette analyse est présentée dans le chapitre IV ;

- l'analyse des performances des algorithmes est étudiée par des simulations réalisées sur la MasPar MP-1 dans le chapitre V.

Chapitre I

Équilibrage de charge MIMD au service du SIMD

Les travaux sur l'allocation des processus sur les systèmes répartis asynchrones ont été très actifs ces dernières années. En particulier, le problème de l'équilibrage dynamique de charge a été l'objet d'une attention soutenue de la part de la communauté informatique. Par contre, le modèle d'équilibrage synchrone a été fort tardivement étudié, car il était d'usage de le considérer comme inutile voire impossible. De timides expériences suggèrent cependant que l'implémentation de techniques de rééquilibrage dynamique pourrait également être profitable aux machines SIMD.

Nous nous proposons dans ce chapitre d'examiner les différentes techniques de rééquilibrage dynamique existantes suivant une classification simple comportant cinq caractéristiques principales. Pour chaque classe, nous étudierons les concepts qui peuvent être appliqués au monde SIMD et ceux dont l'adaptation est par essence impossible. Nous présentons également pour chaque classe des exemples typiques aux machines asynchrones et synchrones. Afin de bien différencier les méthodes MIMD et SIMD, nous débuterons toutes les sections par une description de l'existant en MIMD, le sigle de la figure I.1 introduira la section consacrée au SIMD.

FIG. I.1 - *Figure introduisant le SIMD*

Toutefois, avant de rentrer plus en détails dans l'examen de différentes stratégies, il est important de définir correctement le terme de rééquilibrage ou équilibrage dynamique. Les termes de « load sharing » et de « load balancing » sont parfois confondus alors qu'il existe

une distinction. Le but de toute technique d'équilibrage de charges (« load balancing ») est de maintenir une charge équivalente sur l'ensemble des processeurs tandis que le partage de tâches (« load sharing ») consiste à garder tous les nœuds occupés pendant l'exécution. Intuitivement, il semble logique qu'équilibrer le nombre de tâches parmi les nœuds permet une meilleure stabilité du système. Cependant comme nous le verrons par la suite, quelques soient les techniques employées, l'équilibrage dynamique impose en permanence un sur-coût de travail qui doit être estimé en regard des améliorations qu'il peut apporter.

1 Classification des algorithmes d'équilibrage MIMD

Une taxinomie proposée par Casavant et *al* [CK88] permet de décrire de façon hiérarchique les algorithmes d'ordonnancement de tâches. Les algorithmes d'équilibrage de charges peuvent être considérés comme un cas particulier des techniques d'ordonnancement. Afin de pouvoir prendre en considération les derniers développements des méthodes de redistribution de charges, nous avons adapté les diverses classifications existantes [CK88, WLR89, SKS92, BGT94]. Nous reproduisons sur la figure I.2 notre classification des algorithmes MIMD.

Notre classification repose sur cinq caractéristiques principales ;

Politique de mise à jour des informations La politique d'informations tente d'obtenir un état ou une carte du système en faisant transiter ou en collectant des informations sur l'ensemble ou une partie des nœuds.

Politique de déclenchement La politique de déclenchement détermine à quel moment la redistribution des tâches doit être effectuée à partir des informations fournies par la politique précédente.

Politique de sélection La politique de sélection détermine les différents nœuds déséquilibrés qui joueront le rôle d'émetteur ou de receveur.

Politique de désignation locale La politique de désignation locale identifie les processus qui vont être transférés. Elle est réalisée immédiatement après la politique de sélection lorsque le nœud est désigné comme émetteur. Par contre, elle s'exécute après la politique d'appariement si le nœud est receveur.

Politique d'appariement Il s'agit de trouver un ou plusieurs partenaires au processeur qui a été choisi par la politique de sélection. Cette partie est à juste raison considérée comme étant le cœur d'une méthode d'équilibrage. Le partenaire peut être recherché sur un domaine limité spatialement à un voisinage de processeurs ou sur l'ensemble du système.

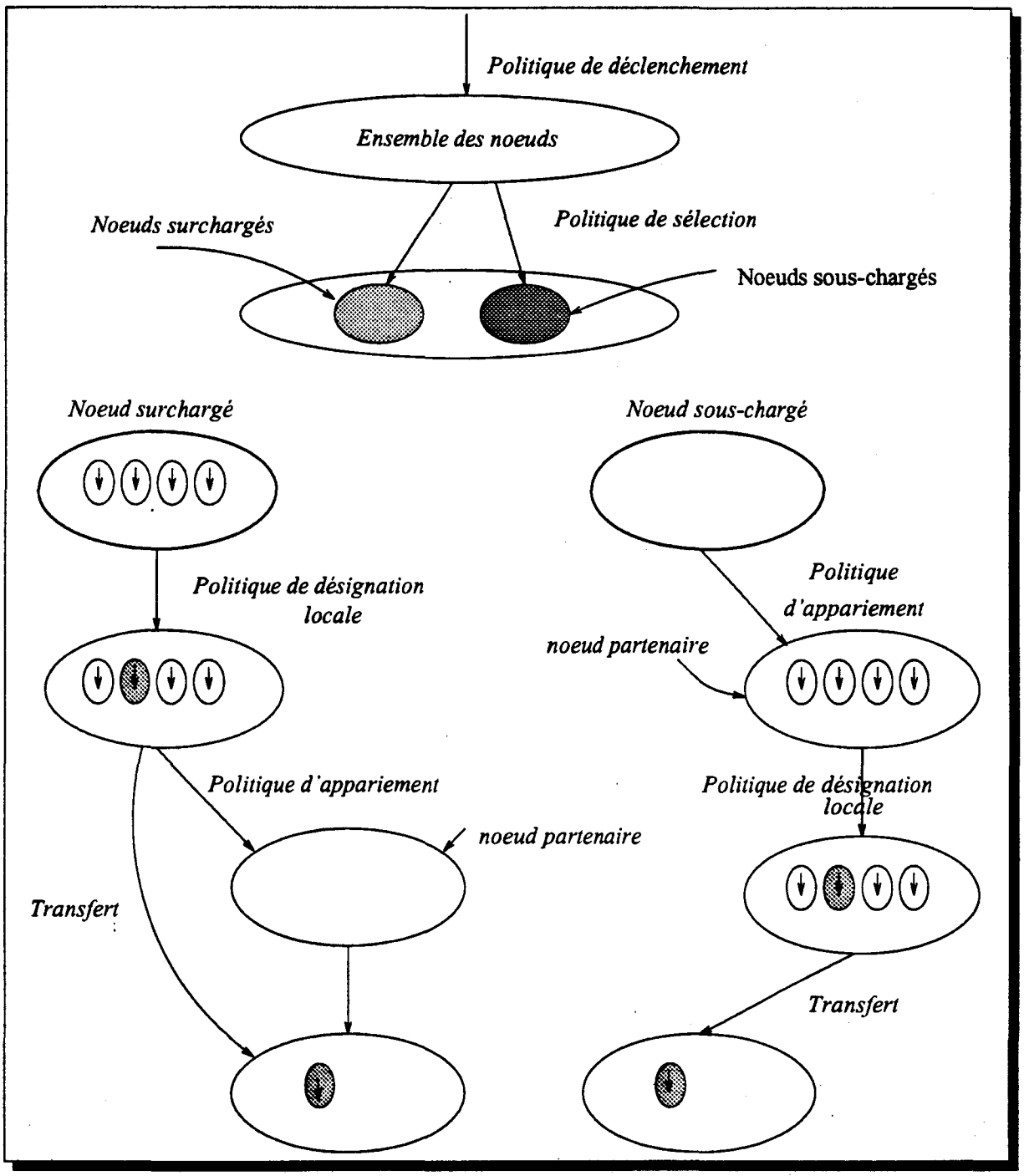


FIG. I.2 - *Classification des algorithmes d'équilibrage MIMD*

2 Politique de mise à jour des informations

Le but de la politique d'informations est de rassembler des informations sur les états des nœuds de l'ensemble multi-processeurs et d'estimer un état global ou partiel du système.

La mise en œuvre de la politique de mise à jour des informations est particulièrement délicate dans les machines MIMD car elle doit répondre à quatre questions différentes :

1. Quelles informations sont nécessaires à l'élaboration d'une technique d'équilibrage ou de partage de tâches?
2. À quel moment ces informations doivent elles être demandées/envoyées?
3. Qui est le récepteur/demandeur de ces informations? Existe-t-il un site centralisateur ou la demande est-elle répartie?
4. Quels seront les processeurs qui répondront à cette requête?

Nous étudierons dans le paragraphe 2.1 la définition de l'indicateur de charge qui répond à la première question. Les différentes solutions existantes répondant au deuxième critère seront notre fil conducteur dans cette section. Pour chacune des solutions envisagées, nous examinerons quelles sont les entités réceptrices et quels sont les processeurs qui répondent à la requête d'informations.

2.1 Mesure de la charge

L'état de charge des différents processeurs est la principale source d'informations pour les techniques d'équilibrage. Ce qui nous amène à expliciter la notion de charge d'un processeur. Plusieurs problèmes apparaissent :

1. Comment apprécier la charge d'un processeur?
2. Quels sont les critères à retenir dans la définition d'un indicateur de charge?
3. Comment éviter des fluctuations subites et obtenir une valeur moyenne?

Le but premier de la mesure de la charge est d'estimer le résidu d'exécution des processus sur chacun des nœuds. Ces valeurs sont ensuite communiquées aux différents nœuds

par l'intermédiaire de la politique de mise à jour des informations. Les trois points suivants répondent aux trois questions soulevées par la définition d'un indicateur de charge.

1. De nombreux critères de mesure de la charge ont été proposés sans qu'aucune solution universelle n'ait été trouvée. Dans une série d'expériences conduites sur un ensemble de stations de travail, Kunz [Kun91] constate après utilisation des indicateurs de charge les plus classiques, un temps de réponse du système variant de plus de 30% pour un même problème.

2. Le critère le plus utilisé pour un indicateur de charge est le nombre de processus en attente d'exécution. Ce critère présente l'avantage d'être calculé très rapidement et de refléter avec justesse la charge instantanée d'un processeur. Il permet de prédire grossièrement l'évolution de la charge dans un futur proche.

3. La mesure du nombre de processus en attente d'exécution est parfois sujette à de rapides fluctuations si, au moment de l'échantillonage, plusieurs processeurs se trouvent dans une phase d'attente, par exemple d'opérations d'entrées/sorties. Une solution pour éviter cette instabilité chronique est de moyenner la charge pendant une certaine période. La mise à jour des informations ne peut cependant plus se faire suivant une fréquence élevée, ce qui entraîne une baisse de la qualité de l'équilibrage.

   Une autre possibilité pour obtenir une valeur relativement stable est de prendre en compte une combinaison linéaire [FZ86, Kun91] de plusieurs critères comme la longueur de la file d'attente, la taille de la mémoire vive disponible, le taux d'utilisation du CPU, le nombre d'appels au système d'exploitation... Une fonction à plusieurs variables $f(d_1, d_2, \ldots d_n) = a_1 \times d_1 + a_2 \times d_2 + \ldots + a_n \times d_n$ peut être définie. Les d_i étant les différents indicateurs de charge et les a_i déterminant l'impact de chaque descripteur.

Mesure de la charge

Dans le cas de machine SIMD, le descripteur retenu pour estimer la charge est l'énumération du nombre de données en attente d'exécution sur chaque processeur élémentaire. Cet indicateur de charge correspond directement au mécanisme le plus utilisé dans le modèle MIMD, le calcul du nombre de processus en attente d'exécution sur chaque nœud. Cet indicateur de charge est aisé à implémenter sur une machine SIMD. Sur une machine synchrone, il reflète *instantanément* la charge des processeurs élémentaires. Les problèmes dus aux délais de transmission de ces informations sur certaines machines MIMD sont inexistants.

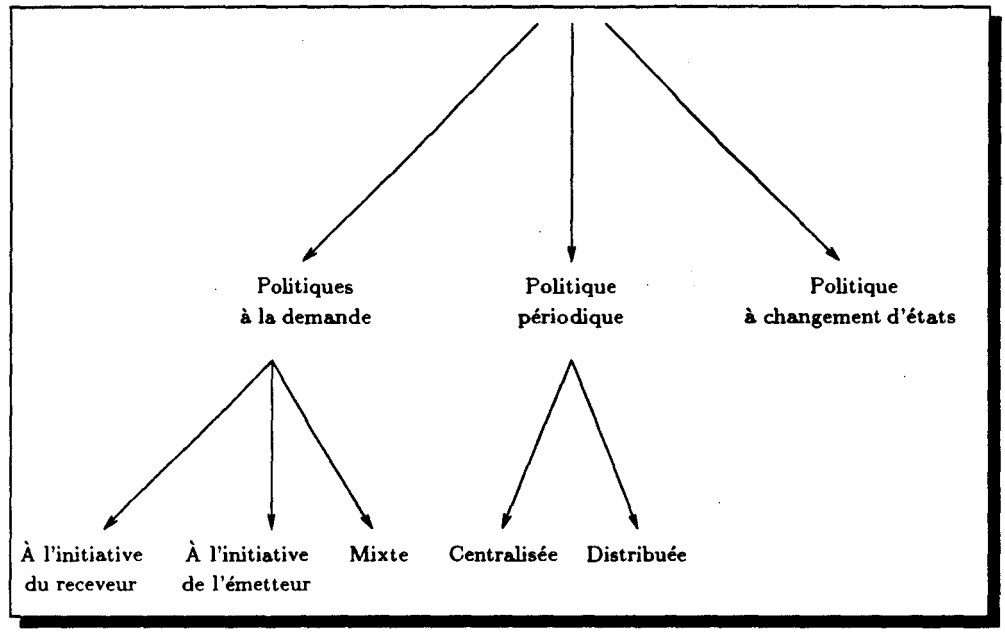


FIG. I.3 - *Classification des différentes politiques d'informations*

2.2 Déclenchement de la transition d'informations

Le second critère caractérisant une politique d'informations décide du déclenchement de la transition des informations. Trois classes peuvent être distinguées :

Politiques à la demande Dans le cadre des politiques à la demande, un nœud décide de réunir des informations sur le système, s'il a préalablement été désigné par la politique de sélection comme un émetteur ou receveur potentiels.

Politiques périodiques Cette politique interroge de manière régulière l'ensemble des processeurs afin de connaître leur niveau de charge. Les résultats obtenus peuvent être soit centralisés, soit distribués sur un ensemble de processeurs.

Politique à changement d'états Au lieu de demander ou de rassembler des informations, une politique à changement d'états dissémine les informations à travers le système. Lorsqu'un nœud estime qu'il passe dans un état remarquable (inactif ou surchargé par exemple), il diffuse cette information à travers le système.

Les caractéristiques de ces trois politiques d'informations présentées en figure I.3 sont précisées dans les sections suivantes :

2.2.1 Les politiques d'information à la demande

Une politique d'information à la demande est caractérisée par une requête d'informations provenant d'un ou plusieurs processeurs. Deux cas peuvent se produire pour désigner le demandeur des informations :

- le demandeur est un nœud surchargé désirant céder une partie de sa charge. On parle alors de stratégie à l'initiative de l'émetteur ou de l'envoyeur ou encore de stratégie active. Les algorithmes *SID* (Sender Initiative Decision) [WLR93, KGV94, DTJ89, ELZ86a, CLZ92, KS92] sont des représentants typiques des politiques d'informations déclenchées par un processeur fournisseur de travail ;
- le demandeur est un nœud inactif ou sous-chargé désirant obtenir du travail. On parle alors de stratégie à l'initiative du receveur ou encore de stratégie passive. Une politique passive représente le pendant de la politique précédente. Dans les algorithmes *RID* (Receiver Initiative Decision) [WLR93, KGV94, DTJ89], les processeurs se trouvant à l'état oisif ou sous-chargé génèrent une demande d'informations.

Avantages et inconvénients des stratégies actives/passives

Les stratégies actives et passives ont un comportement étroitement dépendant de l'état de charge du système :

Système possédant une charge moyenne élevée Les stratégies à l'initiative du demandeur ont un mauvais comportement et provoquent l'instabilité. Les nœuds surchargés ne peuvent pas trouver de partenaires mais continuent à émettre des demandes d'informations. Il arrive un moment où le coût de l'équilibrage devient plus important que son gain. Les stratégies passives offrent par contre des performances correctes. Seuls les nœuds sous-chargés subiront l'overhead de l'algorithme d'équilibrage.

Système possédant une charge moyenne faible Les stratégies actives ont un meilleur comportement que les stratégies passives. Les stratégies passives effectuent un nombre important de demandes d'informations dans un environnement possédant peu de nœuds surchargés.

Quelques méthodes jouent sur les deux facettes et intègrent à la fois une stratégie passive et active. Elles sont appelées stratégies *mixtes*. Elles sont également appelées stratégies symétriques, car elles essaient de concilier les avantages d'une politique dont l'initiative est laissée au receveur avec les avantages des algorithmes où la demande est effectuée par le receveur. Ce particularisme permet une adaptation plus facile du système à son état général.

Dans un ensemble multi-processeurs, où la charge générale est faible, la stratégie mixte sera une technique dont l'initiative est laissée à l'envoyeur. *A contrario*, pour une charge moyenne importante, un algorithme dont l'initiative est laissée au receveur se mettra en place et remplacera la technique précédente. Cependant, Shivaratri [SKS92] remarque qu'une politique à double facette, présente l'inconvénient majeur de cumuler les désavantages des stratégies passives et actives ; une politique dont l'initiative est à l'envoyeur entraîne l'instabilité pour les systèmes à forte charge.

L'algorithme *Actpas* [KW91] autorise par exemple les processeurs oisifs ayant effectués au préalable une requête de travail non satisfaites à se mettre dans un état passif afin d'attendre une émission de travail des nœuds surchargés. De façon semblable, les algorithmes symétriques proposés par Shivaratri et Krueger [SK90a] et Joosen et *al* [JBV93] autorisent la cohabitation de politiques active et passive. Chaque nœud déclenche la stratégie qui correspond à son état.

Dukers et *al* [DD92] montrent empiriquement que les stratégies mixtes présentent de bonnes performances, sans toutefois égaler celles des stratégies passives. Cette réponse doit cependant être quelque peu nuancée par les expérimentations effectuées par Shivaratri et *al* [SKS92]. Avec certaines modifications afin de prévenir l'instabilité, le comportement d'un algorithme symétrique est meilleur que celui basé sur une stratégie passive. Pour éviter l'instabilité, l'algorithme *Symétrique* [SK90a] utilise les informations rassemblées durant les précédentes phases d'équilibrage afin de classifier les nœuds du système comme émetteur, receveur ou neutre. Ces informations sont stockées dans une structure de données maintenue indépendamment par chaque nœud. En utilisant une telle structure, le choix du ou des nœuds à interroger est facilité et permet d'éviter les phénomènes d'instabilité lorsque la charge du système devient élevée.

2.2.2 Les politiques d'information périodiques

Une politique périodique d'informations examine le système suivant une fréquence fixe déterminée au préalable qui peut être modifiée dynamiquement en fonction de l'évolution du système. L'ensemble des processeurs répondent à la requête de demande d'informations. Le récepteur de l'information peut être soit un site unique, soit un ensemble de nœuds du système.

Approche centralisée

L'ensemble des processeurs envoient périodiquement leur charge vers un nœud dédié, souvent appelé *serveur* car il est susceptible de fournir ultérieurement cette information. Dans les

algorithmes *Global* et *Central* [Zho88], une des machines hôtes reçoit régulièrement la mise à jour de la charge de l'ensemble des machines. Ces informations sont alors intégrées à un *vecteur de charges* qui est diffusé à tous les autres sites. Même si cette technique offre de très bonnes performances, sur des systèmes de taille modeste, elle ne peut bien évidemment pas convenir pour les ensembles comportant plusieurs centaines voire plusieurs milliers de processeurs. Zhou et *al* [ZWZD93] proposent une solution pour pallier à cet inconvénient. Il s'agit alors de partager les nœuds par grappes et d'utiliser un algorithme *Central* ou *Global* intra-grappes . Les simulations effectuées sur un ensemble de 1000 stations de travail montre que le sur-coût induit par cette politique d'informations reste largement inférieur à 1%.

Approche distribuée

L'ensemble des nœuds diffuse périodiquement leur charge sur l'ensemble ou une partie du système. Les nœuds des algorithmes *PRandom*, *Pmin* [AGM93] envoient toutes les T_u secondes leur propre charge à tous leurs voisins. Dans l'algorithme *Disted* [Zho88], chaque site diffuse périodiquement sa charge à l'ensemble des autres nœuds.

L'implémentation réalisée par Stankovic [Sta84] est plus subtile. Chaque hôte transmet périodiquement une estimation de la charge de chaque site du réseau, et envoie ces informations à ses plus proches voisins. Les informations sont mises à jour sur chaque site de la façon suivante. Pour chaque site i, le système peut être partitionné en trois classes, le site lui-même, ses voisins et les autres. L'information du site sur lui-même sera considérée comme exacte. L'information fournie pas les voisins sur eux-mêmes sera considérée comme la meilleure information possible. La charge de tous les autres sites sera la moyenne obtenue par les informations reçues. Cette approche permet de limiter le nombre de messages à l'intérieur du système tout en permettant à chaque nœud de posséder la charge de tous les autres sites.

Un échange d'informations réalisé périodiquement présente l'avantage de créer un faible sur-coût. Cependant lorsque la charge du système devient important une politique périodique provoque un sur-coût non négligeable puisque les nœuds sont déjà occupés par l'exécution de processus.

2.2.3 Les stratégies obéissant à un changement d'états

Une politique d'information obéissant à un changement d'états est caractérisée par une diffusion de l'information à travers le système. Si un nœud détecte que son état a été modifié de manière sensible (passage à l'état inactif par exemple), il émet son nouvel état à travers tout le système. Cette stratégie diffère d'une politique à la demande dans le sens où, au lieu d'essayer de récupérer des informations sur d'autres nœuds, elle tente plutôt de disséminer

des informations sur son état. Les sites récepteurs des informations sont répartis.

Les nœuds de l'algorithme *Drafting* [NXG85] peuvent se trouver dans 3 états différents, soit à l'état H-Load lorsqu'il est surchargé, L-Load lorsque la charge est faible ou bien N-Load pour indiquer un état normal. Un vecteur de charge est créé sur chaque nœud. Si l'état local d'un processeur est modifié de manière très sensible, son nouvel état est diffusé à travers tout le système permettant aux autres nœuds de modifier leur vecteur de charge.

Les changement d'états de l'algorithme *STB* (State Broadcast Algorithm) [LM82, DTJ89] sont définis par l'arrivée ou la terminaison d'une tâche. L'algorithme *STB* diffuse un message indiquant son nouvel état. Ceci permet aux autres processeurs de mettre à jour leur vecteur de charges.

Ces algorithmes nécessitent cependant un réseau de communications performant. L'algorithme de Leland et *al* [LO86] basé sur une diffusion de la charge est plus particulièrement destiné à des topologies de type anneau où les opérations de diffusion sont fort peu coûteuses.

Déclenchement de la transition d'informations

Sur une machine SIMD, la politique d'informations ne peut être réalisée qu'à la demande du séquenceur. Aucune latitude n'est laissée aux PEs pour effectuer de telles démarches si leur état change de manière brutale. Un modèle centralisé est donc plus à même de répondre au besoin d'une architecture SIMD. Dans ce cas, les politiques à la demande et par changement d'états ne sont pas adaptables sur une machine synchrone.

Par contre, la collecte d'informations sur une architecture SIMD est aisée, car elle peut s'appuyer sur le particularisme de ces machines :

1. Si la charge mesurée est le nombre d'objets possédé par chaque processeur élémentaire, la charge moyenne du système peut être calculée et communiquée aux autres PEs par les opérations de parallel-prefix.

2. Lorsque la charge instantanée est mesurée, la valeur obtenue est exacte. Les problèmes dûs aux délais sont inexistants

3. La présence d'un unique séquenceur permet la création et le maintien d'une politique d'informations centralisée à faible coût.

4. À la différence des machines asynchrones, un échange global et simultané des informations contenues dans les processeurs élémentaires est facilement réalisable par des opérateurs de diffusion.

2.3 Algorithme centralisé/algorithme distribué

Nous avons présenté de nombreuses politiques, soit centralisées, soit distribuées. Examinons quelque peu les avantages et inconvénients respectifs de ces méthodes.

Une politique d'échange d'informations centralisée conduit fréquemment à une politique de choix d'appariement centralisée et dans ce cas, le même nœud prend la responsabilité des deux politiques. Cependant, des exceptions peuvent exister comme l'algorithme GLOBAL [Zho88] où la politique d'information est centralisée par un serveur nommé LIC (*Load Information Center*) mais où la politique d'appariement est distribuée

L'entité centralisatrice aura la responsabilité de prendre toutes les décisions. Cette centralisation présente de nombreux avantages et inconvénients:

- la mise en œuvre d'une politique centralisée est aisée;
- le sur-coût pour le système reste faible, puisque les autres nœuds dédiés à l'application ne sont pas pénalisés;
- les performances sont bonnes dans des environnements avec un faible nombre de processeurs [CK82] mais dans un système dont le nombre de processeurs devient très important, la collecte des informations devient une entreprise extrêmement coûteuse [SC88]; le nœud centralisateur devant construire un vecteur de charge comportant autant d'entrées que de sites;
- le site centralisateur forme un goulot d'étranglement et par conséquent facilite la dégradation du système [BKW88];
- une panne du nœud centralisateur a des effets catastrophiques sur le système provoquant l'arrêt de tout équilibrage dynamique.

Les approches distribuées permettent de résoudre quelques problèmes causés par les méthodes centralisées sans pouvoir répondre à d'autres problèmes:

- comme les centres collecteurs d'informations sont répartis dans le système, il n'y a pas de formation de goulot d'étranglement;
- les algorithmes distribués provoquent parfois la prise de mauvaises décisions dues à la connaissance partielle ou incomplète de l'ensemble du système;
- la collecte des informations présente le désavantage dans les systèmes de type asynchrone de ne représenter que furtivement l'état réel du système du fait son évolution

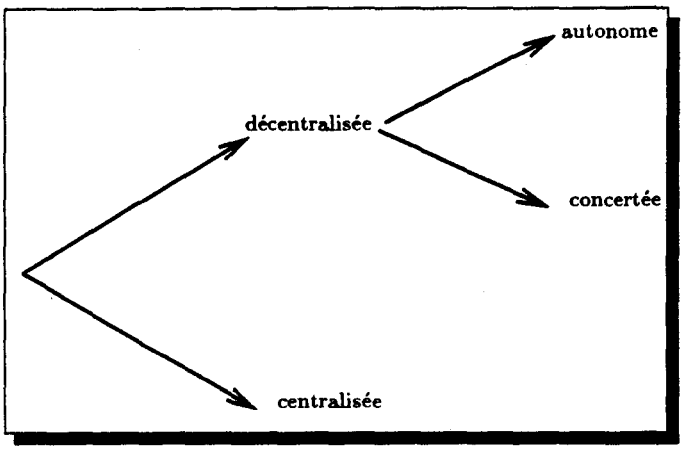


FIG. I.4 - *Différentes politiques de déclenchement*

rapide et permanente. Par conséquent, un nœud a souvent à prendre une décision soit aléatoire [Zho88], soit basée sur des critères probabilistes [Sta85].

Comme les approches pleinement distribuée et centralisée posent toutes les deux un certain nombre de problèmes, des solutions hybrides sont proposées [BGT94].

3 Politique de déclenchement

Afin d'éviter le gaspillage des ressources du système, la politique de déclenchement détermine le moment optimal d'équilibrage. Un équilibrage trop tardif provoque une augmentation de l'inactivité des processeurs, tandis qu'un déclenchement trop précoce entraîne des sur-coûts importants dus à des communications et des redistributions de données inutiles. Deux types de déclencheurs peuvent être envisagés (*cf.* figure I.4), des mécanismes décentralisés dont la responsabilité est distribuée sur l'ensemble du système et des mécanismes centralisés où l'opportunité de déclenchement est décidée par une seule entité. Les politiques de déclenchement décentralisées sont réservées aux machines asynchrones qui permettent une certaine autonomie des nœuds. Par contre, certains éléments peuvent être utilisés dans la définition d'un déclencheur sur une machine SIMD. Les techniques SIMD étant synchrones par nature, on arrive à l'utilisation de politiques centralisées.

3.1 Les mécanismes décentralisés

Les mécanismes de cette classe peuvent être partagés en deux types suivant le degré de connaissance du système.

Les mécanismes autonomes prennent la décision de déclenchement de l'équilibrage après examen de l'état local.

Les mécanismes concertés estiment l'opportunité d'un équilibrage après examen du système ou d'un sous-domaine de celui-ci.

3.1.1 Les mécanismes décentralisés autonomes

Sur une machine asynchrone, la connaissance globale de l'état du système est une opération onéreuse et délicate en particulier si la machine ne dispose pas de réseau de synchronisation. Les techniques autonomes permettent de s'affranchir de ce coût en déléguant la prise de décision du déclenchement au niveau de chaque nœud. Suivant son état local, le processeur commence une phase d'équilibrage de charges ou non. Le processus de décision obéissant généralement à une technique à base de seuils. Ce mécanisme autonome est utilisée par la majorité des algorithmes de rééquilibrage dynamique sur machines MIMD.

L'algorithme *Drafting* proposé par Ni [NXG85] est une parfaite illustration de décision autonome. Lorsqu'un processeur passe à l'état oisif en franchissant un seuil prédéterminé *L_load*, il envoie une requête de travail à ses processeurs voisins surchargés. Inversement, les nœuds de l'algorithme *Random* [GHPS94, ELZ86a] décident de transférer une partie de leur travail lorsque leur charge dépasse un seuil haut.

3.1.2 Les mécanismes décentralisés concertés

Lorsque la décision de déclenchement n'est plus effectuée au niveau local mais par consultation d'un ensemble de processeurs, on parle de mécanisme concerté. Il présente l'avantage de mieux estimer les déséquilibres du système et par conséquent de mieux déclencher l'équilibrage.

Livny et Melman [LM82] utilisent l'opérateur *UBF* (UnBalance Factor) afin d'estimer l'opportunité du rééquilibrage. Cet opérateur calcule grâce aux données fournies par la politique d'informations le degré de déséquilibre du système. S'il devient trop important, les

techniques de réarrangement dynamiques sont appelées. Il est défini de la manière suivante :

> Sur un système de n processeurs dont le nombre de tâches du nœud i est dénoté par n_i, le système est déséquilibré s'il existe deux sites i et j tels que $n_i - n_j > 1$. Le paramètre *UBF* valant alors :

$$\text{UBF} = \max_{1 \leq i,j \leq n} \left(\frac{n_i - n_j}{n_j}\right)$$

L'utilisation de l'opérateur *UBF* est relativement coûteuse car elle suppose la mise en œuvre d'une politique de mise à jour des informations sur l'ensemble du système. Si le calcul de cette valeur est envisageable sur les multi-processeurs de taille réduite, il devient beaucoup plus complexe dans les systèmes de grande taille. Pour cette raison, dans les systèmes parallèles comportant plusieurs dizaines voire plusieurs centaines de processeurs, il est utile de restreindre la concertation à un ensemble moins important de processeurs. Cet amoindrissement du domaine de concertation peut être accompli en partitionnant l'ensemble des processeurs en plusieurs sous-domaines appelés *îlots de redistribution*. La taille de ces îlots peut varier de quelques processeurs au système dans son ensemble. Willebeek-LeMair et Reeves [WLR89, WLR93] autorisent le déclenchement d'un équilibrage si la charge d'un processeur de l'îlot est supérieure d'une certaine valeur à la charge moyenne sur le domaine d'équilibrage. Ces îlots sont généralement constitués d'un processeur et de tous ceux possédant un lien physique avec lui. L'approche envisagée par Kuchen et *al* [KW91] dans l'algorithme *Persend* est assez semblable puisqu'un processeur surchargé déclenche une phase d'équilibrage, s'il existe des nœuds voisins possédant une charge k fois inférieure (k étant un paramètre de l'algorithme).

Ces mécanismes de déclenchement permettent une estimation de l'opportunité de la redistribution des charges. Ils ont été modifiés afin de profiter du particularisme des machines SIMD, comme le synchronisme et l'utilisation aisée des opérations parallel-prefix.

Les mécanismes décentralisés

Réaliser un mécanisme de déclenchement autonome sur une machine SIMD est par nature impossible. Cependant, certaines idées issues des mécanismes concertés peuvent être reprises. Par exemple, le critère de déclenchement SIMD de Mahanti et Daniels [MD93] est directement inspiré de l'opérateur UBF. Ce dernier prend la valeur ∞ et force le déclenchement dès que, un ou plusieurs processeurs deviennent inoccupés. La technique de réarrangement dynamique de Mahanti et Daniels provoque la redistribution dès que un ou plusieurs processeurs élémentaires deviennent inactifs.

L'implantation de cette technique répond au besoin de maximiser le nombre de processeurs actifs. Son grand défaut est de ne tenir aucun compte du comportement global du système. En effet, s'il est acceptable d'envisager un équilibrage de charge sur une machine MIMD pour l'inactivité d'un nœud, il est beaucoup plus discutable d'arrêter l'ensemble des opérations afin de déclencher une phase d'équilibrage sur une machine SIMD pour l'oisiveté d'un processeur élémentaire et ce pour deux raisons :

1. Les machines SIMD comportent généralement plusieurs milliers de processeurs et l'inactivité d'un seul ou de quelques-uns est peu pénalisante.
2. Le coût du rééquilibrage peut dans certains cas être supérieur au gain obtenu du fait du réarrangement dynamique.

3.2 Les mécanismes centralisés

Les mécanismes de déclenchement centralisés sont régis par une entité centralisatrice. À la vue des paramètres fournis par la politique d'informations, cette entité décide ou non de déclencher un rééquilibrage. Sur les machines MIMD, la responsabilité du déclenchement est généralement confiée à un nœud.

Les mécanismes centralisés

Les mécanismes centralisés sont bien adaptés aux machines de type SIMD qui par nature obéissent à une entité centralisatrice. L'unité centrale contrôlant l'ensemble des processeurs élémentaires peut décider d'après les informations qu'elle possède d'effectuer ou non une nouvelle répartition de l'ensemble des données. Certains mécanismes ne prennent pas en compte l'état du système et provoquent un rééquilibrage régulier. Ils sont dits *périodiques*. Les autres déclencheurs qui ne sont pas réguliers et dépendent de l'état du systèmes sont appelés *adaptatifs*. Les paragraphes suivants sont consacrés à l'étude des mécanismes centralisés sur les machines SIMD.

3.2.1 Les mécanismes centralisés périodiques

Pour un ensemble de processeurs synchronisés, la méthode la plus aisée pour le déclenchement est de forcer l'ensemble des processeurs à effectuer un réarrangement des données suivant une fréquence fixée au préalable par l'utilisateur ou au moment de la compilation. La synchronisation globale présente sur machines SIMD rend cette méthode facile à implémenter. Cependant, la difficulté d'estimation d'une période pour un tel déclencheur rend la définition

de cette fréquence malaisée. Une valeur élevée de la période entraînera des rééquilibrages fréquents inutiles pour le système et dégradera les performances de la machine. Une faible valeur risque de laisser le système fonctionner sur une période élevée avec un faible nombre de processeurs. Dans sa technique de rééquilibrage dynamique, Henrich [Hen94] utilise un déclencheur périodique dont la fréquence est élevée. Ce nombre d'appels élevé est compensé par le faible sur-coût induit par la redistribution. Une amélioration d'un tel mécanisme de déclenchement est proposé au chapitre III.

3.2.2 Les mécanismes centralisés adaptatifs

Les mécanismes centralisés adaptatifs sont caractérisés par une évolution temporelle du critère de déclenchement. Nous avons discerné deux types de mécanisme adaptatifs :

Les déclencheurs instantanés forcent la redistribution des données suivant l'état du système à un instant donné. Aucune supposition n'est effectuée sur l'état futur de la machine.

Les déclencheurs prédictifs prennent en compte différents paramètres du système et leur évolution avant d'autoriser un éventuel équilibrage.

Les mécanismes centralisés adaptatifs instantanés

Une politique de déclenchement fort simple, intuitive et qui offre des performances tout à fait correctes compare le nombre de processeurs oisifs avec le nombre de processeurs occupés [PFK91, PFK93, KK92, FM90]. Si ce ratio tombe en dessous d'un certain seuil (par exemple, moins de 80% des processeurs sont occupés), une phase de redistribution des données est alors déclenchée. Frye et *al* ont estimé empiriquement sur un problème de parcours d'arbre, qu'un seuil fixé à 2/3 fournissait des résultats optimums. Cependant, cette valeur n'est représentative que pour ce type de problèmes et devra être modifiée pour d'autres applications.

Formellement, si P est le nombre total de processeurs, $A(t)$ le nombre de processeurs actifs au temps t et x est le ratio d'activité souhaité (x est compris entre 0 et 1), il y aura une phase de déclenchement si :

$$A(t) \leq xP$$

Comme dans le cas du choix de la fréquence d'équilibrage pour un mécanisme périodique,

le choix d'une valeur correcte pour le ratio x est un problème délicat. On peut imaginer de faire varier x par affinements successifs en fonction de l'évolution observée du système.

Dans le chapitre III, nous présenterons un tel déclencheur qualifié de *statique adaptatif.*

Les mécanismes centralisés adaptatifs prédictifs

Pour tenir compte de l'évolution dynamique et temporelle du système des déclencheurs dynamiques sont apparus. Ils permettent de prendre en compte outre l'état instantané de la machine, divers paramètres du système comme le temps des précédentes phases d'équilibrage, l'estimation du gain probable réalisé par le rééquilibrage... Les différentes approches existantes anticipent la durée d'une phase d'équilibrage en réalisant une moyenne des durées des précédents rééquilibrages.

Deux mécanismes ont été proposés. Le premier est basé sur la maximisation du taux de travail effectué par les processeurs élémentaires actifs [PFK91, PFK93] ; le second déclenche une répartition des données si l'inactivité cumulée des processeurs oisifs devient trop importante.

Un mécanisme basé sur le taux de travail des processeurs élémentaires

Soit $W(t)$ la quantité de travail effectuée mesurée en nombre de « processeurs actifs seconde » depuis le dernier rééquilibrage. $W(t)$ correspond à l'intégrale de $A(t)$. L est la durée prévue d'une phase de redistribution. Le ratio $\frac{W(t)}{L+t}$ indique le taux moyen d'activité incluant la phase d'équilibrage. Il est comparé au taux d'activité, $A(t)$, c'est-à-dire le nombre de processeurs actifs. Une phase d'équilibrage est alors déclenchée au temps t, si :

$$\frac{W(t)}{L+t} \geq A(t)$$

Un mécanisme basé sur l'inactivité des processeurs élémentaires

Expérimentalement, on constate que la décroissance de la charge sur les machines SIMD se présente souvent sous le même aspect. La figure I.5 en est un exemple. La surface S_1 représente en quelque sorte le travail perdu par l'inactivité des processeurs oisifs. Le déclencheur proposé [KK92, FDM93] tente de limiter l'accroissement trop important de cette surface. Ce déclencheur est étudié dans le chapitre III. L'équilibrage est déclenché lorsque la surface correspondant à l'inactivité forcée par la redistribution des données (surface grisée S_2) devient plus importante que la surface des processeurs inactifs (S_1). Si le rééquilibrage avait été réalisé plus tôt, alors le sur-coût provoqué par la phase d'équilibrage aurait été supérieur à

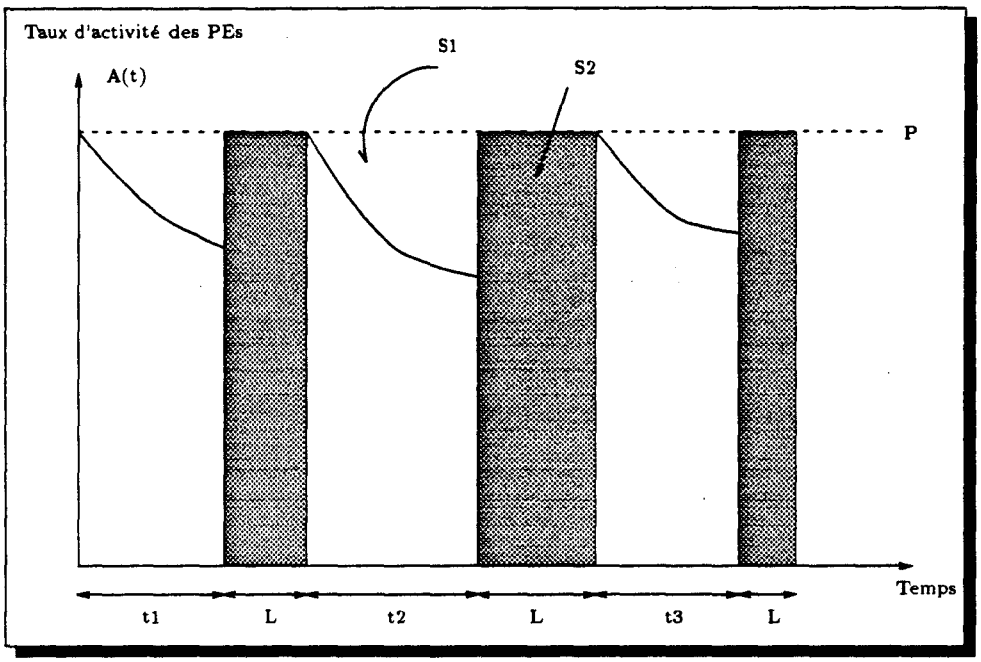


FIG. I.5 - *Mécanisme basé sur la limitation de l'inactivité des processeurs élémentaires*

l'overhead dû à l'inactivité et réciproquement. Ce mécanisme permet d'équilibrer l'inactivité des processeurs élémentaires et le coût d'une redistribution.

Une phase de rééquilibrage est déclenchée si :

$$P \times t - W(t) \geq L \times P \Longleftrightarrow S_2 \geq S_1$$

4 Politique de sélection

La politique de sélection détermine, si la participation d'un site à l'équilibrage est ou non propice. Dans l'affirmative, elle détermine quel rôle jouera chaque site ; dans la plupart des méthodes existantes, on distingue deux cas :

- les nœuds accusent un déficit de charge, ils sont alors désignés comme receveurs d'objets ;
- les nœuds contiennent un excédent de charge, ils devront alors déléguer une partie de leur charge aux nœuds récepteurs.

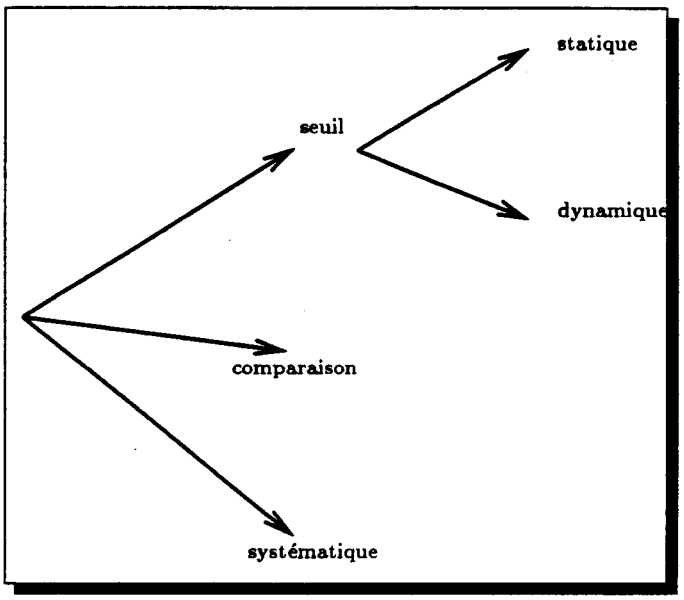


FIG. I.6 - *Différentes politiques de sélection*

Les nœuds peuvent parfois au cours de la même technique d'équilibrage jouer les deux rôles, émetteurs ou récepteurs, on parle de stratégies mixtes. Trois classes peuvent être distinguées dans la politique de sélection comme indiquée sur la figure I.6.

De nombreuses stratégies de sélection sont fondées sur une politique à base de *seuils* où un nœud compare sa charge locale avec un ou plusieurs seuils. Suivant le résultat de cette comparaison, le nœud deviendra émetteur ou récepteur de charge. Des stratégies de *comparaisons* peuvent également être employées, où le processeur adopte un comportement en fonction de l'état de l'ensemble ou d'un sous-ensemble de nœuds.

Certaine méthodes dites *systématiques* ne tiennent pas compte de l'état actuel des nœuds. Ces techniques sont basées sur des résultats théoriques. Elles provoquent des échanges de travail dans un ordre donné et désignent alternativement les nœuds comme émetteurs, receveurs ou inactifs.

4.1 La politique de seuil

Une décision de déplacement ou d'acceptation de charge est généralement prise en comparant la charge du nœud à un seuil. Le site est soit considéré comme surchargé, soit considéré comme faiblement chargé (*cf.* figure I.7). Un site lourdement chargé pourra tenter d'émettre des objets tandis qu'un site faiblement occupé pourra être récepteur. Dans les algorithmes

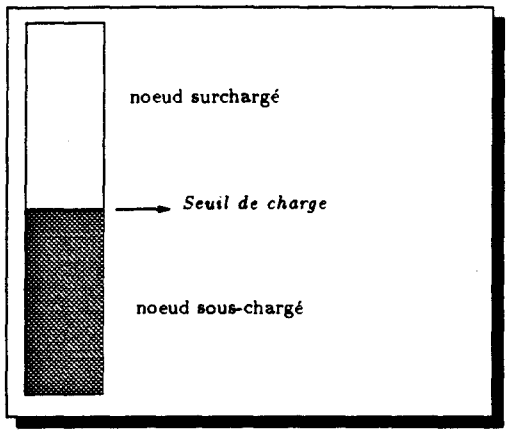


FIG. I.7 - *Stratégie à seuil unique*

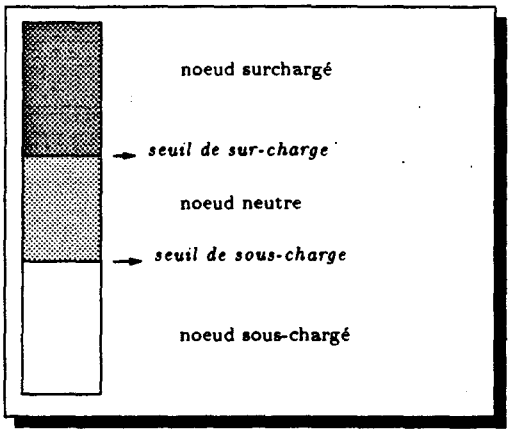


FIG. I.8 - *Stratégie à double seuil*

de partage de charges de [ELZ86a, KW91, DTJ89], le seuil est utilisé afin d'accepter ou de refuser un processus distant. Malheureusement, la définition d'un seuil unique a tendance à être instable et à provoquer soit l'effet ping-pong [Har86, RM90] (cas où un objet est transféré d'un site à un autre, puis re-transféré dans le sens inverse), soit l'effet boomerang (l'objet déplacé une première fois est transféré vers une troisième machine). D'autres algorithmes utilisent deux seuils (*cf.* figure I.8) pour garantir une stabilité plus importante des techniques d'équilibrage. Une politique à double seuil permet de classer les nœuds en trois catégories distinctes : receveur potentiel, neutre et envoyeur potentiel. Un nœud se trouvant dans l'état receveur potentiel indique qu'il est susceptible de recevoir un ou plusieurs processus distants. Un processeur se trouvant dans l'état envoyeur potentiel signifie que certains de ses processus peuvent être transférés. Un site se trouvant à l'état neutre indique qu'il ne participera pas à l'équilibrage de tâches. L'introduction de ces deux seuils évitent donc au processeur de passer trop rapidement de l'état surchargé vers l'état sous-chargé. Par contre, les algorithmes admettent une différence de charge incompressible, égale à la différence entre les deux seuils. Ils ne permettent pas d'effectuer un équilibrage parfait sur l'ensemble du système, théoriquement possible avec un seuil unique.

Ce double seuil a été largement implémenté, soit dans le cadre de systèmes d'exploitations comme *Gatos* [BF92], ou encore dans différents algorithmes de redistribution de tâches [NXG85, CLZ92, Zho88, LK87] ou de données sur machines MIMD [SW91].

Les seuils utilisés lors des différentes stratégies sont généralement des valeurs fixées au préalable, ce qui parfois ne permet pas de suivre avec exactitude l'évolution du système. Eager et *al* [ELZ86a] utilisent des seuils fixés volontairement à un niveau très bas, afin d'interférer le moins possible avec les changements d'état du système. Certaines stratégies proposent une modification dynamique de la valeur des seuils afin de prendre en compte l'évolution de la charge globale du système. Lee et Towley [LT87] proposent un algorithme itératif dans lequel la valeur du seuil est périodiquement mise à jour.

De manière semblable, en utilisant une méthode basée sur le gradient, Pulidas et *al* [PTS88] développent un algorithme dont le seuil est adaptatif et peut réagir à des situations de surcharge subite. Chaque hôte calcule le délai d'attente des processus comme une fonction du taux d'arrivée et le compare au délai minimum communiqué par les autres sites. Ces informations sont utilisées pour mettre à jour la nouvelle valeur du seuil.

La politique de seuil

Les politiques à base de seuil unique ont été adaptées sur les architectures SIMD [PFK93, KK92, FM90]. Les PEs inoccupés sont considérés comme des receveurs potentiels, tandis que ceux possédant une charge non nulle sont considérés comme des émetteurs potentiels de données. Cela revient indirectement à considérer une valeur d'un seuil unique égale à 1. On peut cependant s'interroger sur la validité d'un tel seuil qui considère un PE chargé de plusieurs milliers d'entités sur le même pied qu'un PE contenant 2 composants.

Certains algorithmes de redistribution de données SIMD se sont inspirés des politiques à double seuil. Les algorithmes *Stingy Sharing* et *Generous Sharing* [MD93] utilisent une politique à double seuil dans leur technique de redistribution. Chaque processeur élémentaire peut suivant sa charge être classé comme *nécessiteux*, *riche* ou bien *content*. Les PEs riches pouvant céder une partie de leur travail aux PEs nécessiteux.

Comme nous le verrons dans le chapitre III, l'utilisation des opérations de *parallel-prefix* sur une architecture SIMD permet de calculer aisément une nouvelle valeur de seuil et donc d'adapter le seuil à l'évolution globale du système.

4.2 Les méthodes systématiques

La méthode systématique consiste à définir suivant un schéma fixé au préalable les nœuds émetteurs et récepteurs. L'état des processeurs à un instant donné n'est pas pris en compte.

Durant la même phase, les sites peuvent alternativement jouer plusieurs rôles. Par exemple, Vornberger [Vor86] dans une implémentation parallèle du Branch-&-Bound, sur un réseau d'ordinateurs personnels, utilise un « envoi à la demande ». Par contre, en adaptant sa méthode sur un réseau de Transputers [Vor90], il autorise chaque site à envoyer périodiquement une partie de sa charge vers ses collègues. Chaque Transputer devient et ceci de façon systématique et asynchrone, un receveur et un émetteur de charges. Cette méthode est applicable dans ce cas du fait de la topologie en étoile qui a été utilisée, cette méthode devient difficilement applicable dans des applications où le travail est concentré sur un très petit nombre de processeurs.

Une variante des méthodes systématiques est la méthode d'échange dimensionnelle [XL92]

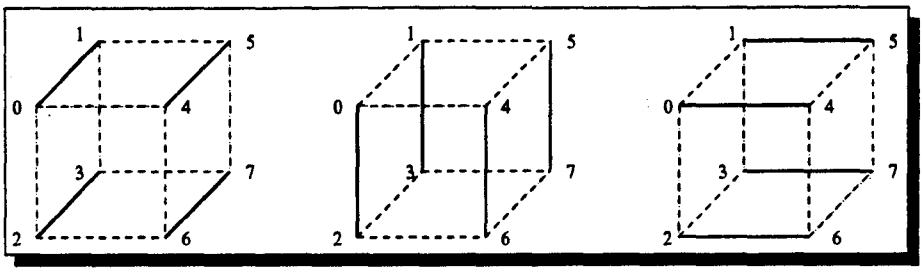
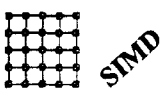


FIG. I.9 - *Stratégie d'échange dimensionnel pour un hypercube de 8 processeurs. L'équilibrage est réalisé successivement dans chacune des 3 dimensions. Les processeurs échangeant leur travail sont ceux reliés par une arête grasse*

analysée de manière théorique par Cybenko [Cyb89]. Dans cette technique, l'équilibrage est effectué dimension par dimension. Chaque dimension correspond à un sous-ensemble de toutes les paires de processeurs connectés. Chaque dimension est équilibrée tout à tour et le processus est répété dans son ensemble jusqu'à l'obtention d'un état satisfaisant. Les processeurs sont alternativement sollicités pour recevoir ou fournir une partie de leur travail. Cette stratégie a tout d'abord été introduite pour réaliser une redistribution de tâches sur des topologies hypercubes. Dans le cas d'un hypercube de n processeurs, l'équilibrage est réalisée successivement dans chacune des $\log n$ dimensions comme indiqué sur la figure I.9. L'échange dimensionnel nécessite une synchronisation de l'ensemble des processeurs afin de procéder à un échange de travail simultané. La synchronisation globale est particulièrement difficile à réaliser dans les systèmes de très grande taille. Les machines disposant d'un mécanisme matériel de diffusion se prêtent bien à cette approche. La technique de l'échange dimensionnel peut être adoptée sur d'autres types de topologie en effectuant une projection de l'hypercube sur l'architecture cible.



Les méthodes systématiques

À notre connaissance, la technique de l'échange dimensionnel n'a pas été implémentée sur les machines SIMD. Nous avons proposé une adaptation de cette méthode sur une architecture SIMD, nous avons constaté son très bon comportement du fait de la synchronisation globale et permanente de la machine.

4.3 Les méthodes de sélection par comparaisons

Une méthode de sélection par comparaison est caractérisée par la prise en compte pour chaque processeur de l'état des autres processeurs du système. À la différence des stratégies de sélection par seuil, où un nœud base son comportement futur sur son seul état, un site est candidat à être receveur ou émetteur en fonction du comportement ou de l'état des autres

nœuds.

La prise en compte de l'état des autres processeurs peut se faire :

Sur un domaine de petite taille. Le processeur examine l'état de son voisinage afin de prendre une décision sur son comportement futur. Dans les algorithmes proposés par [LMR91, WLR89], un processeur participe à une redistribution du travail si sa différence de charge avec ses voisins est supérieure à une certaine valeur. Cette valeur peut être soit fixée par avance [WLR93], soit mise à jour de manière évolutive. [LM93] constate que pour atteindre l'équilibre la différence de charge Δ doit être la plus faible possible. Mais, le choix d'un tel Δ provoque un équilibrage fréquent et un overhead de communications important. Pour cette raison, la valeur de Δ est augmentée périodiquement afin de diminuer les activités d'équilibrage.

Dans, l'algorithme *Persend* [KW91], chaque processeur examine périodiquement la charge de l'ensemble de ses voisins afin de savoir si l'un de ceux-ci ne possède pas au moins k tâches de moins que lui. Dans ce cas, ce processeur est receveur ; d'après les différentes expériences menées, il semble que la valeur $k = 1$ offre les meilleurs résultats.

Sur l'ensemble du système. Le processeur décide de son comportement futur après interrogation d'une partie des processeurs situés à une place quelconque du système. Dans la stratégie *Lowest* [Zho88] un site est désigné comme receveur si sa charge est la plus faible parmi L_p sites sondés aléatoirement. De façon semblable, dans l'algorithme de [Sta84] si la différence entre la charge locale et la charge du site le moins chargé du système est supérieure à un certain biais, un processus est transféré vers ce dernier. Cependant, comme le remarque [DTJ89], le coût moyen de ce type d'algorithmes d'équilibrage est très élevé à cause des nombreux messages échangés.

Les méthodes de sélection par comparaisons

Nous proposons dans le chapitre III une politique de sélection adaptée aux machines synchrones. Un nœud devient émetteur si sa charge est supérieure à la charge moyenne d'un domaine appelé *îlot de redistribution.*

5 Politique de désignation locale

Le but de la politique de désignation locale est de sélectionner les entités désignées pour le transfert. Il existe 2 possibilités pour réaliser l'exportation de processus :

- soit réaliser un transfert d'un processus dont l'exécution partielle a déjà débuté sur le nœud source. Un tel transfert est qualifié de préemptif ;
- soit réaliser un placement de tâches. Une fois le processus placé, il sera exécuté sur le nœud destinataire jusqu'à sa terminaison. Un transfert en cours d'exécution n'est pas autorisé.

Dans une première partie, nous nous intéresserons à cette distinction essentielle et analyserons les causes de son inadaptation au modèle SIMD. Diverses méthodes de désignation locale de processus seront ensuite introduites.

5.1 Les systèmes préemptifs

Le transfert *préemptif* autorise le déplacement d'une tâche dont l'exécution partielle a déjà été réalisée. Le transfert de tâches préemptif, souvent appelé *migration*, s'adapte plus facilement aux changements dynamiques extrêmement rapides du système [BS85]. Il permet aussi d'offrir une meilleure tolérance aux pannes en déplaçant et redémarrant des processus de nœuds défectueux. Nombre de détracteurs font remarquer la difficulté de conservation du contexte et de l'interactivité des processus. Il est nécessaire de tenir compte de l'image virtuelle de la mémoire, des différents tampons d'entrées/sorties, des pointeurs de fichiers, du contexte du processeur, des différents fichiers ouverts et des différentes interactions avec l'extérieur. La mise en œuvre d'une politique migratoire suppose en plus une connaissance approfondie de l'architecture cible et du système d'exploitation utilisé. Il est important de ne pas effectuer systématiquement la migration, car dans ce cas les performances sont faibles [LLM88]. On peut donc s'interroger sur sa véritable utilité. Eager et *al* [LEZ88], après avoir comparé la migration et le placement de processus, estiment que la migration n'améliore que faiblement les temps de réponse du système, sauf dans des cas extrêmes.

La mise en place d'un système préemptif nécessite la présence de quatre critères [DO91], à savoir :

Transparence La transparence se caractérise par deux choses : (i) le processus migré doit avoir le « sentiment » de travailler dans le même environnement que sur la machine originale et (ii), pour le monde extérieur, le processus doit sembler ne jamais avoir

bougé. Toute opération possible sur un processus non migré (envois de signaux par exemple) doit pouvoir être appliquée sur le processus migré.

Dépendances résiduelles minimales Les dépendances résiduelles caractérisent les structures de données « laissées » par une tâche migrée sur son nœud source. Elles sont indispensables pour maintenir la transparence mais peuvent dégrader les performances du système si le processus est migré à plusieurs reprises.

Performance La performance signifie que la migration doit être profitable au système. Dans le cas idéal, le processus distant doit s'exécuter avec autant d'efficacité que si l'exécution avait été locale.

Automatisme Le mécanisme migratoire doit être automatisé dans son choix du meilleur site disponible, l'utilisateur ne doit pas intervenir pour guider le système.

La mise en œuvre de ces critères sur le système d'exploitation UNIX est délicate à réaliser, notamment à cause de l'importance du contexte à déplacer et du coût élevé de son transfert sur une autre machine. Une solution généralement proposée est de construire un nouveau système d'exploitation plus ou moins basé sur UNIX dont le noyau incorpore la migration. Les systèmes d'exploitation spécifiques V-SYSTEM [Stu88], CHORUS [RAA+88] ou encore CHORUX/MIX [Phi93] intègrent la migration.

Récemment, des approches migratoires orientées-objets [JBSV92] ont été proposées. L'utilisation de méthodes orientées-objets offre des avantages dans la maintenance, la modification et l'extension. Une classe appelée *unité de transfert* est définie, chaque instance de cette classe correspondant à un transfert de charge pendant l'exécution. Lorsque le programmeur décide d'utiliser ces méthodes sur un autre système d'exploitation ou pour une autre architecture, il doit « simplement » réécrire certaines opérations comme *Send()*.

Les systèmes préemptifs

La distinction importante entre les approches préemptives et non préemptives peut s'appliquer au monde SIMD. Le transfert préemptif qualifie un transfert de données qui ont déjà été partiellement calculées. Par exemple, dans les techniques de tracking d'un électron, il est tout à fait possible de déplacer l'électron. Cette migration entraîne cependant le transfert du contexte lié à cet objet. Lors du rassemblement des informations disséminées dans le système, il est nécessaire de communiquer aux particules le nouvel emplacement de l'électron dans le système.

5.2 Les différentes méthodes de désignation locale

Les stratégies de désignation locale pour le systèmes préemptifs se basent sur un principe commun : il est avantageux de transférer un processus possédant un résidu d'exécution important. On tente alors d'évaluer le résidu d'exécution par des techniques de filtrage (*cf.* fig I.10). Les méthodes de filtrage proposent une sélection simple des processus au détriment parfois de la transparence. En règle général, pour les systèmes non préemptifs des techniques systématiques sont appliquées. Elles effectuent un choix aléatoire ou suivant un ordre déterminé des processus à transférer.

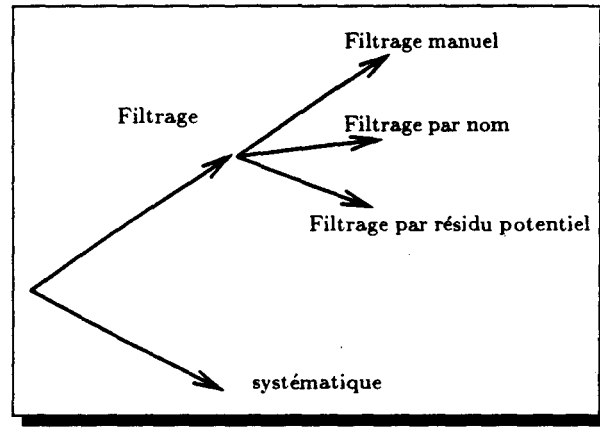


FIG. I.10 - *Différents politiques de désignation locale*

5.2.1 Les méthodes systématiques

La procédure la plus usitée pour la sélection d'un processus est de considérer toutes les tâches existantes comme équivalentes. Elles sont par conséquent toutes susceptibles d'effectuer un transfert. Cette approche présente l'avantage d'être immédiate quant au choix du processus à sélectionner. Elle évite le cas critique des méthodes de filtrage dans lesquelles le nœud passe son temps à envisager si le déplacement d'un processus est profitable ou non. La structure utilisée est soit une pile soit une file de type FIFO (First In-First Out).

La désignation locale peut également être effectuée de façon aléatoire. Tritsher et *al* [TZB94] proposent une implantation sélectionnant aléatoirement les processus qui seront transférés sur un site distant. Cette technique permet en outre le déplacement simultané de plusieurs tâches vers un même nœud. Ce transfert de plusieurs processus évite le déplacement d'un unique processus dont le résidu d'exécution peut se révéler faible.

Les méthodes systématiques

La méthode de désignation locale systématique a largement été adoptée sur les machines SIMD car sa mise en œuvre est évidente. De plus,

- il est possible comme Kumar [KK92] et nous-mêmes le proposons de déplacer simultanément plusieurs données de chaque processeur. Cela permet de limiter les inconvénients de déplacement d'une donnée ultérieurement peu productive ;
- une méthode systématique est réalisée par une structure de données de type file ou pile. La sélection des données à transférer est immédiate. Par contre, l'utilisation d'une stratégie non systématique entraîne la création d'une table d'indirection supplémentaire dans la désignation locale. Cette table d'indirection permet à chaque PE de « pointer » sur une adresse locale différente.

5.2.2 Les méthodes utilisant le filtrage

Afin de sélectionner la meilleure entité à déplacer, un certain nombre de méthodes a été proposé : demande de sélection par l'utilisateur, examen des processus... Nous allons présenter les méthodes les plus couramment utilisées et examiner si une adaptation sur une architecture synchrone est possible.

Le filtrage manuel

Dans certains systèmes d'exploitation autorisant la migration, une commande spéciale est mise à la disposition de l'utilisateur afin d'autoriser une exécution sur un site distant. Par exemple, dans le système d'exploitation SPRITE [DO91], deux commandes lancées par l'utilisateur **pmake** et **Mig** permettent le transfert préemptif. La commande **pmake** est semblable à l'utilitaire **make** d'UNIX [Fel79], mais utilise la migration de processus pour exécuter autant de commandes en parallèle que le système possède d'hôtes inactifs. La commande **Mig** prend en argument une commande du shell et utilise la migration pour exécuter le processus sur un site inoccupé. CONDOR [LLM88] se base sur un filtrage manuel pour autoriser la migration de tâches. Le filtrage manuel assure de bons résultats dès que l'utilisateur lance des migrations « intelligentes » [BSS91], mais on peut rapprocher à cette méthode son manque de transparence et son inadéquation pour des utilisateurs peu expérimentés.

Le filtrage par nom

La méthode du filtrage par nom associe le nom d'une tâche à la possibilité ou non d'effectuer la migration. Elle permet d'éviter à l'utilisateur le choix des processus à transférer. Une table distribuée sur l'ensemble des nœuds ou contenue par un site centralisateur autorisera ou non, en comparant le nom de la tâche à la liste qu'elle possède, d'effectuer la migration.

Zhou [Zho88], classe grossièrement les processus en deux grandes familles : les « gros » jobs qui sont susceptibles d'être transférés et les « petits » travaux qui seront obligatoirement effectués localement. Par exemple, effectuer le listage d'un répertoire avec la commande `ls` est un processus qu'il serait vain de vouloir transférer, tandis que le lancement d'un traitement de texte créera un processus qui devrait normalement durer plusieurs secondes. Quand un processus est déclaré éligible pour une migration et que le site sur lequel il se trouve est lourdement chargé, un hôte inutilisé est alors recherché pour le transfert. Cette restriction de déplacement effectuée sur certains processus ne semblent guère affecter les résultats [FZ87]. Ce mécanisme a d'ailleurs été repris et amélioré afin de développer l'environnement UTOPIA [ZWZD93]. Lorsqu'une commande est entrée par l'utilisateur, l'éligibilité de cette dernière est vérifiée ainsi que l'estimation de la demande de ressource. Si le transfert est possible, un programme se chargera de son placement.

La méthode du filtrage par nom permet une transparence beaucoup plus importante que le filtrage manuel. Cependant, l'éligibilité des candidats au transfert est souvent basée sur une estimation statistique. [FZ87] ont analysé le comportement des 30 commandes UNIX parmi les plus usitées pendant plusieurs mois pour définir une liste type des candidats au transfert. Par exemple, les commandes `cc` ou `troff` sont transférables, tandis que les commandes `sort` ou `wc` sont exécutées localement. Cette autorisation de transferts basée sur un comportement peut parfois provoquer des migrations totalement inutiles ; l'exécution d'une compilation en langage C pouvant fort bien s'arrêter au bout de quelques secondes sur une erreur syntaxique ou pouvant durer plusieurs dizaines de minutes.

Le filtrage par résidu potentiel

La méthode du filtrage par résidu potentiel est caractérisée par la migration des processus dont les durées attendues d'exécution à distance seront les plus importantes. Elle est utilisée uniquement dans le cadre du transfert préemptif. Plusieurs critères peuvent être utilisés pour l'estimation du résidu d'exécution :

Migration des processus les plus âgés Cette méthode repose sur le paradoxe de la vie des processeurs : plus un processus a consommé de temps CPU, plus les chances qu'il ne se termine pas dans un avenir immédiat sont importantes. Le terme « âge » désigne le temps CPU déjà consommé. Par exemple, [Cab86] observe que 40% des processus ayant vécu T secondes vivront $2T$ secondes. Le filtrage par âge est mis en œuvre dans le

système d'exploitation distribué CHORUS [Phi93] : lorsque le système décide d'effectuer une migration, le processus choisi est le plus âgé. Afin d'éviter, que le processus déplacé soit transféré de site en site, un temps minimum d'exécution locale, au moins équivalent au temps moyen de migration, lui est imposé.

Migration des processus les plus jeunes Cette stratégie autorise la migration des processus nouvellement créés. Elle propose la méthode inverse de la stratégie précédente, puisqu'elle suppose que les processus âgés vont se terminer plus tôt que les processus dernièrement créés [GS93]. Cette technique a été mise en œuvre dans V-SYSTEM [Stu88], seules les processus les plus jeunes sont candidats à la migration.

Filtrage par ascendance Cette méthode est caractérisée par la construction d'un arbre ou d'un graphe [KW91] dans lequel les tâches sont les nœuds et chaque tâche pointe sur ses sous-tâches. Les tâches qui sont plus proches de la « racine » ont un résidu d'exécution supposé plus important que celles qui en sont plus éloignées. Le filtrage par ascendance migre les tâches les plus proches de la racine, chaque processeur construisant son propre graphe localement.

Les méthodes utilisant le filtrage

Les méthodes de filtrage manuel et de filtrage par noms ne sont pas utilisables sur une machine SIMD. Par contre, les techniques de filtrage par résidu potentiel sont adaptables sur machines synchrones. Powley et *al* [PFK91, PFK93] dans la résolution d'un problème nécessitant le parcours d'un arbre créé dynamiquement, choisissent d'exporter les objets qui *a priori* généreront un sous-arbre de taille importante. Cette désignation s'apparente à un filtrage par résidu potentiel. Cependant, son utilisation doit être un peu nuancée car les expérimentations effectuées montrent que, la sélection aléatoire d'objets offre des performances toutes aussi correctes.

6 Politique d'appariement

La politique d'appariement a pour but de trouver un partenaire adéquat à un nœud receveur ou envoyeur. L'appariement réalisé est généralement une association (à un nœud est associé un autre nœud) mais peut dans certain cas associer un nœud avec plusieurs partenaires. La politique d'appariement est bien évidemment étroitement liée à la politique d'informations ; il est par exemple beaucoup plus aisé d'introduire une politique d'appariement centralisée si la politique d'information est également centralisée.

Les politiques d'appariement sont extrêmement nombreuses. La classification que nous

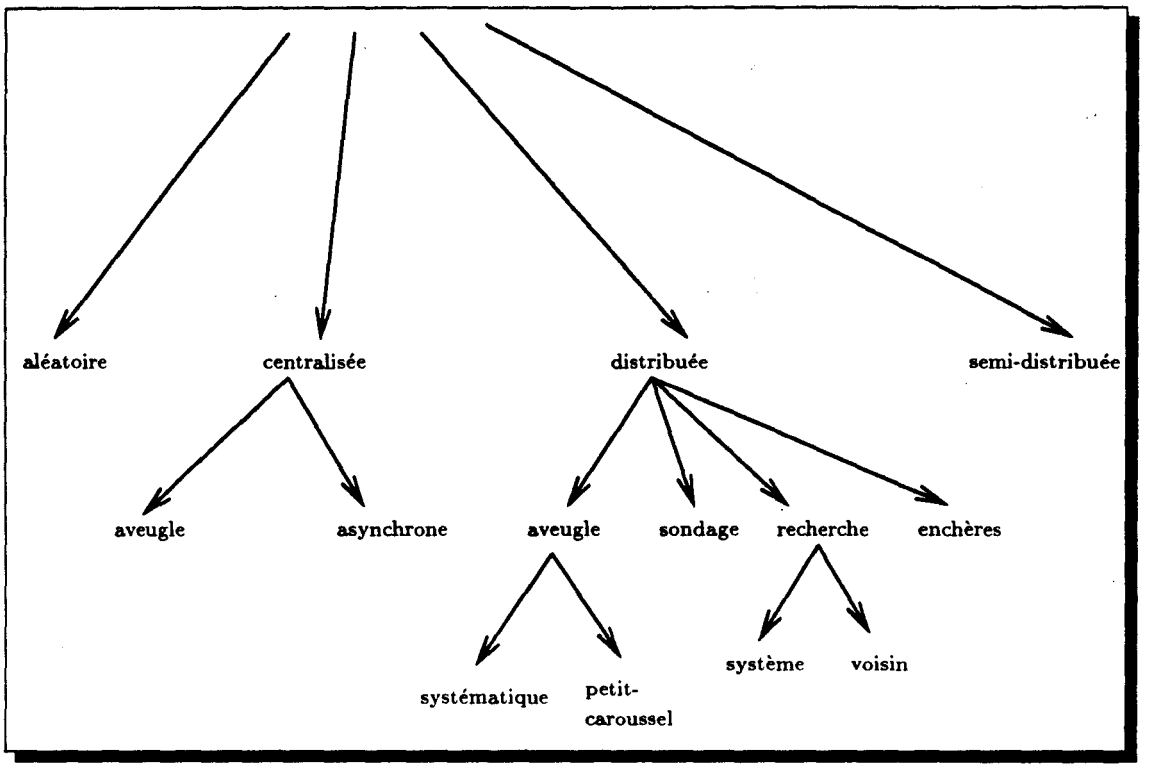


FIG. I.11 - *Classification des différentes politiques de localisation*

proposons sur la figure I.11 est partielle mais permet d'appréhender les stratégies les plus importantes. Les méthodes les plus simples comme les stratégies aléatoires offrent des performances correctes pour un coût très faible. Dans les stratégies centralisés, un nœud est chargé de la gestion de recherche des partenaires. La responsabilité de la recherche de partenaires est répartie dans le système dans les politiques distribuées. Quelques approches ont mêlées ces deux stratégies afin de créer des politiques semi-distribuées.

6.1 Choix du nœud aléatoire

Une politique d'appariement aléatoire est caractérisée par l'assignation d'un processus à un site pris au hasard.

La décision d'envoyer un ou plusieurs processus est effectuée sans concertation préalable avec un ou plusieurs sites. Le nœud décide de lui même vers quel partenaire adresser une partie de sa charge. La décision peut être prise par l'émetteur qui envoie une partie de sa charge vers un processeur sélectionné aléatoirement, ou à l'inverse, par le receveur qui demande du travail à un nœud pris au hasard.

Cette technique aléatoire est mise en œuvre dans l'algorithme *Random Polling* [KGR91]. Un nœud demande à un site distant une partie de sa charge (suivant le principe de partage de travail α/β). La machine sélectionnée n'y a pas la possibilité de refuser la demande de travail.

Le travail peut être diffusé de façon aléatoire sur des domaines de taille variable :

- sur un voisinage de quelques processeurs [AGM93, KW91] ;

- sur l'ensemble du système comme l'algorithme *Random* [ELZ86b, Zho88, RSZ89].

La stratégie aléatoire est intéressante car le sur-coût de la politique d'appariement induit par l'envoi d'un ou plusieurs processus reste faible. Il n'y pas d'examen du système et de recherche du meilleur nœud cible possible. Par contre, le désavantage de cette technique est éventuellement d'envoyer la tâche à un nœud déjà lourdement chargé. Des garde-fous peuvent être installés en autorisant un nœud à re-transférer une charge [SK90b] vers un autre site sélectionné aléatoirement. Cette technique a tendance à devenir instable [ELZ86a] s'il n'y a pas de limites aux re-transferts. Quelque soit l'état de départ, il arrive un moment où *tous les nœuds* passent leur temps à s'échanger des processus sans les exécuter.

Choix du nœud aléatoire

La stratégie d'appariement aléatoire peut aisément s'adapter aux architectures SIMD. Il est possible par l'utilisation d'opérations data-parallèles d'associer un PE à un autre de manière unique. Cette opération, nommée permutation des PEs se réalise en quelques instructions élémentaires. Cependant, dans le cadre de la sélection aléatoire de partenaire, comme dans le cas MIMD, une donnée peut éventuellement être envoyée vers un PE déjà fort occupé.

Pour un coût relativement faible des instructions data-parallèles permettent la mise en place d'une politique d'appariement de meilleure qualité.

La méthode d'appariement aléatoire présente donc peu d'intérêt sur une machine SIMD pendant l'exécution d'une application. Elle peut, par contre, trouver son utilité dans une répartition statique des données précédant l'exécution. Pour les algorithmes générateurs (les données peuvent engendrer d'autres données), on peut à partir d'une donnée unique créer un certain nombre de fils et les disperser de manière aléatoire sur le système.

6.2 Les stratégies d'appariement centralisées

Une stratégie centralisée désigne un nœud comme serveur. Il détient des informations qui seront utilisées pour la recherche de nœuds destinataires.

6.2.1 L'appariement aveugle centralisé

Une méthode centralisée aveugle ne repose sur aucun examen de l'état global du système,. Elle tient compte de ses actions précédentes pour modifier son comportement futur.

La technique dite du *Grand Carrousel* repose sur ce principe. Par exemple, dans l'algorithme développé par Xu et *al* [XH93], une variable globale `TARGET` est stockée par le processeur 0. Initialement, cette variable pointe sur le processeur numéro 1. À chaque fois, qu'un processeur recherche du travail, il envoie une requête au processeur 0, qui lui transmet l'adresse du processeur sur lequel il pointe. Simultanément, la variable `TARGET` est incrémentée. Cette stratégie assure que les demandes de travail seront dispersées sur le système, sans éviter par contre le problème du goulot d'étranglement sur le processeur 0.

Pour éviter le problème de la multiplication des accès sur le processeur 0, une méthode hiérarchique [KGR91, KGV94] peut être utilisée. Grâce à cette technique, toutes les requêtes de lecture du processeur 0 sont exécutées et combinées à des niveaux intermédiaires. La figure I.12 présente l'utilisation d'une telle méthode pour un hypercube de 8 processeurs. Un arbre est construit à partir de l'hypercube, chaque processeur étant une feuille de l'arbre. Lorsqu'un processeur souhaite incrémenter la variable `TARGET`, il envoie une requête vers son père. Le père retient cette demande pendant au plus δ unités de temps avant de l'envoyer à son propre père. Si une requête lui arrive pendant cette période de rétention, elle est combinée avec la demande qu'il possède. Quand la variable `TARGET` redescend dans l'arbre, deux valeurs sont envoyées aux fils gauche et droite s'il y a eu à ce niveau une combinaison de message. Par conséquent, le nombre de requêtes effectuées sur le processeur 0 est fortement diminué.

6.2.2 Les méthodes centralisées asynchrones

Dans ces techniques, un serveur est chargé de fournir au processeur qui le souhaite l'adresse d'un site surchargé ou inoccupé afin de les mettre en correspondance. Cette technique est employée dans divers algorithmes. Le serveur a connaissance de l'état du système et contient une liste des différents sites candidats. Dans la technique proposée par [PB90], un processeur désigné comme ordonnateur maintient une file d'attente de type FIFO de donneurs. À chaque fois qu'un processeur devient oisif, il effectue une requête auprès de l'ordonnateur. Ce dernier, effectue un sondage de tous les processeurs contenus dans sa file, jusqu'à ce qu'il trouve un

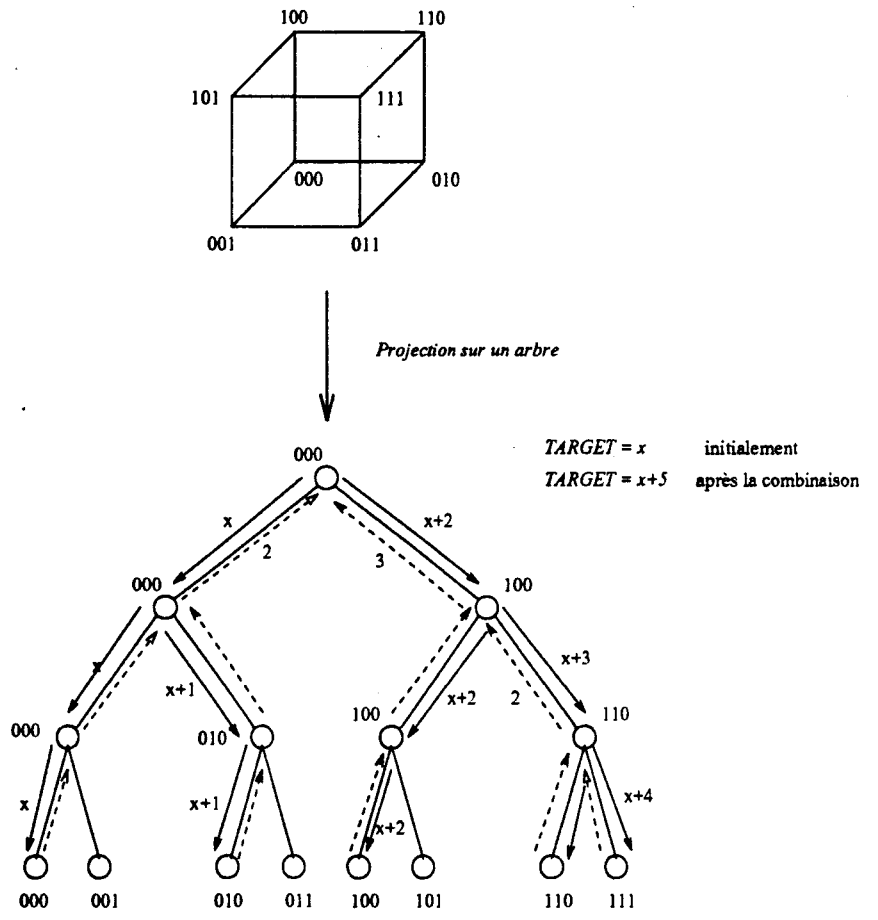


FIG. I.12 - *Présentation d'une incrémentation hiérarchique de la variable* TARGET *dans un hypercube de* 8 *processeurs. La variable* TARGET *est contenue par le processeurs* 000, *sa valeur est* x *à l'étape initiale. La remontée des informations est indiquée par des pointillés, la redescente par des lignes pleines.*

site possédant du travail. Les méthodes centralisées asynchrones présentent une différence importante par rapport aux techniques aveugles : tous les processeurs interrogés sont des partenaires potentiels.

Les méthodes centralisées asynchrones

Des méthodes d'équilibrage centralisées SIMD combinent les avantages d'une gestion centralisée avec le faible coût d'une méthode de localisation aveugle garantissant un appariement idéal. Elles s'inspirent de la classique technique du rendez-vous [Hil88]. Dans une première phase, les processeurs inactifs sont énumérés. Les processeurs contenant une certaine quantité de travail sont ensuite énumérés, et ces deux listes sont appariées comme indiqué sur la figure I.13. La technique d'appariement par rendez-vous est étroitement liée à l'architecture. Les processeurs élémentaires qui bénéficient de l'échange de données sont ceux dont les numéros d'identification sont les plus faibles.

Processeurs	1	2	3	4	5	6	7	8
État	O	O	O	O	O	I	I	O
Rendez-vous classique	1	2	3	4	5			6
Énumération des PEs libres						1	2	

FIG. I.13 - *Présentation de la techniques de rendez-vous. O désigne un processeur occupé, tandis que I désigne son inactivité. Sur l'exemple, le processeur* 1 *sera mis en correspondance avec le PE* 6 *alors que le PE* 2 *sera mis en correspondance avec le PE* 7

Des variantes sont proposées [KK92] afin de forcer la redistribution des données sur l'ensembles des PEs. Par exemple, sur une rangée de processeurs, au lieu de systématiquement utiliser le premier processeur comme base de départ de l'énumération, on peut garder une trace du dernier processeur alloué et ainsi décaler l'index de départ. La figure I.14 montre un exemple de la technique rendez-vous avec décalage pour un système de 8 PEs. Supposons qu'au moment du déclenchement de l'équilibrage, les PEs 6 et 7 soient inoccupés et que le dernier PE alloué dans la phase précédente de rééquilibrage fût le PE 5. La technique de rendez-vous classique (*cf.* figure I.13) apparie les PEs 6 et 7 avec les PEs chargés 1 et 2. La technique de rendez-vous avec décalage fait correspondre les PEs 6 et 7 avec les PE 8 et 1. Le pointeur de décalage avance jusqu'au processeur 1.

Supposons qu'une autre phase d'équilibrage soit appelée. Si les PEs 6 et 7 sont de nouveau oisifs et les autres chargés, la technique de rendez-vous avec décalage permet d'apparier les 2 PEs oisifs avec les PEs 2 et 3.

Cette technique de rendez-vous permet d'éviter une redistribution liée avec l'alignement physiques des processeurs élémentaires.

	Processeurs	1	2	3	4	5	6	7	8
1er équilibrage									
	État	O	O	O	O	O	I	I	O
	Décalage précédent					↑			
	Rendez-vous avec décalage	2	3	4	5	6			1
	Énumération des PEs libres						1	2	
2e équilibrage									
	État	O	O	O	O	O	I	I	O
	Décalage précédent	↑							
	Rendez-vous avec décalage	6	1	2	3	4			5
	Énumération des PEs libres						1	2	

FIG. I.14 - *Présentation de la techniques de rendez-vous avec décalage. O désigne un processeur occupé, tandis que I désigne son inactivité*

6.3 Les stratégies d'appariement distribuées

Les stratégies d'appariement distribuées sont caractérisées par une répartition du choix du site destinataire. Les stratégies distribuées permettent d'éviter l'inconvénient majeur des stratégies centralisatrices ; à savoir la formation d'un goulot d'étranglement lorsque la taille du système devient importante. La recherche d'un site destinataire peut s'effectuer en aveugle sans connaissance de l'état des autres nœuds. Elle peut aussi être fait sous forme de sondages en interrogeant un certain nombre de nœuds pour choisir le plus adapté. Il est possible d'effectuer une recherche exhaustive ou de lancer des enchères.

6.3.1 Les stratégies distribuées aveugles

Comme dans le cas des stratégies centralisées, les algorithmes d'équilibrage aveugles ne se préoccupent pas de l'état du système ou de leurs voisins pour prendre de décisions. Deux types de techniques aveugles peuvent être distinguées :

Systématiques Les stratégies distribuées aveugles systématiques émettent (ou reçoivent) une partie de leur charge vers un certain nombre de nœuds quelque soit leur état. L'algorithme proposé par Vornberger [Vor90] répond à ce principe. Un nœud diffuse régulièrement à ses voisins une partie de sa charge. Les différentes stratégies d'échanges dimensionnelles [XL92] associent cycliquement un processeur à un autre, sans tenir compte de l'état du système.

Petit Carrousel La technique du *Petit Carrousel* est caractérisée par la maintenance d'une variable locale `TARGET` par chaque processeur. Chaque fois qu'une demande de travail

est effectuée auprès du processeur, la valeur de **TARGET** est retournée. La variable **TARGET** est ensuite incrémentée. Cette approche présente l'avantage d'éviter le goulot d'étranglement du *Grand Carrousel*, mais présente le désavantage de permettre la simultanéité de plusieurs requêtes sur le même processeur [KGR91], ce qui provoque des conflits d'accès.

6.3.2 Les stratégies distribuées utilisant le sondage

Une méthode à base de sondage est caractérisée par une interrogation d'un ensemble de nœuds. Parmi les nœuds sondés, le mieux adapté suivant certains critères est sélectionné pour le transfert. C'est une des méthodes les plus utilisées pour effectuer une recherche de partenaire en MIMD. Le nœud recherché qui sera soit receveur soit émetteur peut être sondé de différentes manières :

- soit sur un domaine restreint comme par exemple le voisinage [Zho88, LMR91, LM93] ;
- soit sur le système dans son ensemble en sélectionnant aléatoirement un processeur [ELZ86a, DTJ89].

Une limite au nombre de sondages peut être fixée arbitrairement et ce de manière statique afin de limiter le coût d'une telle politique dans les réseaux de taille importante. Étonnamment, il semble d'après les simulations effectuées par Eager et *al* [ELZ86b], que le comportement moyen de la stratégie soit relativement indépendant de cette limite. Afin de limiter le nombre de sondages, des informations recueillies durant les précédents sondages peuvent aussi être utilisées.

L'algorithme *Asyn* [SU87] utilise également le sondage pour se renseigner sur l'état du réseau et pour choisir son partenaire. Un processeur émet un message contenant C emplacements vides qui seront remplis par la charge des processeurs visités. Le choix du processeur est effectué après le retour de ce message. Dans ce cas, on peut se rendre compte de l'étroite imbrication de la politique d'informations et de la politique de localisation.

6.3.3 Les stratégies décentralisées utilisant la recherche

Les stratégies décentralisées utilisant la recherche sont caractérisées par une prise de décision basée sur des informations contenues par les nœuds. Chaque nœud peut par exemple posséder l'état de la charge de l'ensemble des autres nœuds du système. Le transfert (respectivement réception) de charge sera effectué vers le processeur le moins (respectivement plus) chargé.

Dans l'algorithme distribué *MultiEnvoi* [Sta84], chaque processeur possède la table de charge de l'ensemble des autres processeurs. Si la différence de charge entre le site le moins chargé et la charge locale est supérieure à un certain seuil, plusieurs objets sont transférés vers le nœud le moins chargé. Cet algorithme est amélioré dans la technique *Enregistrement* [Sta84]. Elle permet d'éviter un envoi répété vers le site le moins chargé. Lorsqu'un nœud transfère un ou plusieurs objets vers un site, il enregistre ce fait dans sa table de charge locale. Il s'interdit d'effectuer de nouveaux transferts vers ce même site pendant une durée δ appelée fenêtre.

Comme la place prise en mémoire par une carte du système devient prohibitive dès que le nombre de nœuds devient important, des heuristiques ont été proposées pour réduire le nombre d'informations.

Barak et *al* [BS85] proposent une méthode en quatre étapes afin de mettre à jour les vecteurs de charge d'un système distribué en utilisant un vecteur de charge dont la taille l est limitée. Cet algorithme a été intégré dans le noyau du système d'exploitation distribué MOS :

Pour tous les processeurs :

Étape 1. Mise à jour de la charge locale
Étape 2. Choix d'un entier aléatoire compris entre 1 et n. (n étant le nombre de processeurs)
Étape 3. Envoi de la première moitié du vecteur de charge L_R au processeur i
Lors de la réception d'un vecteur de charge L_R émis par un site distant :
Étape 4. Fusion de cette information avec le vecteur de charge local, de la façon suivante :

$$L(i) \rightarrow L(2i), 1 \leq i \leq l/2 - 1$$

et

$$L_R(i) \rightarrow L(2i+1), 0 \leq i \leq l/2 - 1$$

La taille l du vecteur est bien entendue particulièrement difficile à apprécier. Plus l est grand, plus la mise à jour des informations sera coûteuse en temps. À l'inverse, une faible valeur de l conduit à des mauvaises prises de décision dans la politique d'appariement (manque d'informations).

6.3.4 Les techniques d'enchères

Les techniques d'enchères tentent de transformer le système en une économie miniature, avec des demandeurs de service, des prestataires et des prix fixés selon les lois de l'offre et

de la demande [Tan92]. Les processus tentent d'obtenir du temps CPU pour effectuer leur travail, ce que proposent les processeurs à un certain prix. Cette méthode a été proposée par Smith [Smi80].

À la création d'un processus, une requête d'enchères est diffusée dans le système. Les processeurs intéressés répondent par une offre. Cette offre correspond généralement au coût d'exécution. À la réception des offres, le processus sélectionne le processeur qui offre le coût le plus bas. Les enchères peuvent être émises sur le système tout entier [HCG+82, RSZ89]. La quantité de communications émise est non négligeable et peut saturer le système. Un moyen de restreindre le nombre de communications est d'adapter dynamiquement la distance sur laquelle s'effectue les enchères. Les enchères sont effectuées initialement à une distance d du processus à placer. Si l'initiateur ne reçoit pas assez de propositions pendant un intervalle de temps fixé, la valeur de d est augmentée [LMR91, SS84, CK87].

Les stratégies d'appariement distribuées

Les stratégies d'appariement distribuées sont du fait même de leur décentralisation et de l'autonomie laissée aux processeurs assez éloignées du modèle SIMD.

Les stratégies aveugles décentralisées n'ont pas été mises en œuvre sur les machines SIMD, bien qu'une telle approche soit envisageable.

Les différentes techniques utilisant le sondage pour trouver des partenaires sont véritablement spécifiques à l'asynchronisme décentralisé qui ignore l'état du système et ne présentent pas d'attrait pour une adaptation au modèle synchrone.

Frye et Myczkowski [FM90] ont utilisé une méthode similaire aux stratégies décentralisées utilisant la recherche pour une implémentation SIMD d'un algorithme d'équilibrage. Dans leur algorithme, les processeurs élémentaires chargés interrogent séquentiellement leurs voisins afin de connaître leur état. Chaque voisin inactif recevra une donnée.

Les techniques d'enchères sont également ciblées pour les machines MIMD. Elles ne présentent pas de contrôle centralisée et chaque composant évalue les informations reçues suivant son propre jugement. Les communications sont effectuées de manière asynchrone et bidirectionnelle. Pour ces raisons, une approche SIMD des techniques d'enchères ne présente guère d'intérêt.

6.4 Les stratégies d'appariement semi-distribuées

Les stratégies d'appariement semi-distribuées offrent un compromis entre les approches centralisées et distribuées. Au lieu de désigner un nœud comme responsable de la recherche de partenaire ou de répartir la décision sur l'ensemble des nœuds, on découpe le système en plusieurs petits domaines. L'équilibrage est alors réalisé dans chacun de ces domaines. Deux approches semi-distribuées existent :

L'approche hiérarchique Le système est découpé en une hiérarchie de sous-systèmes. Des nœuds sont désignés comme responsable des différents niveaux de la hiérarchie. La structure hiérarchique la plus souvent retenue est un arbre binaire [WLR93]. L'équilibrage est réalisé en remontant l'arbre et en équilibrant la charge des domaines adjacents à chaque niveau de la hiérarchie.

L'approche indépendante L'approche indépendante propose le partitionnement du système en sous-domaines indépendants. Un nœud est désigné comme responsable de chaque sous-domaine et effectue un équilibrage à l'intérieur de chaque sous-domaine. En cas de besoin, un processus peut être migré d'un sous-domaine vers un autre. Ahmad et *al* [AG90] proposent le découpage d'un hypercube en « sphères ». Lorsqu'un processus est créé, il est attribué en priorité à un nœud appartenant à sa « sphère ». Si l'allocation n'est pas possible, une autre « sphère » est sélectionnée.

Les stratégies d'appariement semi-distribuées

Les techniques d'appariement semi-distribuées peuvent s'adapter sur les machines synchrones comme nous le proposons au chapitre III. Un équilibrage est réalisé sur des petits domaines à l'aide de communications de voisinage. Les domaines sont ensuite équilibrés par l'intermédiaire de communications globales aux systèmes.

7 Conclusion

La majorité des travaux concernant l'équilibrage dynamique est spécifique aux machines asynchrones à mémoire distribuée. Nous avons montré que l'équilibrage dynamique sur une machine obéissant à un modèle d'exécution à parallélisme de données présente lui aussi d'intéressantes perspectives de développement. Le fil conducteur que nous avons suivi tout au long de ce chapitre est une présentation parallèle des approches MIMD et SIMD. Nous avons présenté les techniques qui sont identiques aux deux approches, les méthodes qu'il paraît utiles d'adapter sur une architecture synchrone et les stratégies qu'il semble plus délicat de

vouloir transposer (par exemple la migration), ou même inutiles d'adapter (par exemple les techniques d'enchères). Nous avons montré que les approches MIMD et SIMD ne sont pas opposées mais plutôt complémentaires car elles s'adressent à des problèmes différents. Nous avons présenté une classification de cinq critères, et nous estimons que seules quatre étapes seront nécessaires pour réaliser une description d'une technique SIMD.

1. La politique d'informations ne présente pas une correspondance directe dans le modèle SIMD. Elle est souvent définie de manière implicite pour les machines synchrones. C'est cette politique qui présente les différences les plus importantes entre les systèmes MIMD et SIMD.

2. La politique de déclenchement permet de provoquer une phase d'équilibrage. Elle est généralement très simple sur une machine MIMD : les processeurs prenant en charge eux-mêmes la décision d'entreprendre un équilibrage de charge. Une politique de déclenchement provoque sur une machine SIMD, l'arrêt des calculs dans tous les processeurs élémentaires. Elle doit pouvoir apprécier rapidement et de façon la plus précise possible s'il est opportun d'effectuer un équilibrage. Elle constitue une phase primordiale d'une technique d'équilibrage sur une machine synchrone.

3. La politique de sélection désigne les processeurs qui participent à l'équilibrage. Les stratégies les plus employées sur les machines MIMD sont celles utilisant une politique de seuil ou de comparaison. Les techniques systématiques s'adaptent avec succès sur les machines synchrones.

4. La politique de désignation locale sélectionne les entités pour le transfert. La méthode la plus usitée est l'indifférenciation des processus. La politique de désignation locale est alors réduite à une simple file d'attente. Les processus sont exécutés par le nœud suivant une politique de type « Premier Arrivé-Premier Servi ». Certaines applications, comme les systèmes d'exploitation intégrant la migration de tâches doivent sélectionner avec plus de rigueur les éventuels candidats au transfert. Des techniques de filtrage peuvent être appliquées. Dans le modèle synchrone, la technique d'indifférenciation des données est la plus utilisée. Des méthodes plus élaborées de désignation locale ont également été proposées pour les architectures synchrones. Par exemple, la recherche en parallèle dans un arbre nécessite une sélection rigoureuse du nœud à déplacer.

5. La politique d'appariement réalise les transferts effectifs des entités. De nombreuses politiques d'appariement MIMD ont été proposées. Les stratégies centralisées offrent un équilibrage de bonne qualité au prix de conflits d'accès sur le processeur détenteur des informations. Les techniques distribuées sont adaptées au système comportant un grand nombre de processeurs, mais réalisent difficilement un équilibrage de qualité sur la totalité du système. Les approches semi-distribuées sont en général plus ciblées pour des architectures particulières comme les structures en arbre.

La politique d'appariement trouve une correspondance immédiate dans le modèle SIMD. Les politiques d'appariement SIMD seront de type Rendez-vous ou distribuées-systématiques. Comme les communications locales sont d'une grande importance, nous avons introduit une politique supplémentaire appelée politique de *communications* composée de deux classes : (i) les stratégies de transfert local qui limitent les échanges de données à un voisinage restreint et (ii) les stratégies globales qui autorisent les transferts de travail dans l'ensemble du système.

Chapitre II

Méthodes d'analyse mathématique

Le mesure de la qualité des différents algorithmes étudiés dans le chapitre I est généralement effectuée par de nombreuses simulations. L'algorithme est étudié avec des charges moyennes plus ou moins importantes, avec un nombre de processus plus ou moins grand, avec une vitesse d'exécution des unités centrales plus ou moins rapide... L'impact de surcharges soudaines de travail provoquées sur certains processeurs, la panne de certains nœuds, est simulée pour examiner et quantifier les réactions de l'algorithme.

Les programmeurs utilisent des systèmes multi-processeurs présentant des différences importantes : nombre de processeurs, vitesse d'exécution des processeurs, homogénéité ou hétérogénéité des nœuds, mémoire disponible, topologie, vitesse de transmission dans le réseau... Dans de telles conditions, la comparaison des stratégies d'équilibrage dynamique est difficilement réalisable par des méthodes basées sur la simulation. L'analyse de la complexité et du nombre d'itérations nécessaire à l'obtention d'un état stable d'un algorithme d'équilibrage dynamique est rarement pris en compte mais permettrait d'obtenir une comparaison plus fiable et plus efficace des différentes stratégies.

Il semble cependant difficile d'analyser le comportement d'une technique d'équilibrage de charges, puisque du fait même de leur dynamicité et de leur adaptation permanente à un environnement qui se modifie de manière imprévisible, elles échappent aux méthodes classiques d'analyse de complexité. Des techniques d'analyse ne sont apparues qu'au début des années 90 pour évaluer le comportement *asymptotique* des algorithmes. Ces différentes méthodes permettent d'appréhender des notions importantes qui n'étaient pas définies auparavant comme la convergence d'un algorithme d'équilibrage dynamique ou encore son éventuelle stabilité.

Les différentes approches que nous allons présenter ne s'intéressent pas toutes aux mêmes aspects algorithmiques, certaines supposent la création et l'arrivée permanente de nouveaux processus tandis que d'autres travaillent sur des systèmes déséquilibrés pour lesquels le nombre

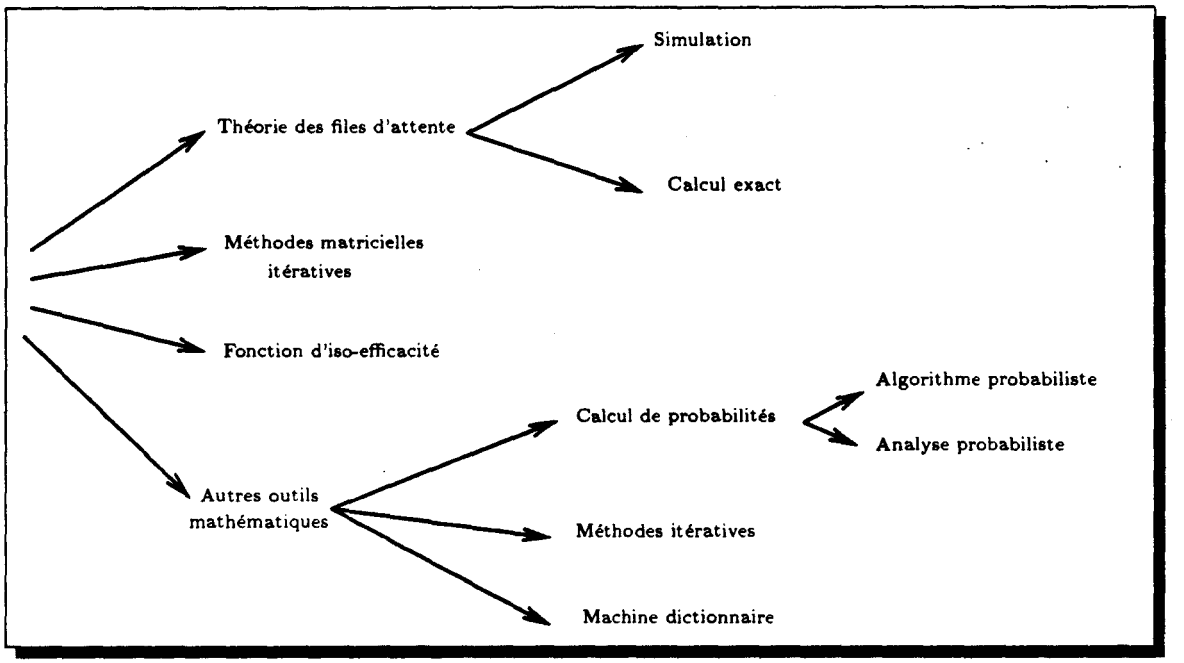


FIG. II.1 - *Présentation des différentes méthodes mathématiques utilisées pour l'analyse des algorithmes*

de tâches est fixe. Certaines approches ont particulièrement retenu notre attention et nous avons analysé le comportement de deux de nos stratégies en s'appuyant sur de telles méthodes.

La première approche examinée (*cf.* figure II.1), la théorie des files d'attente, s'inspire de manière directe des méthodes de simulation existantes en les prolongeant et en les paramétrant afin de permettre une analyse. Elle permet aussi bien de réaliser une analyse fine que grossière en modifiant le nombre et le réalisme des divers paramètres. La méthode matricielle s'éloigne nettement des méthodes de simulation car elle ne prend pas en compte certaines données comme la durée d'exécution de l'équilibrage. Elle est couramment utilisée dans l'analyse des méthodes par diffusion et des techniques d'équilibrage par échange dimensionnel. La fonction d'iso-efficacité introduite dans le paragraphe 3 permet l'analyse de l'extensibilité d'un grand nombre de stratégies. En fin, dans la section 4 sont regroupées diverses méthodes d'analyse moins génériques que les précédentes et qui s'attachent plutôt à l'analyse de techniques d'équilibrages uniques ou liées à un problème. Une discussion sur les mérites et inconvénients des diverses méthodologies est présentée au paragraphe 5.

1 Analyse de système de rééquilibrage à l'aide de la théorie des files d'attente

1.1 La théorie des files d'attente

La *théorie des files d'attente* est chronologiquement le premier outil utilisé pour analyser les algorithmes d'équilibrage dynamique. Elle permet une représentation relativement souple du système et l'étude de son comportement en faisant varier quelques paramètres.

Dans le cadre d'une modélisation d'un système multi-processeurs par la théorie des files d'attente, chaque processeur est considéré comme un serveur offrant aux différents processeurs arrivant appelés clients, un service. Le service proposé est l'exécution d'une tâche.

L'analyse des techniques d'équilibrage dynamique par la théorie des files d'attente suppose la création d'un modèle représentant le système multi-processeurs et ses interactions avec l'algorithme d'équilibrage. Deux approches sont possibles suivant l'utilisation des équations obtenues :

1. Des simulations sont réalisées tout en faisant varier un certain nombre de paramètres afin d'obtenir des solutions approchées.
2. Les équations sont résolues analytiquement. Elles permettent d'étudier directement l'influence des paramètres sur le temps de réponse moyen du système.

Une étude théorique complète peut être trouvée dans [Kle76]. La théorie des files d'attente permet de représenter aisément les processus circulant sur un système multi-processeurs. Les processus circulant dans le système sont appelés « clients », la phase d'attente des processus pour le traitement sur un processeur étant modélisée par une file d'attente. L'exécution des processus sur les processeurs est nommée « service ». Cette modélisation relativement souple est caractérisée par les paramètres suivants :

- le nombre de stations de services et leurs dispositions ;
- la méthode de choix des unités dans la file d'attente (FIFO, aléatoire...) ;
- le processus de distribution des arrivées (processus d'arrivée poissonnien, Gaussien...) ;
- le processus de distribution des durées de service des clients.

Dans le cadre de l'analyse des méthodes de redistribution, chaque nœud du système sera considéré comme un centre de file d'attente.

Des informations plus détaillées sur la théorie des files d'attente peuvent être trouvées dans l'article de Allen [All80] et le livre de Deitel [Dei84]. La sélection des unités dans les files d'attente obéit en général à la politique « Premier arrivé-Premier servi » « FIFO », qui est la plus représentative de l'attente d'une tâche à un nœud. D'autres politiques sont parfois utilisées, comme la discipline « BIFO » (acronyme de Biggest In First Out) où la première entité choisie à la sortie de la file est la plus importante en demande de temps d'exécution.

Dans tous les cas de figures, on considère que les clients arrivent à des temps $t_0 < t_1 < t_2 < \ldots < t_n$, cela signifie qu'il ne peut y avoir qu'une arrivée à un instant donné, et qu'il n'existe pas de collisions entre les clients. Les variables aléatoires $\tau_k = t_k - t_{k-1}$, $(k \geq 1)$ mesurent le temps entre deux arrivées successives et sont appelées *temps d'inter-arrivées.* On suppose que la séquence des τ_k est identiquement distribuée dans le temps. Le modèle le plus utilisé pour l'analyse des algorithmes d'équilibrage est un système obéissant à une loi d'arrivée poissonnienne. Dans un tel cas, la probabilité que n tâches ou processus arrivent au nœud pendant une période t est de $e^{-\lambda t}(\lambda t)^n/n!$, où λ est le taux d'arrivée moyen par unité de temps. Nous verrons par la suite que les analyses sont souvent conduites en faisant varier le paramètre λ de 0.01 arrivée par unité de temps à 1 arrivée par unité de temps.

De façon identique à l'arrivée, le service est défini par une loi poissonnienne. S est la durée moyenne d'un service (c'est-à-dire le coût moyen d'exécution d'une tâche). Le symbole μ est utilisé pour définir le taux de service moyen par seconde.

Une notation standardisée dite *notation de Kendall* permet de décrire la majorité des systèmes de file d'attentes. Elle se présente sous la forme $A/B/c$. Cette notation suppose que :

- la longueur de la file d'attente n'est pas bornée ;
- la source ou le générateur de processus est infini ;
- la méthode de sélection des clients à l'intérieur de la file est du type FIFO.

Le premier paramètre A indique la distribution des arrivées, le second la distribution temporelle des services et le dernier c le nombre de serveurs. Dans les différentes techniques que nous allons examiner par la suite, A et B prendront la valeur M, ce qui signifie que les lois auxquelles obéissent les services et les arrivées sont du type poissonnien.

En utilisant cette méthode analytique, un système multi-processeurs dans lequel aucune redistribution de processus n'a lieu est aisément définissable. De même, on peut construire un système dans lequel un équilibrage est réalisé pour un coût nul. Ces deux grandeurs estimées, forment une borne supérieure et inférieure à laquelle les algorithmes proposés peuvent être comparés.

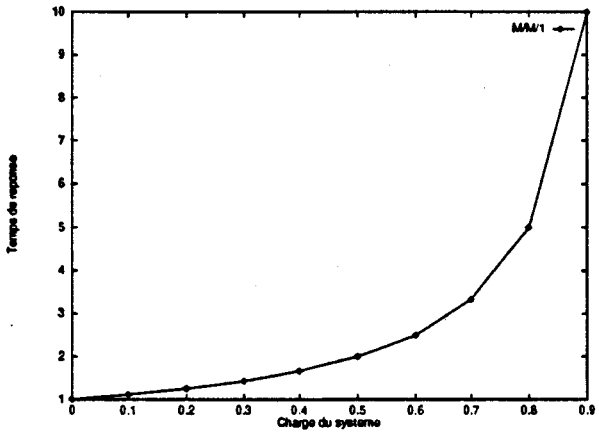


FIG. II.2 - *Temps de réponse en fonction de la charge pour un nœud $M/M/1$*

Pour un ensemble de N processeurs dont l'arrivée et le service sont du type poissonnien, chaque nœud forme une file d'attente $M/M/1$ si aucune charge n'est partagée, le système se comporte donc comme N files de type $M/M/1$.

De manière inverse, un équilibrage à coût nul se représente simplement comme un système $M/M/N$ où tout nouveau processus arrivé ou créé peut utiliser une des N files d'attente. La figure II.2 représente pour une file d'attente de type $M/M/1$ (1 signifiant la présence d'un serveur) le temps de réponse en fonction de la charge $\rho = \lambda/\mu$, (où $\mu = 1$ s, taux moyen de service). Le temps de réponse correspond à la somme de l'attente et de l'exécution des processus. Ainsi, pour un système dont la charge est de $0,9$ (en moyenne $0,9$ processus nouveaux arrivent sur le nœud durant une période de 1s), le temps moyen de réponse pour un système mono-processeur est de 10 s!

De façon similaire, la figure II.3 représente un système de 4 processeurs où le coût de transfert serait nul (système de type $M/M/4$). Ainsi, un processus nouvellement arrivé pourrait être automatiquement re-dirigé vers un des quatre nœuds. Dans ce cas, un système dont la charge ρ est de $0,9$ a un temps de réponse moyen de $2,9$ s. Ce résultat en lui-même montre, s'il en est besoin, le gain qu'il est possible d'obtenir grâce à la redistribution dynamique des processus. Cette courbe sera donc utilisée de manière quasi systématique comme une indication des performances maximales d'un système.

1.2 Méthodes basées sur la simulation

Dans les méthodes basées sur la simulation, un modèle analytique simple est défini ; il permet d'examiner les propriétés du système multi-processeurs couplé aux techniques d'équilibrage ; certains paramètres importants comme la charge du système ρ sont modifiés afin d'estimer les répercussions sur le système. Historiquement, le langage de programmation SIM-

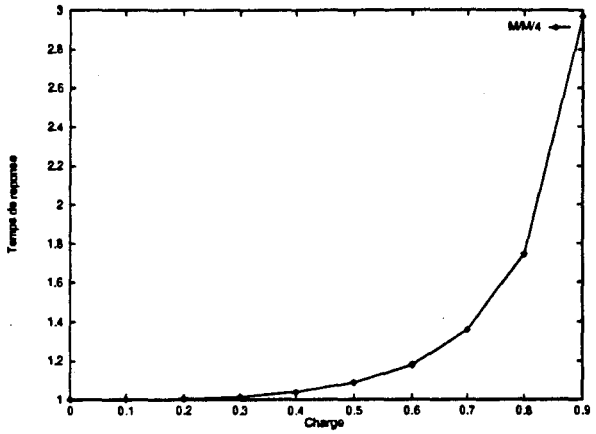


FIG. II.3 - *Temps de réponse en fonction de la charge pour un ensemble de 4 nœuds en supposant un équilibrage à coût nul*

SCRIPT a régulièrement été utilisé à des fins de simulation [KVM68, Mar69].

Afin d'illustrer la méthode basée sur la simulation, nous étudions deux exemples d'algorithmes d'équilibrage :

- l'algorithme d'équilibrage de charges *PID* ;
- l'algorithme de partage de charges *Glouton*.

Étude de l'algorithme PID

Livny et Melman [LM82] ont examiné le comportement asymptotique de trois techniques de distribution des charges et ont étudié leurs performances en les comparant aux courbes classiques $M/M/1$ et $M/M/N$. Nous présentons brièvement selon la classification définie au chapitre I l'algorithme *PID - Poll when IDle algorithm*. C'est un cas typique d'algorithmes dont l'initiative est laissée au receveur.

Politique de mise à jour des informations Les informations sont demandées de manière asynchrone par les processeurs lorsqu'ils deviennent oisifs. C'est donc une politique à la demande.

Politique de déclenchement La politique de déclenchement repose sur le calcul de l'opérateur UBF (UnBalance Factor) présenté dans la section 3 du chapitre I. Ce facteur permet d'estimer l'opportunité de déclencher un équilibrage.

Politique de sélection Un nœud est apte à participer au transfert de processus s'il se retrouve à l'état oisif.

Politique de désignation locale Pour la majorité des techniques dont l'analyse est basée sur la théorie des files d'attente, la tâche sélectionnée pour le transfert effectif obéit à la politique « Premier arrivé-Premier servi », ce qui est le cas pour la technique *PID*.

Politique de d'appariement La politique d'appariement est basée sur une technique de sondages. Le processeur interroge un sous-ensemble de R nœuds du système. R étant un paramètre de l'algorithme. Les transferts s'effectuent sur la totalité du système.

L'algorithme est décrit par le langage SIMSCRIPT II.5. Aucun délai sur le contrôle des opérations n'est introduit, les seuls délais considérés sont ceux dus aux communications. De nombreux paramètres sont ainsi modélisés comme la bande passante du système, le temps de transmission d'un message, le temps de service... Les simulations effectuées sur cet algorithme permettent de s'apercevoir qu'il existe un nombre de serveurs optimal au-delà duquel les performances globales du système se déprécient fortement. L'introduction d'un nouveau serveur entraînant automatiquement un accroissement des messages dans le système. Le facteur d'équilibre β est également étudié. Il correspond au ratio entre le temps moyen d'exécution d'une tâche et le temps moyen nécessaire au transfert d'une tâche. Les simulations indiquent que le facteur β a une influence très importante sur les performances du système.

Étude de l'algorithme Glouton

Le travail de Chowdhury [Cho90] illustre bien la précision avec laquelle peut être décrite un algorithme de partage de charges en utilisant la simulation. L'algorithme proposé appelé *Glouton* fonctionne de la façon suivante :

Politique de mise à jour des informations Les informations sont échangées à la demande des processeurs.

Politique de déclenchement Le déclenchement est asynchrone et commandé par les processeurs.

Politique de sélection La politique de sélection du programme *Glouton* est très simple. Un processeur se trouve impliqué dans l'équilibrage dès que sa charge est non nulle (i.e. dès qu'il contient au moins un processus).

Politique de désignation locale Comme pour l'algorithme *PID*, le transfert de tâches obéit au principe « Premier arrivé-Premier servi ».

Politique d'appariement Le poids d'un nœud est défini par le nombre de processus présents dans la file d'attente. Si une tâche arrive ou est créée sur un nœud dont le poids est n, $n > 0$, elle est par définition de la politique de sélection automatiquement candidate

au transfert. La politique d'appariement fonctionne par sondages et cherche pour un tel nœud, un partenaire se trouvant à l'état $f(n)$. Les performances de l'algorithme sont bien sur dépendantes de la fonction $f(n)$. Il est préférable d'utiliser une fonction f telle que $f(n) < n$. Le choix d'une fonction f telle que $f(n) < n$ implique que les processus seront transférés vers un nœud destinataire dont la charge est plus faible que le nœud source. La fonction choisie est $f(n) = n$ div 3 qui offre les meilleurs résultats.

Un problème apparaissant lors de la définition du modèle analytique de l'algorithme est l'estimation du sur-coût induit par le transfert d'une tâche. Alors que [LM82] se limitait à un sur-coût fixe, Chowdhury calcule le temps de réponse d'une tâche transférée comme une somme d'overheads :

$$R = \alpha P + C + N + Y + X$$

où α est le nombre de sondages effectué afin de trouver une destination, P le temps pour une telle interrogation, C le temps nécessaire au nœud source à la préparation de l'envoi du processus, N le temps total du transfert et de l'envoi d'un message de confirmation de la destination vers la source, Y l'attente de la tâche transférée au processeur destinataire et X le « classique » temps de service ou durée d'exécution.

Les différentes simulations menées ont permis de s'apercevoir que la technique *Glouton* a un comportement moyen plus performant que les algorithmes à seuil [ELZ86a]. Cette méthode analytique montre qu'il existe une charge critique du système ρ_c au-dessus de laquelle, le temps de réponse moyen devient pire que le temps de réponse d'un système sans rééquilibrage, c'est-à-dire une file du type $M/M/1$.

1.3 Les solutions exactes

Il est souvent possible de décrire un système comme un ensemble d'états *discrets* et *exclusifs*. Le passage d'un état vers un autre est caractérisé par une transition possédant une certaine probabilité. Cette représentation sous forme de chaîne est appelée chaîne de Markov. Chaque nœud d'un système multi-processeurs est représenté par une chaîne de Markov, les états représentant la charge du nœud.

Obtenir une solution exacte d'un modèle Markovien dans le cadre qui nous intéresse est une chose complexe car l'espace d'états des systèmes est immense. Si l'on considère un nœud composé d'une file d'attente de 20 places, un système avec 20 états différents peut être construit. Un système constitué de 10 processeurs aurait un nombre d'états égal à 20^{10} !!

Afin de simplifier l'analyse, on suppose que l'état d'un nœud est indépendant des états de tous les autres. Chaque nœud peut donc être étudié isolément. Les actions du reste du

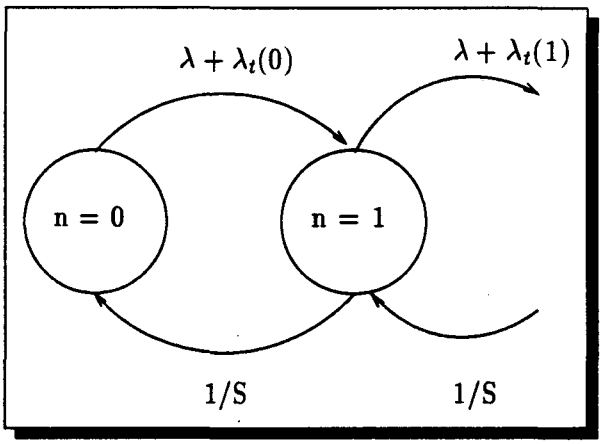


FIG. II.4 - *Représentation des états d'un nœud*

système sur le nœud sont représentées par une arrivée de tâches transférées, le taux d'arrivée étant évidemment différent suivant la technique utilisée.

La figure II.4 explicite la représentation du modèle client/serveur. À chaque état, le taux d'arrivée des tâches est égal à la somme des taux des nouvelles tâches (λ) et des tâches transférées à ce nœud par le reste du système ($\lambda_t(n)$). Ce dernier terme étant dépendant de la longueur de la file d'attente n.

Dans [ELZ86a], trois politiques de partage de charges sont comparées en utilisant la méthode analytique. Elles possèdent un certain nombre de points communs et ne diffèrent que sur leur politique d'appariement. Leurs caractéristiques communes sont:

Politique d'informations Les échanges d'informations sont réalisés à la demande de nœuds.

Politique de déclenchement Les processeurs décident eux-mêmes suivant leur état de l'opportunité d'un rééquilibrage. La politique de déclenchement est autonome.

Politique de sélection Une tâche est acceptée par un nœud si le nombre de processus, en attente ou en train d'être exécuté, est inférieur à seuil statique T. Si la charge du nœud cible est supérieure au seuil, la tâche est transférée vers un autre nœud. La politique de sélection est donc une politique de seuils.

Politique de d'appariement Cette dernière diffère suivant la technique choisie.

Les politiques d'appariement examinées sont des politiques qui obéissent à des lois relati-

vement simples :

Aléatoire Avec cette politique, un nœud destinataire est sélectionné au hasard pour le transfert de la tâche. La tâche est alors envoyée à ce nœud. Si le processeur destinataire possède une charge inférieure à un seuil fixé, le processus est accepté. Dans le cas contraire, le processus est à nouveau transféré vers un autre site. Une politique aléatoire a malheureusement tendance à provoquer l'instabilité. Après un moment, l'ensemble des nœuds passent tout leur temps à tenter de transférer des objets sans exécuter aucun processus. Afin d'éviter ce phénomène un nombre limite de transfert L_t est imposé. Le processus destinataire du $L_t$ième transfert exécute la tâche **quelque soit** son état.

Seuil Dans cette stratégie, un nœud est sondé aléatoirement. Si sa charge est inférieure au seuil T fixé au préalable, la tâche est transférée. Au plus, L_p sondages sont effectués, avant que la tâche soit imposée à un nœud.

Le Plus Court Dans cette politique, L_p sondages au plus sont effectués, le nœud sélectionné pour le transfert est celui qui possède la file d'attente la plus petite parmi les L_p nœuds interrogés.

Dans tous les cas proposés une politique de transfert à base de sondages a été utilisée pour rechercher un site destinataire au lieu d'une technique se servant de la diffusion. Les sondages permettent de limiter le coût des communications dans les réseaux de dimension importante.

Les analyses effectuées permettent de démontrer qu'une technique de partage de charges basée sur un système de seuil améliore de façons particulièrement sensible les performances du système. Les auteurs expriment leur étonnement devant le mauvais comportement de l'algorithme *Le Plus Court*. En effet, l'algorithme sondant un certain nombre de nœuds en vue de la connaissance du serveur le moins chargé n'apporte que des performances relativement modestes au système et a un comportement qui n'est pas meilleur que la politique à base de seuil.

La méthode analytique [ELZ86b] a en outre permis de comparer les politiques où le receveur prend l'initiative avec celles où l'envoyeur prend l'initiative. L'analyse démontre que pour un système dont la charge moyennée sur tous les processeurs reste soit faible soit modérée, un algorithme dont l'initiative est prise par les nœuds chargés semble plus adéquate, tandis que pour un système relativement chargé, les algorithmes où l'initiative est laissée aux processeurs sous-chargés offrent de meilleurs résultats. Ces résultats théoriques sont à modérer avec les résultats expérimentaux obtenues par [WLR93] qui dénotent un avantage aux stratégies passives.

1.4 Conclusions

La théorie des files d'attente permet de construire un modèle représentant un système multi-processeurs utilisant une technique de rééquilibrage dynamique. Ce modèle peut être soit utilisé en vue de simulation soit résolu par des méthodes analytiques.

Analyse des systèmes de rééquilibrage à l'aide de la théorie des files d'attente

La méthode de la théorie des files d'attente peut s'appliquer sur une machine synchrone. Cependant, les techniques examinées posent le principe d'un taux de service (temps moyen d'exécution d'un processus) et d'un taux d'arrivée (temps moyen entre l'arrivée ou la création de deux processus) observant une certaine régularité. Cette régularité obéit en règle générale à une loi statistique comme la loi de Poisson. Or, dans les cas que nous examinons dans cette thèse, le comportement des objets est totalement imprévisible ; certaines données pouvant s'éteindre après quelques « itérations », d'autres au contraire pouvant adopter un comportement explosif. L'approche par la théorie des files d'attente nous semble donc incompatible avec la problématique que nous avons étudiée sur les machines SIMD , où la vie et la mort des données n'obéissent pas à un comportement moyen.

2 Analyse des systèmes de rééquilibrage à l'aide des méthodes matricielles itératives

L'utilisation des méthodes matricielles afin d'estimer les performances relatives d'un algorithme découle de l'utilisation des graphes dans la représentation de systèmes multi-processeurs. En effet, dans de nombreux articles traitant de l'allocation statique [ST85], l'architecture cible est représentée par un graphe connexe, non orienté $G_m = (V_m, E_m)$ où V_m l'ensemble des sommets désigne les processeurs, et E_m l'ensemble des arêtes, représente les liens physiques [MT91]. Un graphe peut facilement être représenté sous forme matricielle en numérotant les nœuds du graphe, une entrée non nulle de la matrice indiquant un lien physique entre les deux processeurs concernés. Ainsi, un anneau de 3 processeurs peut être représenté par la matrice de la figure II.5.

Cette approche a pour la première fois été définie par Cybenko [Cyb87] en 1987 et a par la suite conduit à de nombreux travaux. L'idée de base est de suivre temporellement l'évolution de la charge de tous les processeurs. Deux techniques classiques de l'équilibrage dynamique ont ainsi été analysées, la méthode par diffusion et l'échange dimensionnel. La méthode par diffusion autorise les transferts de travail *asynchrones* entre les processeurs tandis que l'échange dimensionnel est basé sur des transferts de charge *synchrones*.

$$M = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

FIG. II.5 - *Représentation sous forme matricielle d'un anneau de 3 processeurs*

2.1 Analyse de la méthode par diffusion par la méthode itérative matricielle

Le principe de la méthode de diffusion est simple : à des instants t, les processeurs i et j comparent leur charge locale et échangent une partie de leur travail. La méthode de diffusion est asynchrone, certains processeurs pouvant être dans une phase de calcul tandis que d'autres sont dans une phase d'équilibrage.

Pour un système comprenant n processeurs, la distribution des charges à l'instant t sur l'ensemble multi-processeurs est définie comme un vecteur de dimension n, $W^{(t)}$ où les $w_i^{(t)}$ correspondent au nombre de tâches possédées par le processeur i à l'instant t.

Le modèle d'équilibrage dynamique peut être décrit par l'expression suivante :

$$w_i^{(t+1)} = w_i^{(t)} + \sum_j \alpha_{ij}(w_j^{(t)} - w_i^{(t)}) \tag{II.1}$$

dans laquelle α_{ij} est une constante non négative indiquant les échanges de travail réalisés par tous les processeurs j ayant un lien avec le processeur i. La sommation est effectuée sur tous les processeurs j avec la convention , $\alpha_{ij} = 0$ si les processeurs i et j ne sont pas reliés.

L'équation II.1 peut également être réécrite de la manière suivante :

$$w_i^{(t+1)} = (1 - \sum_j \alpha_{ij})w_i^{(t)} + \sum_j \alpha_{ij} w_j^{(t)} \tag{II.2}$$

La formule précédente peut s'écrire simplement sous la forme d'une équation matricielle :

$$W^{(t+1)} = MW^{(t)} \tag{II.3}$$

La matrice M appelée *matrice de diffusion* est symétrique puisque les échanges sont bi-

directionnels. Elle possède un certain nombre de propriétés. Cybenko [Cyb89] montre que l'équation II.3 converge en général vers la matrice dont toutes les entrées sont égales à $1/n$. La matrice dont toutes les entrées sont égales à $1/n$ représente le système multi-processeurs dans un état stable (tous les processeurs possèdent la même charge).

Afin détudier la condition de convergence de la méthode par diffusion, il est nécessaire d'introduire les notions de graphe induit et de graphe biparti.

Définition 1 *Étant donné un graphe N et une matrice de diffusion M, le graphe induit est donné par $N_m = (V, E_m)$, où pour tout $i, j \in V$, nous avons $(i, j) \in E_m$ ssi $\alpha_{ij} > 0$.*

Le graphe induit est dans l'essentiel identique au graphe $G_m = (V_m, E_m)$ représentant le système à l'exception de quelques arêtes manquantes. Les arêtes supprimées sont celles pour lesquelles il n'y a pas d'échange de travail, c'est-à-dire les arêtes telles que $\alpha_{i,j} = 0$.

Définition 2 *Un graphe est dit biparti [Sed88] si l'on peut diviser les nœuds en deux ensembles distincts dans lesquels il n'existe aucune arête reliant deux nœuds du même ensemble.*

Théorème 1 *L'équation II.3 converge vers la distribution uniforme ssi le graphe induit est connecté et soit l'une des deux conditions suivantes est vérifiée:*

(i) $(1 - \sum_i(\alpha_{ij})) > 0$ *pour au moins un* j
(ii) le graphe induit n'est pas biparti

La méthode matricielle itérative a été utilisée pour étudier la convergence de la méthode de diffusion sur un hypercube de dimension quelconque.

2.2 Analyse de la méthode d'échange dimensionnel par la méthode itérative matricielle

La méthode d'échange dimensionnel s'inspire de la technique par diffusion vue dans le paragraphe précédent mais impose une contrainte de synchronisme dans les échanges. La redistribution des tâches a lieu simultanément sur l'ensemble des processeurs.

De façon semblable à la méthode de diffusion, la technique d'équilibrage *d'Échange Dimensionnel* (*Dimension Exchange* [Cyb89, XL92]) a été analysée. À la différence de la méthode

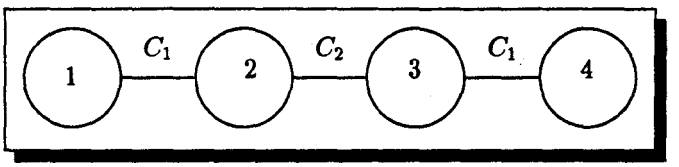


FIG. II.6 - *Exemple de graphe coloré*

par diffusion, le système n'est plus représenté par un graphe non orienté mais par un graphe **coloré** $G_k = (V, E_k)$ de k couleurs où V représente toujours les processeurs, mais E_k est un ensemble de triplets de la forme $(i, j; c)$, $(i, j; c) \in E_k$ ssi c est la couleur de l'arête (i, j). Le graphe est coloré avec un nombre minimum de couleurs de telle sorte que deux arêtes ayant un lien sur un même nœud ne possèdent pas la même couleur. La figure II.6 montre la représentation sous forme de graphe d'une chaîne de 4 processeurs. Deux couleurs sont nécessaires à sa construction.

```
for all p in parallel do
   while (not Terminate) do
      for  c := 1 to k do
         if  (il existe une arrête de couleur c)  then
            Charge_Voisin := Echange(c)
            w[p] := (1 − λ) × w[p] + λ × Charge_Voisin
         fi
      od
   od
od
```

FIG. II.7 - *Pseudo-code de l'algorithme d'Échange Dimensionnel*

L'algorithme du *Dimension Exchange* fonctionne sur chaque processeur de manière décentralisée. Soit k le nombre de couleurs du graphe ; pendant k itérations chaque processeur échangera une partie de sa charge avec ses voisins et suivant l'arête de couleur i à l'étape i (voir figure II.7). Pour un système représentant le graphe coloré G_k, w indique la charge des processeurs et λ indique la valeur du paramètre d'échange. L'opérateur parallèle $Echange(c)$ permet à tout nœud de recevoir la charge de son voisin relié par une arête de couleur c.

Cet algorithme peut être formalisé comme la méthode par diffusion, la distribution des charges est mise sous la forme d'un vecteur $W^{(t)}$ de dimension n (nombre de processeurs).

Si la quantité de travail échangée est égale à la moitié de la charge initiale, cette technique est nommée *Échange Dimensionnel* (« *Dimension Exchange* ») [Cyb89], si ce n'est pas exactement la moitié on la nomme *Échange Dimensionnel Global* (« *Global Dimension Exchange* » ou *GDE*) [XL92, HLM+90, XL94a, XL94b].

Les échanges à chaque itération peuvent être mis sous forme d'équation. La charge du processeur i peut être modélisée à l'étape t avec $c = (t \bmod k) + 1$:

$$\begin{cases} w_i^{(t+1)} = (1-\lambda)w_i^{(t)} + \lambda w_j^{(t)} & \text{si } \exists (i,j;c) \in E_k \\ w_i^{(t+1)} = w_i^{(t)} & \text{sinon} \end{cases} \tag{II.4}$$

Soit λ le paramètre d'échange, la distribution de charge de l'équation II.4 peut être mise sous forme matricielle et calculée matriciellement :

$$W^{t+k} = M(\lambda)W^t \tag{II.5}$$

où $M(\lambda) = M_k(\lambda) \times M_{k-1}(\lambda) \times \ldots \times M_1(\lambda)$.

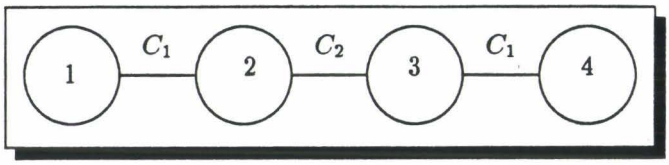
Par exemple, les deux matrices représentant l'évolution de la chaîne de processeurs de la figure II.6 sont représentées dans la figure II.8. Les matrices $M_1(\lambda)$ et $M_2(\lambda)$ indiquant respectivement l'activité des liens de couleur n° 1 et n° 2. Cette matrice appelée matrice GDE, converge vers une distribution uniforme selon les critères du théorème suivant :

Théorème 2 *Soit $M(\lambda)$ une matrice GDE d'un graphe coloré de k couleurs G_k, $\bar{M}$ une matrice d'ordre n dont toutes les entrées sont égales à $1/n$, et $\bar{W}$ un vecteur de distribution uniforme dont tous les éléments sont égaux à 1. Alors,*

1. $\lim_{t\to\infty} M^t(\lambda) = \bar{M}$ *si* $\lambda \in (0,1)$.
2. *Pour toute distribution de charge W^0, la suite $\{W^k\}$ converge vers $b\bar{W}$, avec $b = \sum_{1\le i\le n}(w_i^0)/n$.*

De plus, la vitesse de convergence de la matrice vers la distribution uniforme dépend de la valeur propre sous-dominante, c'est-à-dire de la seconde plus grande des valeurs propres.

Ces propriétés permettent de prouver la convergence **réelle**. Cela signifie que le système est assuré de se trouver à l'équilibre après un certain nombre d'itérations *a contrario* d'autres méthodes comme la théorie des files d'attente qui le prouve simplement d'une manière probabiliste. Les propriétés de la matrice permettent d'estimer la valeur optimale du paramètre λ pour différentes architectures (nombre de couleurs).



$$M_1(\lambda) = \begin{pmatrix} 1-\lambda & \lambda & 0 & 0 \\ \lambda & 1-\lambda & 0 & 0 \\ 0 & 0 & 1-\lambda & \lambda \\ 0 & 0 & \lambda & 1-\lambda \end{pmatrix} \qquad M_2(\lambda) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1-\lambda & \lambda & 0 \\ 0 & \lambda & 1-\lambda & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

FIG. II.8 - *Construction de matrices d'un graphe coloré*

Xu et Lau ont étudié la modification du paramètre λ selon la position du lien dans la topologie. Intuitivement, il semble logique d'augmenter la valeur de λ dans certains cas. Par exemple dans une chaîne, les deux processeurs situés aux deux extrémités devraient utiliser un paramètre λ de forte valeur, tandis que les processeurs situés dans le milieu de la chaîne auraient plutôt intérêt à utiliser des valeurs moindres. Une méthode permettant de choisir la valeur des différents λ et leur placement sur une topologie donnée n'a pas été trouvée. Seule la convergence d'une méthode avec différentes valeurs du paramètre d'échange a pu être prouvée.

La méthode *d'Échange Dimensionnel* suppose que la charge est toujours divisible en 2. Hosseini et *al* [HLMV87] ont étudié les éventuels effets de bord que provoquent une non divisibilité de la charge en deux parties. Ils démontrent que le système atteint un état « presque » équilibré où à tout instant la différence de la charge pour tout processeur à la moyenne est majorée.

2.3 Conclusion

Les méthodes itératives matricielles permettent d'étudier le comportement asymptotique des algorithmes d'équilibrage dynamique. La notion primordiale de convergence d'une technique de rééquilibrage est étudiée. Les nombreuses méthodes numériques de calcul de valeurs propres ont permis l'estimation de la vitesse de convergence.

Les méthodes itératives matricielles

La technique d'analyse dite de l'*Échange Dimensionnel* est adaptable sur un système SIMD, elle se fonde sur le principe du déclenchement synchrone qui est la base des architec-

tures que nous étudions. Nous utiliserons cette méthode pour la comparaison de deux de nos algorithmes au chapitre IV.

3 Utilisation de la fonction d'iso-efficacité

La méthode d'iso-efficacité permet d'apprécier l'extensibilité des algorithmes de rééquilibrage pour une topologie donnée. Comme le font remarquer Kumar et Karypis [KK92], le coût des techniques d'équilibrage est complexe à analyser du fait même de leur dynamicité. Les performances des différentes techniques sont très sensibles aux changements des caractéristiques du matériel tels que le réseau d'interconnexion, la vitesse des unités centrales, la bande passante des canaux de communication...

Lorsqu'un algorithme parallèle est utilisé pour résoudre un problème de taille fixe, on constate que son efficacité décroît avec l'augmentation du nombre de processeurs. La raison en est, que l'overhead grandit avec le nombre de processeurs. Pour d'autres algorithmes parallèles, pour un nombre fixe de processeurs, l'efficacité grandit lorsque la taille du problème augmente, car le sur-coût augmente moins rapidement que le travail total. Pour ces algorithmes, l'efficacité peut être maintenue à un niveau constant avec l'accroissement des processeurs si la taille du problème augmente également. Un tel algorithme est dit **extensible.**

Pour un algorithme parallèle donné, le taux dans lequel doit être augmenté le travail pour conserver une efficacité constante en fonction du nombre de processeurs est appelé **fonction d'iso-efficacité** [KR87, KGR91]. Si la quantité de travail doit augmenter de façon exponentielle lorsque le nombre de processeurs croît linéairement, l'extensibilité de l'algorithme est considérée comme faible. Par contre, si le travail s'accroît de manière linéaire avec le nombre de processeurs pour une efficacité identique, l'algorithme possède une excellente extensibilité. L'analyse de l'extensibilité faite en conjonction avec la topologie du système est fort utile pour sélectionner l'architecture la mieux adaptée à un problème donné [KG91, Hwa93].

Les algorithmes étudiés sont les classiques algorithmes *Receveur/Demandeur* suivant différentes politiques d'appariement. La politique de sélection mise à la disposition des algorithmes est basée sur la technique de partage α/β [KR87, FM87].

Définition 3 *Une charge de travail W est partagée en deux entités W_j et W_k suivant la méthode α/β si elle respecte la propriété suivante: aucune des deux charges W_j et W_k ne peut être inférieure à αW, où α est une constante telle que* $0 < \alpha \leq 0.5$

Cette condition est imposée afin d'avoir une fonction de partage raisonnable. La méthode

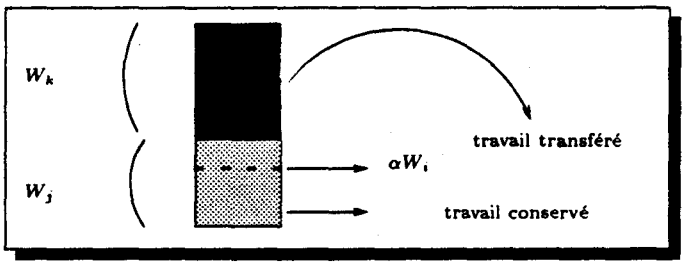


FIG. II.9 - *Mécanisme de partage α/β*

de partage α/β permet l'analyse de techniques de réarrangement dynamique.

Lorsqu'un processeur de charge W_i partage son travail en deux de taille respective W_j et W_k suivant la méthode α/β, W_k est envoyé sur le processeur demandeur, la partie W_j reste sur place. Aucun des deux processeurs (le donneur et l'envoyeur) ne possédera une charge supérieure à $(1-\alpha)W_i$. Un exemple de partage pour cette méthode est présenté sur la figure II.9. Un processeur ne peut plus diviser son travail si sa charge est inférieure à ϵ (ϵ étant généralement égal à 1).

Le demandeur de travail possède une charge nulle. On sait d'après la propriété du partage α/β que la charge $W_k < (1-\alpha)W_i$ est envoyée vers le processeur distant et que la charge $W_j < (1-\alpha)W_i$ est conservée localement. Après le transfert, les deux processeurs possèdent une charge inférieure à $(1-\alpha)W_i$. Supposons que $V(P)$ soit le nombre de phases de rééquilibrage nécessaires pour que **tous** les processeurs aient échangés au moins une fois leur travail. Si le processeur 0 possède initialement W unités de travail et tous les autres sont oisifs, le calcul de la distribution de charge sur les processeurs peut être mené de la façon suivante :

- après $V(P)$ phases d'équilibrage, le travail maximum disponible dans chaque processeur est inférieur ou égal à $(1-\alpha)W$;
- après $2 \times V(P)$ phases d'équilibrage, le travail maximum disponible dans chaque processeur est inférieur ou égal à $(1-\alpha)^2 W$;

  $\vdots$
- après $(\log \frac{1}{1-\alpha} \frac{W}{\epsilon})V(P)$ phases d'équilibrage, le travail maximum disponible dans tout processeur est inférieur à ϵ.

Il « suffit » d'évaluer la valeur de $V(P)$ afin de pouvoir calculer le sur-coût induit par l'équilibrage dynamique qui est directement proportionnel à $V(P)$ [FM87, KR87, KGR91]. Plusieurs routines de réarrangement dynamique ont été analysées en utilisant la méthode

d'iso-efficacité. Nous présentons l'algorithme du *Petit Carrousel* pour illustrer la méthode d'iso-efficacité.

3.0.1 Étude du Petit Carrousel par la méthode d'iso-efficacité

Politique d'informations La politique d'informations s'appuie sur un échange de données effectué à la demande des processeurs.

Politique de déclenchement La politique de déclenchement est autonome.

Politique de sélection Le *Petit Carrousel* est une technique dont l'initiative est laissée au receveur. Un nœud participe aux échanges de travail lorsqu'il entre dans l'état inactif, c'est-à-dire lorsque sa charge devient nulle.

Politique de désignation locale La politique de désignation locale utilisée est le mécanisme de partage α/β.

Politique d'appariement Chaque processeur possède une variable indépendante *cible* et fera ses demandes de travail au processeur pointé par la variable *cible*. Si P est le nombre de processeurs, la variable *cible* est incrémentée (modulo P) à chaque fois qu'une requête est réalisée.

Étudions le nombre de phases d'équilibrage nécessaires au *Petit Carrousel*.

Le cas le plus défavorable se présente lorsque l'ensemble des processeurs effectuent une demande de travail de façon quasi simultanée et que les variables *cible* pointent sur le même processeur.

Supposons que sur un système de P processeurs numérotés de 0 à $P-1$, le processeur numéro $(P-1)$ possède l'intégralité du travail. Supposons de plus, que les variables *cible* des autres nœuds pointent sur le processeur numéro 0. Après chaque requête infructueuse, le processeur demandeur incrémente sa variable *cible*. Pour obtenir une partie de la charge du processeur $P-1$, le processeur 1 devra effectué $P-1$ requêtes et les $P-2$ processeurs restants effectuant pendant ce laps de temps jusqu'à $P-2$ demandes. Ainsi, $V(P)$ vaut dans ce cas $(P-1)+(P-2)(P-2)$, la borne supérieure de $V(P)$ est en $O(P^2)$.

3.1 Conclusion

L'avantage offert par la méthode d'iso-efficacité est qu'elle permet aisément le calcul de la fonction d'iso-efficacité pour différentes topologies, comme les architectures hypercubes, les réseaux de station de travail, ou encore les anneaux. Cette différenciation est introduite

par le calcul d'un paramètre donnant le temps nécessaire pour effectuer un transfert dans les différents modèles.

D'autres méthodes comme la *Grand Carrousel*, l'algorithme *Aléatoire* ou encore la technique du *Plus Proche Voisin* ont été étudiées grâce à cette méthode [KGR91, KGV94]. Le tableau II.1 donne quelques résultats de la fonction d'iso-efficacité pour diverses architectures pour les techniques du *Carrousel Asynchrone* et *Aléatoire*.

TAB. II.1 - *Fonctions d'iso-efficacité pour différentes architectures. Ces valeurs indiquent la proportion dans laquelle le travail doit être augmenté afin de maintenir une efficacité constante lorsque le nombre de processeurs croît.*

Topologie	*Petit Carrousel*	*Aléatoire*
Cube	$O(P^2 \log^2 P)$	$O(P \log^3 P)$
Anneau	$O(P^3 \log P)$	$O(P^2 \log^2 P)$
Mémoire partagée	$O(P^2 \log P)$	$O(P \log^2 P)$
Grille	$O(P^{2.5} \log P)$	$O(P^{1.5} \log^2 P)$
Réseau	$O(P^3 \log P)$	$O(P^2 \log^2 P)$

L'analyse de l'extensibilité de diverses techniques d'équilibrage dynamique permet l'étude des performances et de faisabilité des algorithmes. Elle a permis de démontrer pour un hypercube que le *Grand Carrousel* et l'allocation *Aléatoire* sont les méthodes qui possèdent les meilleures fonctions d'iso-efficacité. Cependant, en l'absence de dispositif matériel évitant la formation de goulot d'étranglement, ces performances peuvent se dégrader fortement lorsque le nombre de processeurs devient très important. Le principal intérêt de cette méthode est de permettre pour une architecture donnée de déterminer la technique d'équilibrage dynamique la mieux adaptée. Elle ne permet cependant pas de déterminer pour un problème donné, le temps nécessaire à son achèvement.

Utilisation de la fonction d'iso-efficacité

La méthode d'iso-efficacité a également été utilisée pour l'analyse de l'équilibrage dynamique pour une machine SIMD [KK92]. Le principe utilisé est le même que celui employé pour le calcul de la fonction d'iso-efficacité sur une machine asynchrone tout en prenant en compte divers paramètres tels que le temps moyen d'une phase de redistribution des données, le travail non effectué lorsqu'un ratio de PEs est inoccupé... Pour la technique de *Rendez-vous* présentée au chapitre I, la fonction d'iso-efficacité est en $O(P \log P)$ où P est le nombre total de processeurs élémentaires.

4 Autres outils mathématiques

Cette partie traite d'autres outils mathématiques visant à analyser et à démontrer la convergence des solutions proposées. Les approches basées sur des outils probabilistes permettent l'obtention de résultats statistiques sur l'efficacité des algorithmes. Enfin, sont présentées d'autres méthodes comme les méthodes itératives asynchrones et le cas de la machine dictionnaire qui s'exécute sur une architecture SIMD.

4.1 Utilisation de techniques probabilistes

Du fait de leur comportement difficilement prévisible, peu de méthodes offrent une connaissance complète du comportement des algorithmes d'équilibrage et de partage de charges ; par contre les méthodes probabilistes offrent un excellent moyen de connaissance partielle du système étudié.

Définition 4 *Un algorithme est dit probabiliste, si les actions effectuées sont aléatoires et choisies parmi l'espace de probabilités [Rab76].*

L'ordonnanceur de tâches proposé par Stankovic [Sta84] peut être considéré comme un algorithme probabiliste. Il utilise la théorie de décision baysienne pour décider des actions futures.

La théorie du calcul de probabilité peut être appliquée, soit dans l'utilisation des algorithmes probabilistes [SU87], soit au contraire en analysant le comportement d'algorithmes « classiques » au moyen de variables aléatoires [QY94]. On parle plutôt dans le dernier cas d'analyse probabiliste.

4.1.1 Les algorithmes probabilistes

Shamir et Upfal [SU87] étudie le comportement d' algorithmes asynchrone et synchrone probabilistes de partage de charges et en effectue une analyse. Pour illustrer la démarche d'étude d'un algorithme probabiliste, nous présentons l'analyse d'une technique asynchrone.

Étude d'un algorithme probabiliste asynchrone

L'algorithme étudié possède une stratégie asynchrone à l'initiative du receveur. Il n'y a pas de diffusion d'informations, la communication est effectuée par envoi de messages. k

représente le nombre de processeurs oisifs, n le nombre total de processeurs du système.

Politique d'informations La collecte d'informations est effectuée uniquement à la demande des nœuds demandeurs de travail.

Politique de déclenchement Une phase d'équilibrage est déclenchée de manière asynchrone par décision des processeurs.

Politique de sélection Un nœud participe aux échanges et est considéré comme un receveur potentiel s'il passe à l'état oisif.

Politique de transfert Le processeur sélectionné par la politique de sélection initie un paquet contenant C emplacements libres (C étant une constante). Le paquet ainsi créé effectue un parcours aléatoire sur l'ensemble du système jusqu'à ce qu'il rencontre un processeur libre, il revient alors vers son émetteur. Chaque processeur rencontré durant le parcours inscrit sa charge ainsi que son numéro dans un des emplacements[1].

Le paquet contient les adresses des C nœuds les plus chargés visités lors du parcours aléatoire du message. Lorsque le processeur émetteur reçoit ce paquet préalablement émis, il interroge l'ensemble des processeurs rencontrés dans l'ordre décroissant de leur charge jusqu'à trouver un processeur qui n'a pas cédé une partie de son travail.

Le but de cet algorithme est de réussir à apparier les processeurs les plus chargés avec les processeurs inoccupés. Le théorème suivant donne une borne supérieure au nombre de messages émis pour trouver un partenaire à un processeur oisif.

Théorème 3 *Il existe une constante $\gamma > 0$ telle que le nombre moyen de messages émis jusqu'à ce que les k processeurs aient trouvé un partenaire est borné par $\gamma n \log k$.*

La borne du nombre de messages émis permet d'estimer le sur-coût dû à l'équilibrage.

4.1.2 Analyse probabiliste

Qian et Yang [QY94] en utilisant les propriétés des variables aléatoires ont réalisé une analyse fine d'un algorithme d'équilibrage asynchrone simple appelé *LAL (Local Average Load - Moyenne de la Charge Locale)*:

Politique d'informations La politique d'informations est mal définie. Un nœud communique régulièrement son état à ses voisins.

1. Les auteurs n'explicitent pas le processus d'inscription dans les emplacements. Nous supposons qu'il inscrit sa charge s'il reste des emplacements libres ou si sa charge est supérieure à la plus petite des charges contenues dans les C emplacements

Politique de sélection Il n'y a pas de condition sur l'état d'un nœud pour l'autoriser à participer à l'équilibrage de charges. Tout processeur recevra et émettra du travail. C'est une politique de sélection systématique.

Politique de d'appariement La politique d'appariement de l'algorithme de *Moyenne de la Charge Locale* est systématique et distribuée. Si un processeur possède m voisins, il partagera sa charge équitablement en $m+1$ et en émettra une partie pour chacun de ses voisins, le dernier morceau étant conservé localement. Les échanges de l'algorithme sont limités au voisinage.

Sous certaines conditions, Qian et Yang montrent que la charge moyenne dans le système tend vers la même valeur pour tous les processeurs. Formellement :

$$E[\text{Diff}(t)] = 0 \tag{II.6}$$

où $E()$ est l'opérateur « moyenne » réalisé sur l'ensemble des sites et Diff(t) est la différence au temps t entre la charge actuelle d'un nœud et la charge moyenne du système .

La charge de chaque processeur constitué par l'arrivée des tâches arrivants et de l'évolution de la charge locale est décrite par une variable aléatoire. Cependant,l'équation II.6 ne garantit pas l'équilibrage du système puisque la charge de chaque processeur peut varier temporellement. Pour cette raison, la déviation moyenne de la charge est prise en compte par la variance Var[Diff(t)], et les auteurs démontrent que sous certaines conditions :

$$\text{Var}[\text{Diff}(t)] \leq \text{constante} \tag{II.7}$$

Si la variance Var[Diff(t)] $\leq$ constante $= d^2$, les principaux résultats pour la technique LAL sont résumés sur le tableau II.2 (N est le nombre de processeurs).

TAB. II.2 - *Résultats de l'analyse probabiliste*

Anneau	$d^2\frac{N}{4}$
Tore	$[0.916\log_{10}(N)+0.25]d^2$
Hypercube	$1.386d^2$

4.2 Une approche distribuée asynchrone et itérative

Bertsekas et Tsitsiklis [BT89] ont étudié par des considérations simples basées sur l'étude de la vitesse de convergence des suites, un algorithme d'équilibrage des charges où un pro-

cesseur i à des temps t compare sa charge avec l'ensemble de ses voisins ; pour chaque voisin dont la charge est inférieure à celle de i, alors une partie de la charge de i est transférée.

Formellement, le système multi-processeurs est représenté par un graphe $G = (N, A)$, où $N = 1, \ldots, n$ représente l'ensemble des n processeurs et A est l'ensemble des arêtes reliant les processeurs. L'ensemble des processeurs qui possèdent un lien direct avec le processeur i est représenté par $A(i) = \{k | k \in N \text{ et } (i,k) \in A\}$. Chaque processeur i a une charge $x_i(t) \geq 0$.

Chaque processeur i conserve en mémoire la charge de tous ses voisins $A(i)$. La charge du processeur j vue de i sera écrite $x_i^j(t)$. À cause de l'asynchronisme et des délais de communications, la valeur de $x_i^j(t)$ n'est pas égale à $x_j(t)$ mais à une de ces valeurs antérieures. On a alors : $x_j^i(t) = x_j(\tau_j^i(t))$, avec $\tau_j^i(t)$ *entier* compris entre 0 et t.

La formule générale d'équilibrage peut être écrite sous la forme :

$$x_i(t+1) = x_i(t) - \sum_{j \in A(i)} s_{ij}(t) + \sum_{j \in A(i)} r_{ji}(t) \qquad \text{(II.8)}$$

où $s_{ij}(t)$ est le nombre de processus transférés du processeur i vers j ($s_{ij}(t) = 0$ si $x_i(t) \leq x_j^i(t)$, c'est-à-dire si la charge supposée de j est supérieure à celle de i) et $r_{ji}(t)$ est le nombre de processus reçus par i et envoyés par j.

Deux groupes de suppositions sont effectuées, la première étant appelée *asynchronisme partiel* :

Supposition 1 :

(a) Pendant toute période de longueur B, chaque processeur effectue au moins un équilibrage.

(b) L'état de la charge d'un processeur est diffusée à ses voisins dès que possible.

(c) Les délais de transfert sont inférieurs à B

La deuxième supposition impose quelques contraintes sur les transferts de processus :

Supposition 2 :

(a) Si le nœud i détecte un déséquilibre, il transféra une quantité non négligeable de travail vers le processeur moins chargé. Aucune charge ne pourra être envoyé vers un nœud dont le poids est plus important.

(b) Si le nœud i détecte un déséquilibre et envoie une partie de sa charge, il lui est interdit de provoquer un déséquilibre dans le sens inverse.

Si ces deux groupes de supposition sont vérifiées, le théorème suivant est vérifié:

Théorème 4 $\lim_{t\to\infty} x_i(t) = L/n$, *où* $x_i(t)$ *est la charge du processeur* i *à l'instant* t *et* L *la charge totale du système.*

La convergence de cette suite est géométrique.

Cette approche a été étendue de façon originale par Song [Son94] en utilisant une approche similaire pour réaliser l'équilibrage dynamique sur diverses topologies. L'exemple ci-dessous illustré dans la figure II.10 suppose que le processeur 0 est l'initiateur de la technique, $x_j(t)$ indique la charge du processeur j vue par le processeur 0. On suppose que l'on peut ordonner l'ensemble des voisins de la façon suivante, $x_n(t) \geq \ldots \geq x_1(t)$. Il existe également une valeur k pivot telle que, $x_0(t) > x_i(t)$ pour $i \leq k$ et $x_0(t) \leq x_i(t)$ pour $i > k$. Cet algorithme se déroule en trois étapes:

Étape 1 : m_0 objets sont migrés vers le processeur 1 tels que $x_1(t)+m_0 = x_2(t)$ à la condition que $x_0(t) - m_0 \geq x_k(t)$, puis on répète l'opération
$x_1(t) + m_0 + m_1 = x_2(t) + m_1 = x_3(t)$ en respectant toujours la condition $x_0(t) - m_0 - 2m_1 \geq x_k(t)$

Cette opération est répétée jusqu'à ce que la condition ne soit plus vérifiée. On voit que l'algorithme tente d'égaliser au fur et à mesure le voisinage du processeur 0.

Étape 2: À la seconde étape, tous les processeurs qui ont reçus des tâches du processeur 0 peuvent en recevoir une quantité identique si la condition classique continue à être vérifiée.

Étape 3: Cette dernière étape permet d'effectuer un lissage de la charge en permettant l'envoi d'une tâche aux processeurs.

En utilisant le théorème 4 proposé par Bertsekas et *al* Song prouve que, lorsque l'algorithme se termine, la différence de charges entre deux processeurs quelconques est au maximum de d où d est le diamètre du réseau. En proposant une amélioration de l'algorithme, la différence atteint $d/2$, mais il n'est *pas possible* de prouver sa terminaison, bien qu'aucun contre-exemple n'ait été trouvé.

Cette méthode intéressante occulte quelque peu les problèmes que l'on peut rencontrer dans le cas réel, deux suppositions sont effectuées et qui semblent difficilement plausibles, à

	x_0	x_4	x_3	x_2	x_1
Charge initiale	20	9	7	5	3
Étape 1	14	9	7	7	7
Étape 2	11	9	7	7	7
Étape 3	9	9	8	9	9

FIG. II.10 - *Algorithme de Song*

savoir :

- l'équilibrage est réalisé dès qu'un processeur s'aperçoit que sa charge est devenue plus importante que celle d'un de ses voisins ;
- l'émission de l'état de la charge d'un processeur vers ses voisins est effectuée dès sa disponibilité.

4.3 Analyse de la machine dictionnaire

Gastaldo [Gas93] a également démontré la convergence de son algorithme d'équilibrage. Sa démarche est un peu particulière car elle est ciblée pour la résolution d'un problème : la machine dictionnaire sur une machine SIMD. Si les clefs permettant la recherche sont distribuées sur l'ensemble des processeurs élémentaires, l'insertion de nouveaux éléments dispersés dans l'alphabet provoque à terme un déséquilibre de la répartition. En proposant, un algorithme simple de redistribution des données sur les processeurs élémentaires, Gastaldo démontre sa convergence effective et sa stabilité dans cet état.

La technique d'équilibrage implémentée sur une machine MasPar MP-1 se décompose en une partie de pré-calcul et une partie de transfert des données. Durant la première partie, chaque processeur élémentaire p_i calcule la taille de ses donnée et la taille de toutes les données contenues par les processeurs élémentaires dont le numéro est inférieur à i. À partir de ces valeurs, chaque processeur décide si des envois s'effectueront à gauche, à droite ou dans les deux sens. Chaque processeur peut envoyer jusqu'à la moitié des données qu'il possède à chacun de ces voisins.

Afin de démontrer la convergence effective de sa méthode, Gastaldo part du cas le plus défavorable. Le cas défavorable a lieu lorsque toutes les insertions sont réalisées sur le processeur p_0. En utilisant une hypothèse de récurrence sur le nombre de données contenu dans

chaque PE à l'équilibrage t, on arrive à la conclusion que la charge du PE à t est supérieure à sa charge à $t+1$. Au bout de quelques itérations, on arrive à une stabilité de la charge sur chaque PE.

5 Discussion

Les différentes méthodes d'analyse mathématique abordées dans ce chapitre tentent toutes d'estimer les qualités ou les performances des algorithmes d'équilibrage de charge. Les points étudiés sont nombreux et les différents outils permettent aussi bien d'estimer le temps de réponse d'un processus nouvellement créé que de calculer la vitesse de convergence d'un algorithme. Sans vouloir les comparer sur une base commune, nous présentons une évaluation des différentes approches, de leurs atouts et de leurs points faibles.

Dans les paragraphes suivants, nous présentons les mérites des différentes méthodes d'analyse évoquées dans ce chapitre. Nous nous intéresserons particulièrement à la présence ou à l'absence d'un paramètre estimant le coût des transferts, la convergence effective des algorithmes et la vitesse de cette convergence.

5.1 Estimation du coût de transfert

Peu de méthodes d'analyse offrent la possibilité de calculer ou d'estimer le sur-coût induit par le transfert de tâches d'un processeur vers un autre. Ainsi, les méthodes matricielles [Cyb89] occultent totalement ce fait. Seules les analyses utilisant la théorie des files d'attente ou celles utilisant la fonction d'iso-efficacité définie par Kumar permettent l'introduction de ce concept. La théorie markovienne permet aisément d'incorporer le coût d'un transfert en l'ajoutant dans la liste des paramètres; Livny et Melman [LM82] utilise des valeurs constantes comme les temps de transmission ; l'estimation du coût faite par Chowdhury [Cho90] est encore plus fine puisque les informations comme le délai nécessaire pour l'empaquetage d'une tâches est présente ; Eager et *al* [ELZ86a] par contre représentent le coût de transfert d'une tâche d'un nœud vers son partenaire par un coût supplémentaire du nœud envoyeur.

Kumar et Karypis [KGR91, RK87, KR87] utilisent une borne supérieure afin d'intégrer la somme du temps passée par l'ensemble des processeurs pour communiquer entre eux, attendre la réception de messages, obtenir du travail d'un partenaire distant... La définition de ces valeurs dépend de la taille du message transféré, de la distance entre les processeur envoyeur et receveur, de la vitesse de communication et donc dépend *directement de l'architecture utilisée.* C'est une des rares techniques qui permettent d'apprécier les coûts de transferts pour différentes topologies ; ainsi transférer un processus d'un nœud vers un autre aura une valeur en

$O(1)$ pour un réseau de stations de travail, une valeur de $O(\log(\text{nombre de processeurs}))$ pour un hypercube ou bien encore en $O(\sqrt{\text{nombre de processeurs}})$ pour une grille. L'estimation de ce coût de transfert revêt une importance considérable puisque Eager et *al* [ELZ86a] montrent qu'il existe un seuil pour le coût de transfert au-delà duquel les performances d'un système multi-processeurs se dégradent très rapidement.

5.2 Preuve de la convergence

On entend par convergence d'une technique d'équilibrage, le fait d'obtenir en inhibant les entrées (arrivée de nouvelles tâches, création de nouveaux processus) un système dont les nœuds contiennent une « charge » identique au bout d'un certain nombre d'itérations. La majorité des travaux traitant de l'équilibrage dynamique s'appuie sur des simulations afin de prouver leur bon comportement. Les méthodes analytiques basées sur la théorie des files d'attente n'apportent pas de preuve de convergence, elles ne s'intéressent qu'aux temps de réponse moyen d'un système, c'est-à-dire pour un processus possédant un temps d'exécution moyen fixé, quel sera le temps total passé dans le système [Zho88, CK87]. Ces algorithmes sont donc généralement plus utilisés dans la conception de certains noyaux de systèmes d'exploitation réparti tel CHORUS [Phi93], le but premier d'un système d'exploitation distribué n'étant pas d'obtenir une charge identique sur tous les nœuds mais de minimiser le temps de réponse d'une commande de l'utilisateur ou d'un processus. L'utilisation du calcul matriciel permet quant à lui de prouver la convergence des algorithmes ; la machine dictionnaire de Gastaldo [Gas93] est un cas atypique puisqu'il s'agit de remanier la structure des données sur l'ensemble des processeurs, par exemple lors d'insertions de nouvelles clefs. La preuve de convergence apportée par Bertsekas et Tsitsiklis [BT89] indique qu'il suffit à des instants données, que les processeurs partagent leur travail avec leurs voisins et ce de façon asynchrone pour obtenir un système dont la charge des nœuds tend vers la moyenne du système.

5.3 Vitesse de convergence

La vitesse de convergence correspond au nombre d'itérations ou au nombre d'appels à la routine de réarrangement dynamique ou au temps nécessaire pour obtenir un système équilibré. Elle permet d'estimer la vitesse de propagation des méthodes à diffusion. L'équilibrage parfait n'est pas instantané, les charges se propagent de manière plus ou moins rapide dans l'ensemble du système. Un exemple typique de méthode par diffusion est l'algorithme du gradient [LK87]. Les processeurs dont la charge est excessive envoient une portion de leur charge vers le plus proche processeur faiblement chargé. Le résultat est une forme de relaxation où les tâches migrent à travers le système des zones à forte gravité (présence de nombreux processeurs fortement chargés) vers des endroits où la gravité est plus faible.

La méthode définie par Cybenko [Cyb89] permet par l'utilisation du théorème de Perron-Frobenius [Var62] de calculer la vitesse de convergence pour un algorithme ayant une représentation itérative matricielle. Cette vitesse est directement proportionnelle à la seconde plus grande des valeurs propres de la matrice de diffusion. De façon similaire, la *cadence* du processeur définie par Mans [Man92] donne la variation positive ou négative de la charge du processeur suivant le temps. Bien évidemment les méthodes dites de partage de tâches (« load sharing ») ne sont pas concernées par la vitesse de convergence.

6 Conclusion

L'étude de l'analyse mathématique des techniques d'équilibrage est un domaine récent et en perpétuelle évolution. Il permet d'aborder le comportement de ces techniques de redistribution sous un angle différent qui intègre l'étude de son évolution. Diverses méthodes comme la théorie des files d'attente, l'analyse matricielle ou probabiliste... abordent le problème selon divers critères suivant que l'on cherche à quantifier un comportement moyen ou prouver une terminaison effective. Peu d'approches SIMD ont été proposées mais certaines comme le coefficient d'iso-efficacité sont directement applicables.

Il apparaît très appréciable de pouvoir estimer de manière grossière le comportement d'un algorithme grâce à de telles méthodes. Les approches mathématiques et expérimentales sont totalement complémentaires. La simulation permet une confrontation directe aux cas « réels ». C'est dans cette optique qu'en sus des expériences, nous proposons l'analyse de quelques techniques d'équilibrage dans le chapitre IV.

Chapitre III

L'équilibrage sur une machine SIMD

1 Motivations

Les techniques d'équilibrage dynamique proposées dans ce chapitre sont ciblées pour les algorithmes à pile où les données peuvent être représentées sous la forme d'une pile distribuée.

Comme on a pu le voir dans le chapitre I traitant de la comparaison des approches MIMD/SIMD, l'équilibrage sur une machine synchrone peut tout à fait se concevoir et être utilisée avec succès. La première partie de notre travail a consisté à définir un modèle d'équilibrage dynamique à parallélisme de données. Des enseignements que nous avons tirés de la comparaison au modèle MIMD, nous avons construit un modèle SIMD dont nous avons identifié quatre critères :

Politique de déclenchement Faut il entreprendre une phase de rééquilibrage afin d'améliorer le rendement de la machine parallèle?

Politique de sélection Quels processeurs élémentaires participeront à l'échange de données durant l'équilibrage?

Politique de domaine et de communication Quel type de communication utiliser? Sur quel domaine ou sous-domaine des PEs les échanges vont ils être réalisés?

Politique d'appariement À l'intérieur d'un domaine et pour un processeur donné, avec quel(s) partenaire(s) va-t-il échanger des objets?

Nous avons par la suite clairement séparé le processus de décision des autres critères, car il nous apparaît que la politique de décision possède une indépendance certaine et peut

être utilisé de manière générique avec diverses applications. Nous qualifions d'algorithme de rééquilibrage les décisions relatives aux trois derniers points.

La première partie de ce chapitre concerne les notations et le langage que nous avons choisis afin d'obtenir une description efficace des différentes techniques d'équilibrage. Les différents algorithmes sont présentés suivant la politique de domaine utilisée : globale ou locale. Les politiques de décision sont introduites dans le paragraphe 4 et sont séparées en deux classes : (i) les politiques de déclenchement périodiques et (ii) et les politiques de déclenchement adaptatives.

2 Notations

Afin de présenter clairement le fonctionnement des différents algorithmes, nous introduisons dans ce paragraphe un certain nombre de paramètres et de variables ainsi qu'un langage qui sera utilisé pour la description de nos différentes techniques.

L'utilisation d'une fonction indicatrice de la charge instantanée est primordiale pour la conception efficace d'une technique d'équilibrage. Comme les algorithmes présentés dans cette section sont ciblés sur des problèmes dont le comportement est imprévisible, nous avons choisi de mettre en œuvre le mécanisme le plus simple, chaque PE estimant son occupation en calculant le nombre de données qu'il possède. Afin d'éviter de répéter le terme indicateur de charge, il sera désigné plus simplement sous le noms de poids, charge ou poids instantané.

Définition 5 *La fonction de charge instantanée w est la fonction qui a tout processeur i associe sa charge $w[i]$.*

Nous introduisons également quelques variables que nous conserverons tout au long de cette étude. Deux référentiels sont introduits afin de désigner les processeurs élémentaires de manière unique. Le terme P_i représente la vue linéaire (1 dimension) de l'ensemble des PEs, la numérotation des PEs commençant à 0. Parfois, une représentation en deux dimensions de la machine SIMD est utilisée. Dans ce cas, chaque PE sera désigné par deux indices $P_{i,j}$.

Définition 6 *À chaque PE est associé un numéro unique compris entre* 0 *et* $n-1$ *Ce numéro d'identification est appelé* index. *Il correspond à la vue linéaire de la machine.*

n est le nombre de processeurs élémentaires

Définition 7 *La charge totale instantanée N du système est l'ensemble des objets contenus*

par tous les processeurs

$$N = \sum_{i=0}^{n-1} w[i]$$

Un processeur élémentaire dit oisif, inactif ou inoccupé est un PE dont le poids instantané est *nul.*

Sur une machine de type SIMD, deux types de variables peuvent être utilisés :

Les variables parallèles sont allouées sur chaque processeur élémentaire. Elles seront désignées sous la forme $x[k]$ où k désigne l'index d'un processeur.

Les variables scalaires sont allouées sur le séquenceur. Pour les distinguer des variables parallèles, elles seront écrites sans crochet.

Un processeur élémentaire d'une machine SIMD peut se trouver dans deux états : actif ou inactif. Dans l'état actif, il obéit aux instructions transmises par le séquenceur. Dans l'état inactif, aucune instruction n'est réalisée. Cette activité est dirigée par le séquenceur.

Afin de décrire nos algorithmes, nous utiliserons la notation spécifique au modèle de programmation à parallélisme de données introduite par Hillis et Steele [HS86]. Elle permet d'indiquer avec clarté le déroulement d'un programme à parallélisme de données. Nous avons classé les instructions de description en trois catégories :

1. Les *instructions* du langage de Hillis et Steele provoquant une ou des opérations parallèles et séquentielles.
2. Les instructions provoquant des *transferts* d'entités entre les piles distribuées.
3. Les opérateurs permettant de prendre en compte les communications de *voisinage.*

2.1 Instructions data-parallèles

Les commandes suivantes présentent la base nécessaire pour décrire un algorithme data-parallèle. s désigne une variable de type scalaire, $par[k]$ une variable parallèle contenue sur chaque PE.

`for all` k `in parallel do` c `od`

Cette instruction provoque l'exécution simultanée par l'ensemble des processeurs actifs de

resultat[*k*] :=**rank**(i[k])

k (index)	0	1	2	3	4
Activité	1	1	0	1	1
i	4	7	2	2	0
resultat	2	3	–	1	0

FIG. III.1 - *Fonctionnement de l'instruction* **rank**

la commande *c*. *k* désigne l'index des processeurs élémentaires. Cette instruction déclenche l'exécution d'un programme data-parallèle.

s :=**sum**(*par*[*k*])
L'opérateur **sum**(*var*[ ]) est utilisé pour calculer la somme de la variable parallèle (*var*[ ]) détenue par les PEs actifs. Le résultat obtenu est de type scalaire. Le coût en communications de l'instruction **sum** est en $O(\log(n))$.

s :=**count**
L'instruction **count** compte le nombre de processeur actifs. Le résultat retourné est compris entre 0 et $n-1$. Le décompte du nombre de processeurs actifs a un coût en communications en $O(\log(n))$.

par[*k*] := **enumerate**
L'instruction **enumerate** affecte à chaque PE actif un entier distinct. La numérotation commence à 0 pour le PE actif ayant l'index le plus bas et est incrémenté de 1 dans l'ordre de l'index pour chaque PE actif. Le coût en communications de **enumerate** est en $O(\log(n))$.

*par*1[*k*] :=**rank**(*par*2[*k*])
L'opérateur **rank** provoque un classement de tous les processeurs actifs suivant la valeur locale de la variable *par*2[ ]. Un exemple d'utilisation est présenté sur la figure III.1. Le coût en communications est en $O(log^2(n))$.

Remarque :

L'ensemble des commandes data-parallèles intègrent de manière implicite le transfert de valeurs entre les processeurs. L'opération $x[k] = y[k+1]$ provoque pour chaque processeur élémentaire l'obtention de la valeur de la variable $y[\ ]$ de son successeur[1]. Les instructions peuvent être composées pour réaliser une indirection supplémentaire. Une commande data-parallèle du type $par[k] := a[b[k]]$ entraîne pour chaque PE la lecture de la variable sur le processeur dont l'index est contenu dans la variable $b[\ ]$. La figure III.2 présente un exemple

1. Le successeur désigne dans ce cas le PE qui le suit dans l'ordre des numéros d'identification

$resultat[k] := a[b[k]]$

k (index)	0	1	2	3	4
Activité	1	1	0	1	1
b	2	1	2	4	3
a	6	8	0	7	7
resultat	0	6	–	7	7

FIG. III.2 - *Présentation de l'indirection pour les communications*

de l'utilisation d'indirection.

Nous avons dû étendre les opérations existantes afin de pouvoir intégrer le transfert de données dans la description de nos algorithmes.

2.2 Fonctions de transfert de charge

Les algorithmes visés pour les techniques d'équilibrage SIMD sont les algorithmes distribués à pile. Les fonctions introduites dans cette section permettent le transfert de données d'une pile vers une autre. Elles sont écrites à partir des instructions vues dans le paragraphe précédent.

send($pe[k]$,$taille[k]$)
L'instruction **send**($pe[k]$,$taille[k]$) réalise une émission taille $pe[k]$ données à partir des piles locales des processeurs actifs vers les piles des processeurs $pe[k]$. La fonction indicatrice de la charge des processeurs élémentaires cibles est augmentée en conséquence.

receive($pe[k]$,$taille[k]$)
L'opérateur **receive** réalise l'opération duale de la précédente. Il provoque la réception par les piles des processeurs actifs de $taille[k]$ entités provenant des piles des processeur $pe[k]$. La charge locale des processeurs actifs est augmentée de $taille[k]$.

Les variable $pe[k]$ et $taille[k]$ sont *parallèles*, ce qui signifie que les commandes **send** et **receive** sont effectuées en *parallèle*, chaque processeur élémentaire actif possédant une valeur locale propre pour $pe[k]$ et pour $taille[k]$. La figure III.3 présente l'emploi de l'instruction **send**. Notons que dans toutes les politiques présentées dans ce chapitre, l'instruction de transfert **send** utilise des schémas de communications sans conflit, c'est-à-dire basé sur une permutation.

i	2	4	4	1
Activité	1	1	0	0
Charge initiale $w[k]$	5	3	4	2
Charge finale $w[k]$	3	1	6	4
k (index)	1	2	3	4

FIG. III.3 - *Action de l'opération* `send`*(i,2). La variable* 2 *dans cette instruction est un* 2 *« parallèle » ; chaque processeur émettant 2 données de sa pile locale.*

2.3 Extension de la notation data-parallèle aux transferts de voisinage

Sur les machines SIMD, les communications de voisinage occupent une place prépondérante. Les systèmes actuels atteignent ou dépassent aisément plusieurs milliers de processeurs. La communication entre plusieurs PEs situés à des endroits quelconques dans le système est plus lente que l'échange d'informations entre deux processeurs élémentaires placés matériellement côte à côte. L'exploitation habile des communications de voisinage permet d'améliorer de manière sensible les performances d'une application. Cette prise en compte de la topologie du système doit être réalisée de la façon la plus générique possible. Le référentiel doit s'adapter à l'architecture sur laquelle il s'applique. Pour prendre en compte les communications de voisinage dans une structure bi-dimensionnelle, un index/référentiel comportant 2 indices est indispensable.

Dans les exemples que nous présentons, l'introduction du concept de voisinage en deux dimensions est suffisant. Chaque processeur aura la possibilité d'effectuer des communications ou des transferts de données avec ses 4 voisins : nord, sud, est et ouest. Afin d'exprimer correctement l'architecture 2D, l'instruction `for all` k `in parallel do` s `od` se transforme en `for all` i, j `in parallel do` s `od`. Cette opération provoque l'exécution synchrone de la même commande s dans toutes les colonnes j et dans toutes les lignes i. Bien sur, les variables i et j comme l'index mono-dimensionnel possèdent des valeurs différentes sur chaque processeur (respectivement l'abscisse et l'ordonnée).

Il est possible d'ordonner les voisins d'un processeur élémentaire de façon unique (en utilisant le sens trigonométrique par exemple pour une topologie à 2 dimensions). Parler du i^e voisin d'un PE a donc un sens précis et désigne pour chaque PE un unique PE.

Dans les communications produites par un algorithme data-parallèle, deux types de communications peuvent être distingués :

Les communications globales sont des communications effectuées par des processeurs situés à une place quelconque dans le système.

Les communications uniformes sont caractérisées par une communication dans la même direction et à la même distance de tous les processeurs élémentaires. Par exemple sur une topologie mono-dimensionnelle, l'expression $a[k] := a[k + d]$ où d est un entier scalaire réalise une opération de communication uniforme. Chaque PE accédant au PE situé à une distance d.

Les communications induites par les opérations de transfert de charge **send**$(pe[k], taille[k])$ et **receive**$(pe[k], taille[k])$ peuvent être globales ou uniformes suivant la valeur de la variable parallèle $pe[k]$.

Architecture 1D : Si $pe[k]$ prend des valeurs quelconques, le schéma de communications sera global. Par contre si $pe[k]$ est de la forme $k + d$ où d est une valeur scalaire de type entier, les communications seront uniformes.

Architecture 2D : Les communications sont uniformes si $pe[k]$ est de la forme $[k + \alpha, k + \beta]$ où α et β sont des entiers de type scalaire. Comme cette notation est « lourde », nous utiliserons une nouvelle fonction réalisant cette opération de façon transparente: `mon_voisin`

La fonction $pe[k]$:=`mon_voisin`$(i, distance)$ permet d'associer à tout PE actif l'adresse du processeur situé à une distance *distance* dans la direction du ième voisin. Si la distance est égale à 1 on accède directement à l'adresse du ième voisin. Les opérations **send**(`mon_voisin`$(i, distance), taille[k])$ et **receive**(`mon_voisin`$(i, distance), taille[k])$ garantissent un schéma de communications uniformes. La notation 1D $w[k] + w[k + d]$ sera « transformée » en 2D en $w[k] + w$[`mon_voisin`(i, d)].

La fonction `mon_voisin` peut être utilisée pour des topologies de dimension supérieure.

3 Les algorithmes

Les techniques de réarrangement dynamique permettent l'échange de données entre piles distribuées.

Diverses stratégies sont envisagées. Certaines pour un coût d'équilibrage relativement élevé permettent d'obtenir un équilibrage quasi parfait ; d'autres pour un coût moindre réalisent un équilibrage de plus ou moins bonne qualité. La plupart des algorithmes présentés sont divisés en deux sections distinctes, une phase calculatoire plus ou moins importante et une phase de mouvement des données.

Dans le reste de ce chapitre, nous présentons les différents algorithmes suivant la politique de domaine/communication qu'ils utilisent, soit une politique globale dans laquelle les données

sont propagées dans tout le système, soit au contraire une politique accentuée sur la localité où les transferts sont limités au seul voisinage. Notre choix de description s'explique par l'importance que revêtent les communications inter-voisinages pour les machines SIMD. Nous présenterons enfin une solution mixte ou *Hybride* qui utilise une politique de communications globale et locale.

3.1 Les algorithmes mettant en œuvre une politique de domaine globale

Une politique de domaine globale permet d'éviter une trop grande localité de l'équilibrage. Elle permet de mieux réagir à une surcharge subite de travail, localisée spatialement. Par contre le désavantage d'une telle méthode est son insouciance des communications induites ; les PEs pouvant alors partager leur travail avec d'autres processeurs situés à une place quelconque du système et créer des conflits [2] et des goulots d'étranglement sur certaines régions du système.

Les algorithmes *Central* et *Rendez-vous* possèdent une politique de domaine étendue à l'ensemble des processeurs. Ces deux techniques autorisent des communications globales en couplant les processeurs 2 à 2, l'algorithme *Rendez-vous* proposant une méthode de couplage plus performante que celle proposée par la technique *Central*.

3.1.1 L'algorithme Central

Motivations

La technique d'équilibrage *Central* s'inspire des méthodes classiques de rendez-vous utilisées dans divers algorithmes SIMD [PFK93] et originellement proposée par Hillis [Hil88] sous le nom d'allocation par rendez-vous. L'algorithme *Central* repose sur le fait que, sur les architectures SIMD, on peut facilement réaliser une énumération des processeurs. Dans une première phase, les ensembles de processeurs actifs et inactifs sont énumérés et mis en relation afin de provoquer un échange de travail. L'algorithme *Central* reprend ce concept mais modifie légèrement la politique de sélection. Au lieu d'apparier un PE oisif avec n'importe quel PE possédant une charge non nulle, nous n'autorisons la rencontre des PEs inactifs qu'avec les processeurs dont le poids est supérieur à la charge moyenne du système. Cette modification de la politique de sélection présente l'avantage de transformer une politique à base de seuil unique en une technique à double seuil. L'algorithme *Central* possède un seuil bas dont la valeur est fixée à 1 et un seuil haut qui s'adapte à l'évolution du système. La création d'un

2. Des conflits sont engendrés sur certaines machines. Sur la MasPar MP-1, chaque grappe de 16 PEs est reliée par un fil unique au réseau global. L'émission vers plusieurs processeurs de la même grappe provoque des conflits qui sont résolus séquentiellement

second seuil permet en outre:

- d'éviter qu'un processeur oisif demande une partie de la charge d'un processeur dont le poids est relativement faible ;
- de limiter les phénomènes d'instabilité.

Implantation

Le pseudo-code est présenté dans la figure III.4, l'évolution d'un exemple est présentée en parallèle à la figure III.5.

Durant la phase d'initialisation, la charge totale du système N est calculée en sommant la charge locale de chaque PE. La charge moyenne du système est calculée (*seuil*), c'est la valeur qui serait attribuée à chaque PE en cas d'équilibrage parfait. Dans notre implantation de l'algorithme *Central*, la valeur *seuil* est utilisée pour désigner les PEs possédant un excès de charge. Ces PEs surchargés enverront une partie de leur travail vers des processeurs élémentaires inactifs.

Pendant, la 1re étape, seuls les PEs dont la charge est nulle seront actifs. Chaque PE sera énuméré par l'instruction `enumerate`. Chaque PE connaît son numéro d'ordre dans la liste des processeurs inoccupés.

Pendant la seconde étape, chaque PE actif de l'étape précédente envoie vers le PE dont l'index correspond à son numéro d'énumération, son numéro d'identification (son index). Dans l'exemple de la figure III.5, le PE n° 9 envoie vers le PE n° 1 son numéro d'identification, soit 9. Les processeurs situés en début de la liste d'index servent donc de boîtes aux lettres. Ils possèdent l'adresse d'un processeur inoccupé.

Pendant, l'étape 3 les PEs possédant une charge supérieure à la charge moyenne globale deviennent actifs. Ils sont énumérés comme dans l'étape 2. Le résultat de cette énumération est placé dans la variable parallèle *ami*[ ]. Il suffit de faire communiquer les PEs surchargés et les PEs inactifs par l'intermédiaire des processeurs boîtes aux lettres définis dans l'étape 2. C'est la fonction de l'étape 4, qui par l'intermédiaire de la variable parallèle *rendez_vous*[ ] récupère l'adresse « déposée » dans les processeurs boîtes aux lettres par les PEs oisifs.

L'étape 5 correspond à la phase de mouvement effective. Chaque PE envoie vers le partenaire oisif qui lui a été attribué la moitié de la charge qu'il possède.

Discussion

```
for all k in parallel do
Phase d'initialisation :
    N := sum(w[k])
    seuil := N/n
    rcv[k] :=nill
    if w[k] = 0 then
Étape 1 :
        dest[k] :=enumerate
Étape 2 :
        rcv[dest[k]] := k
    fi
    rendez_vous[k] :=nill
    if w[k] > seuil then
Étape 3 :
        ami[k] :=enumerate
Étape 4 :
        rendez_vous[k] := rcv[ami[k]]
    fi
Phase de mouvement :
    if rendez_vous[k] ≠nill
Étape 5 :
        send(rendez_vous[k], w[k]/2)
    fi
od
```

FIG. III.4 - *Pseudo-code de l'algorithme Central*

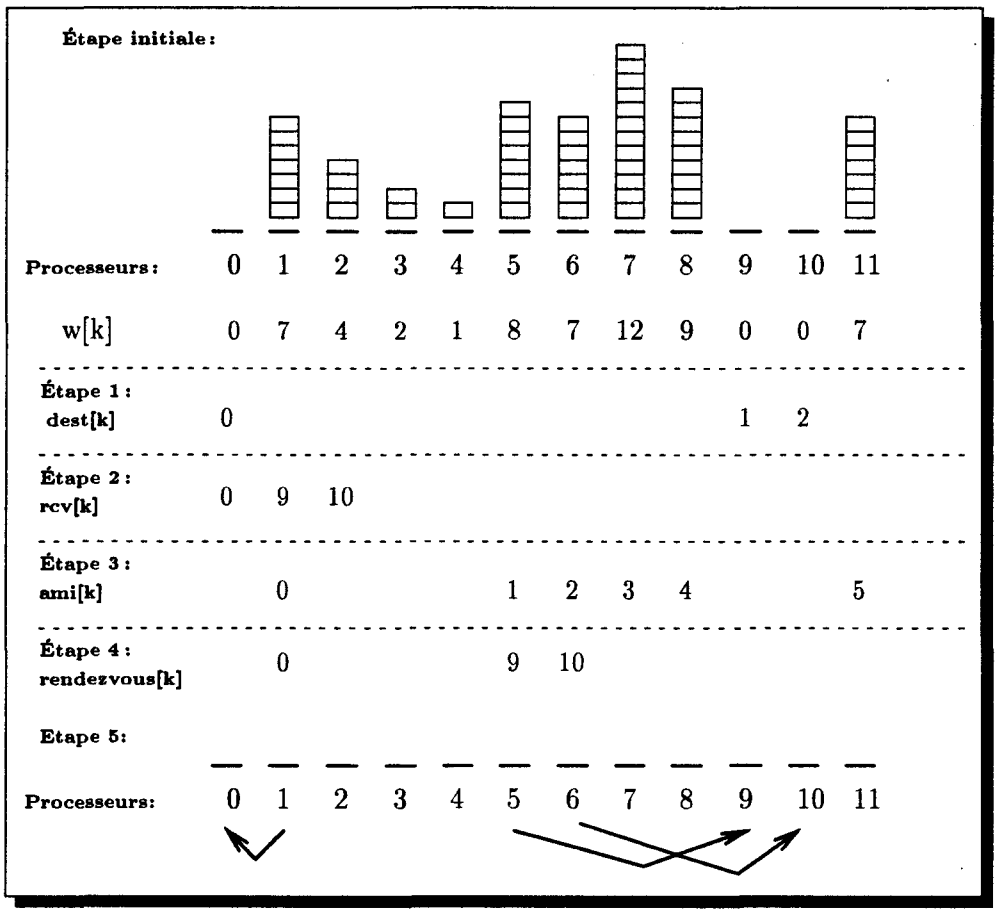


FIG. III.5 - *Exemple d'exécution de l'algorithme Central*

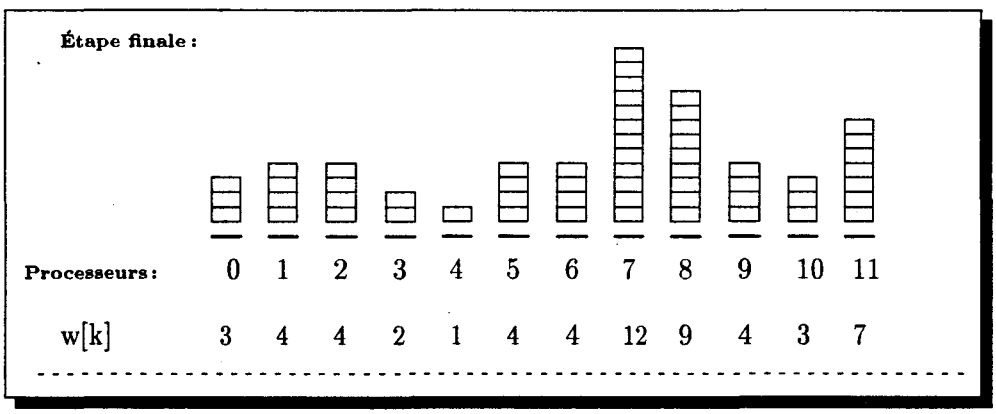


FIG. III.6 - *Résultat de l'exécution de l'exemple de la figure III.5*

L'algorithme *Central* propose une méthode simple et efficace pour réaliser une redistribution des données. L'appariement réalisé n'est pas toujours de très bonne qualité. On remarque sur la figure III.5 que le PE n° 7 qui possède la charge la plus élevée n'a aucun partenaire. Il présente, en outre, le désavantage d'utiliser un schéma de communications globales généralement coûteuses. Il ne garantit pas non plus qu'un processeur chargé aura pour partenaire le processeur oisif le plus proche. Si le nombre de processeurs inactifs est inférieur à 50%, un nouvel appel ne permet pas d'améliorer l'éventuel déséquilibre du système.

3.1.2 L'algorithme Rendez-vous

Motivations

L'algorithme *Rendez-vous* se caractérise par une politique de communications globales et un appariement performant des processeurs. L'algorithme *Central* permet l'échange de travail entre un processeur inoccupé et un processeur chargé, même si le poids du processeur chargé est « faible ». L'algorithme *Rendez-vous* permet d'apparier les processeurs élémentaires possédant les charges les plus extrêmes. Un processeur très lourdement chargé (dont le poids se situe très au-dessus du seuil) trouvera un partenaire très faiblement chargé, voire même inoccupé. De manière symétrique, un processeur dont la charge est légèrement inférieure au seuil tentera de s'apparier avec un PE faiblement excédentaire. Les processeurs utilisent une partie des autres PEs afin de de communiquer, ces derniers offrant la fonctionnalité de boîte aux lettres. Certains déposent leur adresse tandis que d'autres viennent la récupérer. Le but premier d'une telle technique est de réaliser une redistribution plus régulière des données à chaque phase d'équilibrage et donc d'effectuer un réarrangement dynamique moins fréquemment.

Implantation

La figure III.7 présente le pseudo-code de l'algorithme *Rendez-vous*. L'évolution du système ainsi que les valeurs des principales variables peuvent être suivies sur la figure III.8. L'algorithme *Rendez-vous* présente de fortes ressemblances avec la technique *Central*. Dans sa description, nous insisterons particulièrement sur les parties qui diffèrent de l'algorithme *Central*.

La phase d'initialisation est la même que celle présentée pour l'algorithme *Central*. La moyenne de la charge globale est calculée et placée dans la variable *seuil*. Elle sert dans cette technique comme seuil afin de classer les PEs parmi les surchargés ou les sous-chargés.

La 1re étape présente une différence importante par rapport à l'algorithme *Central*. Tous les PEs, (et non plus seulement les PEs oisifs), dont la charge est inférieure au seuil calculé

précédemment sont actifs. La variable *value*[ ] est utilisée comme variable temporaire afin de stocker la différence des PEs sous-chargés au seuil. Un tri sur le poids est effectué parmi les processeurs par l'instruction **rank** et est placé dans la variable *dest*[ ]. De manière distribuée, chaque PE connaît son ordre dans la liste des nœuds les moins occupés. Le processeur élémentaire possédant le poids le plus faible de l'ensemble des PEs sous-chargés se verra attribuer le n° 0. Plus le poids du PE dans cet ensemble est important, plus un numéro d'ordre élevé lui sera attribué.

La seconde étape est identique à celle de la technique *Central*. Les processeurs actifs pendant la 1re étape communiquent leur adresse au processeur dont l'index correspond au numéro d'ordre obtenu dans le tri. Ainsi, dans l'exemple de la figure III.8 le PE n° 10 enverra vers le processeur n° 2. son numéro d'identification, c'est-à-dire 10. Les processeurs dont l'index se situe au début de la numérotation servent de boîte aux lettres. Ils possèdent l'adresse d'un correspondant faiblement occupé avec la convention suivante : plus l'index du PE boîte aux lettres est bas, plus le PE correspondant est sous-chargé.

À partir, de l'étape 3, les PEs des phases précédentes sont inhibés, et on active les processeurs dont le poids dépasse le seuil. On réitère le processus effectué sur les processeurs élémentaires sous-chargés mais cette fois pour les surchargés. La variable parallèle *ami*[ ] grâce à l'opérateur **rank** contient le rang de chaque PE suivant sa charge. Comme on travaille sur l'opposé du poids, le classement obtenu est tel que le processeur élémentaire qui possède le poids le plus élevé est celui qui aura le rang le plus petit. Le rang obtenu grâce à cette instruction est symétrique au rang calculé durant l'étape 1 parmi les PEs sous-chargés.

La communication entre les deux listes de PEs est réalisée de façon identique à l'algorithme *Central*. Enfin, l'étape 5 permet l'échange de données entre processeurs surchargés et sous-chargés. Le résultat des transferts finaux est présenté dans la figure III.9.

Discussion

L'algorithme *Rendez-vous* permet un appariement de meilleure qualité que l'algorithme *Central* en permettant à des PEs dans des états extrêmes d'échanger leur travail. Lorsque le nombre de processeurs élémentaires oisifs est important ou dépasse même 50% du nombre total de PEs, il n'y a pas à se soucier de l'appariement des PEs. Tous les processeurs libres trouveront un partenaire. Par contre, lorsque le nombre de processeurs libres est faible en regard du nombre de PEs de la machine, l'appariement doit être réalisé avec beaucoup d'attention si l'on souhaite ne pas faire appel trop souvent aux routines de redistribution de données. Un processeur doit être mis en correspondance avec un PE possédant une charge importante. L'algorithme Rendez-vous permet ainsi de conserver un niveau d'utilisation des

```
for all k in parallel do
Phase d'initialisation :
    N := sum(w[k])
    seuil := N/n
    rcv[k] := 0
    value[k] := 0
    if w[k] < seuil then
Étape 1 :
        value[k] := w[k] − seuil
        dest[k] :=rank(value[k])
Étape 2 :
        rcv[dest[k]] := k
    fi
    rendez_vous[k] :=nill
    if w[k] > seuil then
        value[k] := w[k] − seuil
Étape 3 :
        ami[k] :=rank(−value[k])
Étape 4 :
        rendez_vous[k] := rcv[ami[k]]
    fi
    if rendez_vous[k] ≠nill
Étape 5 :
        send(rendez_vous[k], (w[k] + w[rendez_vous[k]]/)2)
    fi
od
```

FIG. III.7 - *Pseudo-code de l'algorithme Rendez-vous*

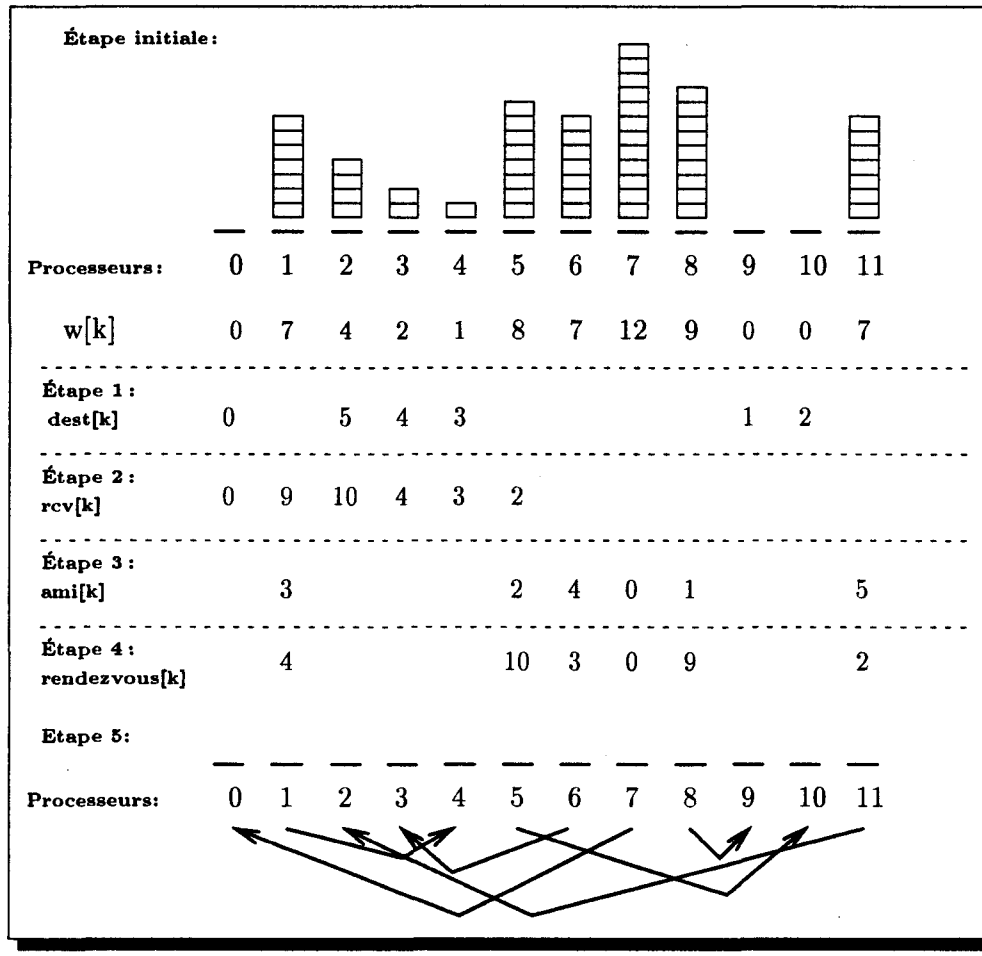


FIG. III.8 - *Exemple d'exécution de l'algorithme Rendez-vous*

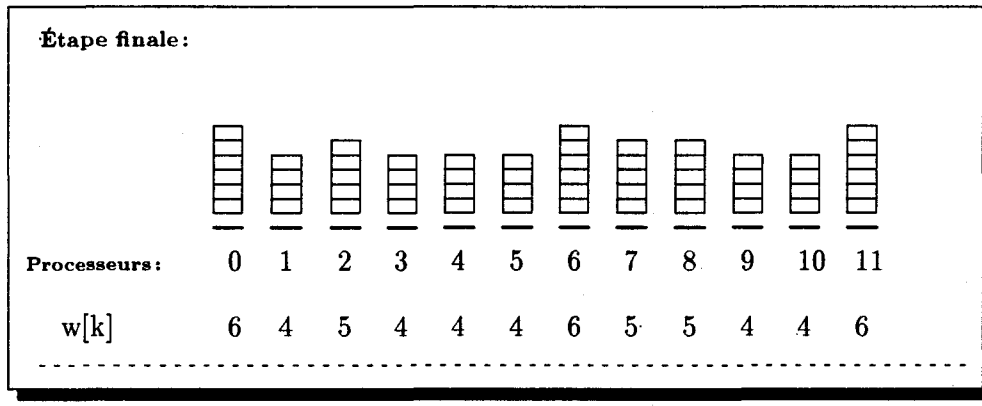


FIG. III.9 - *Résultat de l'exécution de l'exemple de la figure III.8*

PEs élevé en étant appelé moins fréquemment que la technique *Central*.

Dans l'implantation effective de la technique *Rendez-vous* sur machine SIMD, nous avons choisi une approche légèrement différente. Au lieu de trier dans une première étapes, les PEs sous-chargés puis, dans un deuxième temps les PEs excédentaires, nous effectuons un tri sur la totalité des processeurs. Le processeur dont le rang de tri est i est apparié avec le PE dont le rang est $n - i$. Cette modification permet de réaliser un seul tri (`rank`)pour la technique *Rendez-vous*.

La réitération de l'algorithme *Rendez-vous* provoque la convergence de tous les processeurs élémentaires vers la charge moyenne du système. Les PEs déficitaires s'apparient avec des PEs surchargés jusqu'à l'égalisation de la charge du système. Nous montrons dans le chapitre consacré à l'analyse mathématique que la vitesse de convergence est en log du nombre de processeurs.

3.2 Les algorithmes mettant en œuvre une politique de domaine locale

Les stratégies locales limitent les communications aux nœuds directement adjacents ou dans un proche voisinage. Ces algorithmes permettent de tirer profit de l'efficacité des communications locales ou régulières plus performantes sur certaines machines que les communications générales.

L'algorithme *Râteau* permet un équilibrage quasi parfait en ne s'autorisant que des communications de voisinage. Il servira de base de comparaison commune pour les autres algorithmes. Une diminution des communications de l'algorithme *Râteau* est proposée dans la technique du *Glissement Pré-calculé*. L'algorithme Voisinage est une technique distribuée *purement* locale ; une stratégie étant considérée comme *purement* locale si les échanges ne s'effectuent qu'entre des PEs possédant un lien physique direct. Afin d'étendre la notion de voisinage et de palier à certaines anomalies rencontrées dans les expériences menées, nous avons transformé la notion de communication de voisinage en une notion de communication uniforme. Les communications pouvant alors s'effectuer à une distance quelconque dans une même direction. Les communications uniformes permettent de conserver un débit d'échange assez élevé et d'étendre la notion de voisinage. Cette modification a donné naissance à l'algorithme *X-Voisinage*. Nous introduisons ensuite les algorithmes systématiques *Pavage* et *X-Pavage* où les transferts de données sont fixés et réalisés dans un ordre donné. L'algorithme *X-Pavage* est une modification de la technique *Pavage* permettant l'utilisation de communications uniformes.

3.2.1 L'algorithme Râteau

L'algorithme *Râteau* propose une méthode simple pour réaliser un équilibrage dans une topologie de dimension quelconque.

Motivations

Le but de cette stratégie est de redistribuer de manière idéale l'ensemble des données sur l'architecture cible. À la fin de la phase d'équilibrage, le poids de chaque processeur élémentaire sera proche de N/n avec une différence de charge instantanée au maximum égale à la dimension sur laquelle l'algorithme s'exécute. Le sur-coût provoqué par l'équilibrage est élevé. Il dépend directement de la « taille » de chaque dimension. Il nous servira de base pour la comparaison des autres algorithmes : l'overhead de l'algorithme *Râteau* sera une borne supérieure.

Implantation

Nous présentons le pseudo-code de l'algorithme *Râteau* sur un anneau.

Le pseudo-code de l'algorithme est présenté en figure III.10. Durant la phase calculatoire de l'algorithme *Râteau* (étape 1), la charge instantanée totale de l'anneau est calculée en utilisant l'instruction de réduction **sum**. Elle est placée dans la variable scalaire N. Durant la seconde étape, les variables q et r sont calculées telles que $N = n \times q + r$; q est donc la charge moyenne du système, r est le reste de la division entière. La phase de redistribution des données proprement dite peut alors débuter. Chaque PE envoie vers son voisin de droite de façon circulaire tous les points situés « au-dessus » de la valeur moyenne q pendant $n - r$ itérations (étape 3), puis toutes les données se trouvant « au-dessus » de $q + 1$ pendant r itérations (étape 4). La variable *num* est utilisée pour conserver une trace du nombre d'itérations effectuée.

L'algorithme *Râteau* s'exécute en n itérations et a donc un coût en $O(n)$.

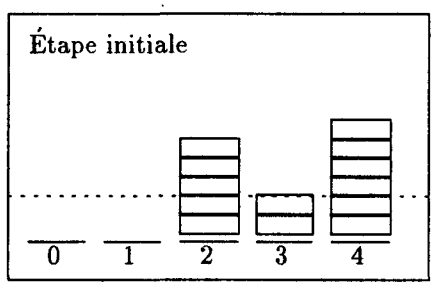
La figure III.11 présente un exemple de déroulement de l'algorithme *Râteau*. Dans ce cas de figure, les paramètres choisis sont $N = 13$, $n = 5$. On a donc $q = N/n = 2$ et $r = 3$. Le comportement du processeur n° 4 est suivi durant les étapes 3 et 4. Durant l'étape 3, le processeur n° 4 émet vers son voisin de droite les 4 données qu'il possède se trouvant au-dessus de la charge moyenne $q = 2$.

Lors de son application sur des systèmes à plusieurs dimensions, l'algorithme *Râteau* est appelé pour chaque dimension en utilisant une généralisation de l'algorithme pour un anneau.

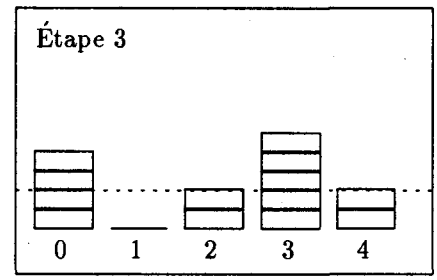
Algorithme *Râteau* pour un anneau
Phase calculatoire
`for all` k `in parallel do`
Étape 1 :
N := `sum`$(w[k])$ réduction, le poids total de l'anneau est calculé
Étape 2 :
$q := N/n$ q est la charge moyenne de l'anneau
$r :=$ N `mod` n `mod` est l'opérateur modulo

Phase de mouvement
$num := n - r$
Étape 3 :
`for` $i := 1$ `to` $n - r$ `do`
`if` $w[k] > q$ `then`
`send`$(k + 1$ `mod` $n, w[k] - q)$ transfert du surplus ($w[k] - q$ données) au voisin de droite pendant $n - r$ itérations
`fi`
`od`
Étape 4 :
`for` $i := 1$ `to` r `do`
`if` $w[k] > q + 1$ `then`
`send`$(k + 1$ `mod` $n, w[k] - q + 1)$ transfert du surplus ($w[k] - q + 1$ données) au voisin de droite durant r itérations
`fi`
`od`
`od`

FIG. III.10 - *Pseudo-code de l'algorithme Râteau pour un anneau*

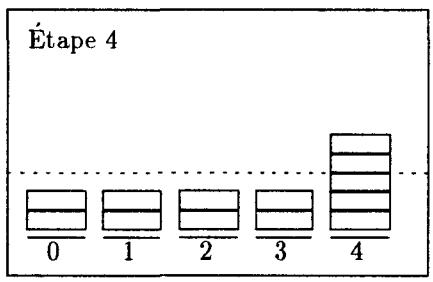


num = 1
$w[4] = 6$
send(0,4)

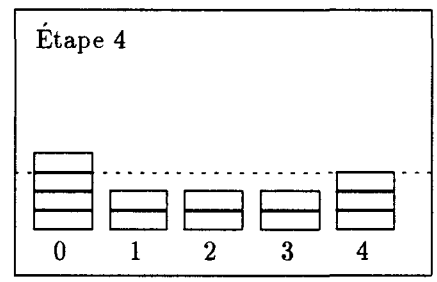


num = 2
$w[4] = 2$

Transfert de 4 données vers le processeur 0

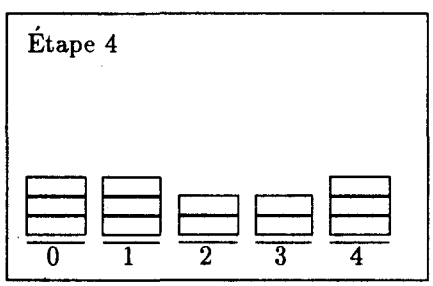


num = 1
$w[4] = 5$
send(0,2)



num = 2
$w[4] = 3$

Transfert de 2 données vers le processeur 0



num = 3
$w[4] = 3$

FIG. III.11 - *Exemple d'exécution de l'algorithme Râteau sur un anneau. On « suit » l'exécution du processeur 4*

Pour une grille, l'algorithme *Râteau* est appliqué simultanément sur l'ensemble des lignes. Un deuxième appel est nécessaire pour réaliser un équilibrage en parallèle sur toutes les colonnes. Ces deux appels à la fonction de rééquilibrage permettent une redistribution des données sur l'ensemble du système. Le même principe peut être appliqué à un cube. L'équilibrage est réalisé simultanément dans une direction orthogonale à une des faces du cube. Trois appels à la routine d'équilibrage sont nécessaires.

Discussion

Dans le chapitre IV consacré à l'analyse des différentes stratégies, nous montrons qu'à la fin d'une unique phase de redistribution, pour un anneau :

$$\forall k,\ q \leq w[k] \leq q+1$$

Pour une topologie de dimension α, après l'applications de l'algorithme *Râteau* le long de α dimensions, on a:

$$\forall k,\ q \leq w[k] \leq q+\alpha$$

Ces formules démontrent la puissance de l'algorithme puisqu'il autorise un équilibrage quasi parfait, tout en n'utilisant que des communications de type local. Comme on peut le voir sur un tel exemple, une redistribution dynamique de type SIMD profite énormément du synchronisme, permettant ainsi un échange simultané de données sur l'ensemble des PEs.

Un nombre conséquent d'algorithmes data-parallèles en particulier dans le domaine du traitement d'images nécessitent la préservation de la localité dans leur exécution. Les données doivent pouvoir accéder aux valeurs des données voisines dans le traitement. Si la charge des processeurs est déséquilibrée, un réarrangement des données peut être réalisé. Cette redistribution doit cependant se faire avec précaution pour garantir le principe de la localité. Si la gestion de la charge locale à chaque processeur est de type « FIFO », l'algorithme *Râteau* réalise un équilibrage où la localité est maintenue en une dimension. La figure III.12 présente un tel rééquilibrage, les données « voisines » sont conservées sur le même processeur ou sur des processeurs adjacents.

Le principal inconvénient de l'algorithme *Râteau* est sa durée. n itérations sont réalisées même si la charge totale est déjà équitablement répartie sur une partie des processeurs. Nous proposons une amélioration de l'algorithme *Râteau* sur une dimension appelée algorithme du *Glissement Pré-calculé*. Elle réalise un équilibrage de la même qualité que la technique *Râteau* avec un nombre moindre de communications.

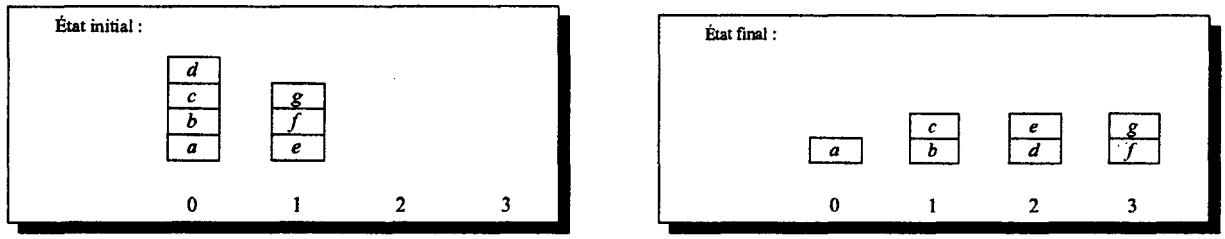


FIG. III.12 - *Exemple de préservation de la localité sur un anneau de 4 processeurs*

3.2.2 L'algorithme du Glissement Pré-calculé

Motivations

L'algorithme du *Glissement Pré-calculé* permet sur une topologie mono-dimensionnelle de réaliser un équilibrage parfait en effectuant un nombre minimum de communications et de transferts. Il se décompose en deux étapes :

- la première partie effectue un pré-calcul des communications qui seront réalisées ;
- la seconde partie réalise les transferts calculés.

Il s'inspire de l'algorithme *Râteau* en ne conservant que les communications utiles. Sur une chaîne de processeurs, chaque PE connaît le nombre d'objets qu'il va envoyer à ses voisins. À la différence de la technique *Râteau*, l'émission de données peut s'effectuer dans les deux directions.

Implantation

Le pseudo-code de l'algorithme du *Glissement Pré-calculé* pour une chaîne de processeurs est présenté à la figure III.13. Un exemple d'exécution est présenté à la figure III.14.

Dans l'algorithme du *Glissement Pré-calculé*, une phase de mouvement vers la gauche (respectivement vers la droite) signifie une phase de communications vers les processeurs d'index décroissant (respectivement croissant).

Pendant la phase d'initialisation, la charge moyenne du système est calculée et placée dans la variable scalaire $q = N/n$. La variable parallèle *somme*[ ] pour le processeur i_0 contient la sommation des charges de tous les processeurs dont l'index est inférieur ou égal à i_0 : $\sum_{i \leq i_0} w[i]$. La variable parallèle *but*[ ] est affectée à la charge moyenne multipliée par l'index du PE. Cela correspond à la sommation de la charge des PEs, si chaque processeur

élémentaire contenait la charge moyenne. La variable *transfert* contient la différence entre *but*[ ] et *somme*[ ]. Elle correspond aux transferts *à réaliser* pour atteindre l'équilibre.

Une valeur positive de la variable *transfert*[ ] indique un mouvement des données vers les numéros de processeurs décroissants. À l'inverse, une valeur négative correspond à une surcharge des processeurs situés à gauche et donc à un mouvement vers la droite. La variable parallèle *possible*[ ] donne pour chaque PE la charge qu'il est *possible* de transférer (minimum entre le nombre de données offert par son voisin et la charge souhaitée).

Le but de la seconde étape est de compléter la charge des processeurs situés à gauche jusqu'à l'équilibre. On provoque un vaste mouvement de données vers la gauche afin de combler le déficit cumulé des PEs situés vers la « gauche ». Les valeurs positives de la variable *transfert*[ ] indiquent le nombre de données qu'il est nécessaire de fournir à gauche pour atteindre l'équilibre (égalité des variables *somme*[ ] et *but*[ ]).

On réitère le processus jusqu'au moment où il n'y a plus de valeurs strictement positives dans la variable *transfert*[ ]. On peut alors passer à la troisième étape qui est la phase symétrique de la deuxième étape pour les transferts vers les numéros croissants de processeurs (mouvement vers la droite).

Discussion

La version présentée de l'algorithme de *Glissement Pré-calculé* s'exécute sur une architecture à une dimension. Il est possible d'étendre l'algorithme du *Glissement Pré-calculé* dans une topologie à plusieurs dimensions comme pour l'algorithme *Râteau*

Il est cependant possible de simuler une vue linéaire des processeurs élémentaires sur un système à plusieurs dimensions. Dans le cas d'une machine à grille torique comme la MasPar MP-1, cette opération peut réalisée avec seulement deux opérations de communications. Il suffit de réaliser une opération de décalage dans chaque ligne et pour les colonnes à l'extrême gauche de fournir leur valeur vers le sud. Un exemple est présenté à la figure III.15. La simulation des opérations (1) et (2) permet de transformer une grille torique en une chaîne de processeurs.

Dans le cas le plus défavorable, où un processeur contient la totalité de la charge du système, le nombre de communications effectuées est le même que pour l'algorithme *Râteau*. Dans le cas général, le nombre de communications est plus faible.

```
for all k in parallel do
Phase d'initialisation :
    N :=sum(w[k])
    q := N/n
    somme[k] :=scanAdd(w[k])
    but[k] := (k + 1) × q
    transfert[k] := but[k] − somme[k]
Phase de transferts vers la gauche :
    if transfert[k] > 0 then
        while transfert[k] ≠ 0 do
            possible[k] := min(transfert[k], w[k + 1])
            if possible[k] > 0 then
                receive(k + 1, possible[k])
                transfert[k] := transfert[k] − possible[k]
            fi
        od
    fi
Phase de transferts vers la droite :
    if transfert[k] < 0 then
        while transfert[k] ≠ 0 do
            possible[k] := min(|transfert[k]|, w[k])
            if possible[k] > 0 then
                send(k + 1, possible[k])
                transfert[k] := transfert[k] + possible[k]
            fi
        od
    fi
od
```

FIG. III.13 - *Pseudo-code de l'algorithme Glissement Pré-calculé sur une ligne de processeurs*

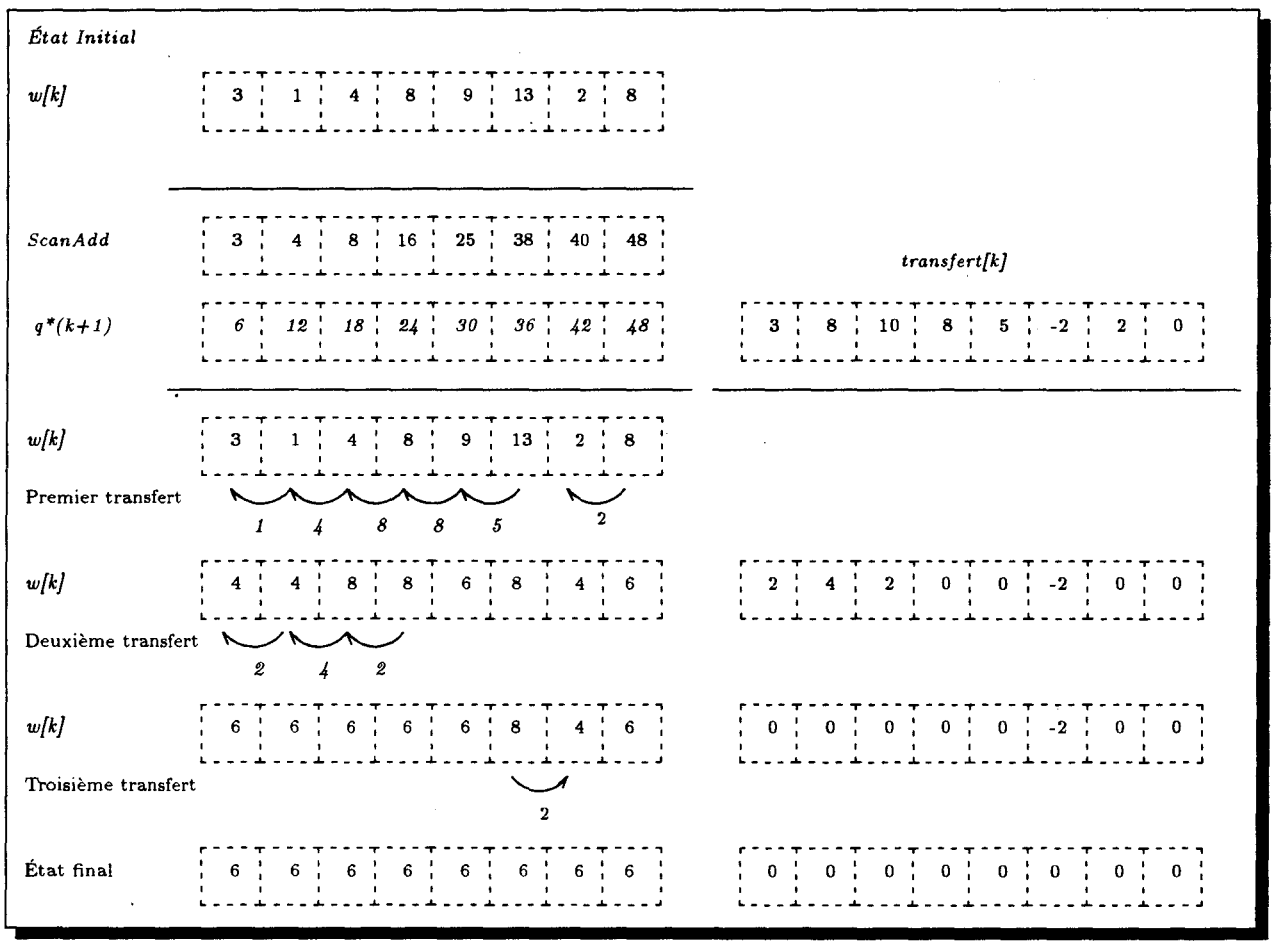


FIG. III.14 - *Présentation de l'algorithme du Glissement Pré-calculé. La partie gauche suit l'évolution de la charge des PEs, tandis que la droite reprend la valeur de la variable* $transfert[k]$

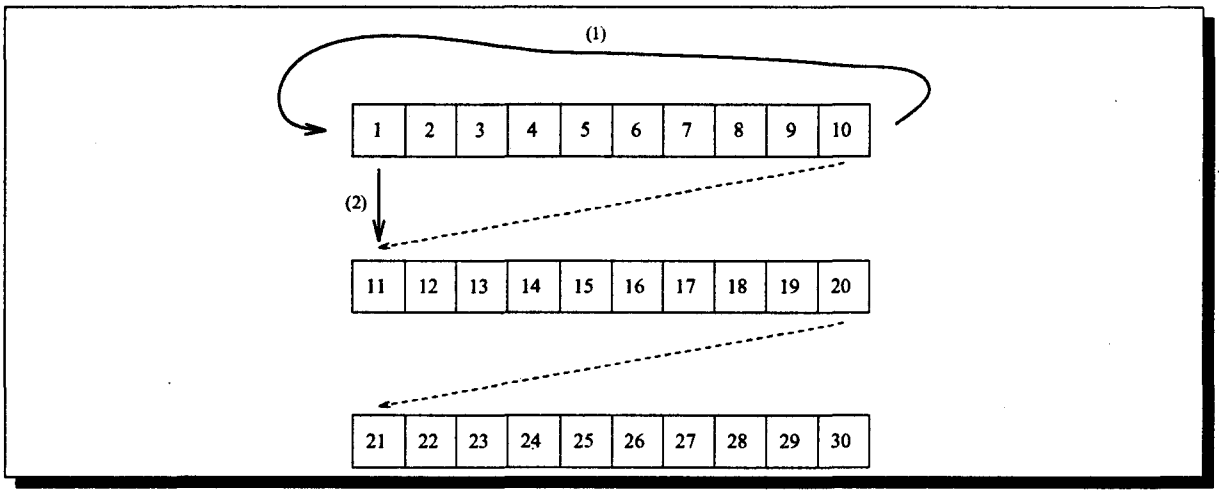


FIG. III.15 - *Projection d'une ligne de PEs sur une grille torique*

3.2.3 L'algorithme Voisinage

Motivations

La stratégie de l'algorithme *Voisinage* est une adaptation des différents algorithmes d'équilibrage utilisés sur des systèmes MIMD [WLR93, Sal90]. L'architecture cible est découpée en une multitude de sous-domaines, chaque domaine étant appelé *îlot* de redistribution. Le but de cette politique de redistribution des données est de réussir à égaliser la charge de tous les nœuds à l'intérieur de chaque îlot. Le domaine de redistribution est centré autour d'un PE; il est constitué d'un processeur et de l'ensemble de ses PEs directement voisins. Pour diffuser la charge sur le système, les îlots se recouvrent en partie et chaque PE appartient à plusieurs domaines de redistribution. Le processeur élémentaire constituant le centre du domaine est appelé PE « centre ». Chaque PE en excès à l'intérieur de chaque îlot tentant d'apporter une partie de sa charge aux PEs « centres » déficitaires. Le but de la technique *Voisinage* est de combler la charge des PEs « centres ».

Par exemple, dans le cas d'une topologie 2D comme une grille, chaque processeur ainsi que ses 4 voisins (Nord, Sud, Est et Ouest) forment un îlot de redistribution. Chaque PE appartient donc à 5 domaines, un dont il est le centre et quatre autres dont il occupe une position de voisins (*cf.* figure III.16). Le nombre d'îlots de redistribution est égal au nombre de processeurs élémentaires.

Dans le modèle MIMD, l'algorithme *Voisinage* s'exécute de façon asynchrone. Chaque processeur suivant ses propres critères décide de transférer une partie de sa charge vers ses voisins. Dans une architecture synchrone, chaque processeur calcule indépendamment des autres la quantité de données qu'il va envoyer. Les transferts s'effectuent simultanément dans la même

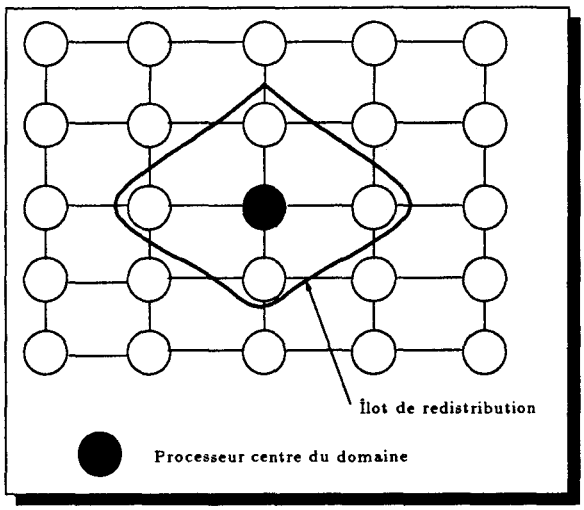


FIG. III.16 - *Algorithme Voisinage sur une architecture 2D*

direction pour tous les PEs permettant ainsi l'utilisation d'un schéma de communications de voisinage.

Implantation

La figure III.17 présente le pseudo-code de l'algorithme *Voisinage* pour un réseau de voisinage à 4 voisins.

La 1re étape calcule la charge moyenne de chaque îlot. Comme la machine appartient au monde SIMD, chaque voisin fournit son poids au PE « centre ». Ces valeurs sont ensuite rangées dans la tableau parallèle *charge_voisin*. La taille de ce tableau est proportionnelle au nombre de voisins d'un PE « centre ». Le calcul de la valeur moyenne de chaque domaine est effectué simultanément dans tous les îlots du système. Cette valeur est conservée dans la variable parallèle *moy*[ ]. Le but de la seconde étape est d'estimer le nombre de données en surplus dans chaque domaine. Les valeurs de la variable *charge_voisin* indiquent le nombre de données surnuméraires qu'il possède. Le nombre de données en excès est défini comme le nombre d'objets supérieur à la charge moyenne de l'îlot. La variable *exces*[ ] gardera la somme du surplus de chaque voisin par rapport à la charge moyenne du domaine. La 3e étape réalise les transferts de données, chaque voisin du processeur élémentaire lui fournissant une portion de son travail. La charge reçue par chaque PE « centre » de la part d'un PE voisin est donnée par la formule suivante :

$$\forall j \in [1,4] \qquad \frac{moy[k,l] \times (charge\_voisin[k,l][j] - moy[k,l])}{exces[k,l]}$$

```
for all k,l in parallel do
Phase d'initialisation :
    var charge_voisin[ ] : array[1..4] of integer
    moy[k,l] := 0
    exces[k,l] := 0
    for j := 1 to 4 do
Étape 1 :
        charge_voisin[k,l][j] := w[mon_voisin(j,1)]
        moy[k,l] := moy[k,l] + charge_voisin[k,l][j]
    od
    moy[k,l] := (moy[k,l] + w[k,l])/5
    for j := 1 to 4 do
Étape 2 :
        if charge_voisin[k,l][j] > moy[k,l] then
            exces[k,l] := exces[k,l] + charge_voisin[k,l][j] - w[k,l]
        fi
    od
Phase de mouvement
    for j := 1 to 4 do
        if charge_voisin[k,l][j] > moy[k,l] then
Étape 3 :
            p[k,l] := charge_voisin[k,l][j] - moy[k,l]
            envoi[k,l] := moy[k,l] × p[k,l]/exces[k,l]
            receive(mon_voisin(j,1), envoi[k,l])
        fi
    od
od
```

FIG. III.17 - *Pseudo-code de l'algorithme Voisinage en 2 dimensions*

où les variables $moy[\,]$, $charge\_voisin[\,][j]$, $exces[\,]$ désignent respectivement la charge moyenne de l'îlot, la charge du voisin j et le surplus total d'objets du domaine. La différence $charge\_voisin[\,][j] - moy[\,]$) représente le nombre d'objets en surplus du voisin j par rapport à ce domaine. L'emploi d'une telle formule est justifiée par l'appartenance d'un PE à plusieurs domaines de rééquilibrage. Le processeur élémentaire doit obéir à deux contraintes :

1. Il ne doit pas transférer une charge supérieure à celle qu'il possède.
2. Il ne doit pas non plus se transformer en PE déficitaire après avoir cédé une partie de son travail.

Examinons le cas le plus défavorable où un PE i_0 est entouré de PEs possédant une charge nulle jusqu'à une distance de 2 (cf. figure III.18). Il cédera sa charge à quatre domaines différents.

La charge moyenne des quatre domaines dont le PE i_0 est un voisin est égale à :

$$moy_1 = moy_2 = moy_3 = moy_4 = w[i_0]/5$$

De même, la surcharge de chaque domaine est constituée par une partie de la charge du PE i_0 :

$$exces_1 = exces_2 = exces_3 = exces_4 = \frac{4}{5} \times w[i_0]$$

Le processeur i_0 cédera dans chaque domaine un nombre de données égale à :

$$\frac{moy \times (w[i_0] - moy)}{exces} = \frac{w/5 \times (w - w/5)}{4/5 \times w} = w/5$$

Le PE i_0 cédera $1/5^e$ de sa charge à chacun des 4 domaines dont il est un voisin. Les deux critères sont vérifiés :

1. Il n'a pas cédé plus de données qu'il n'en possédait.
2. Son poids actuel correspond au $1/5^e$ de sa charge précédente, il ne se retrouve pas en déficit par rapport aux PEs « centres » des domaines voisins.

Discussion

La politique de sélection de l'algorithme *Voisinage* s'inspire des politiques de seuil proposées dans les techniques MIMD. L'appariement réalisé est de type distribué par recherche.

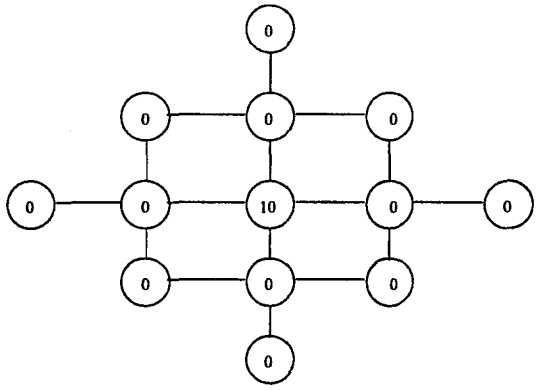


FIG. III.18 - *Étape initiale de l'algorithme Voisinage*

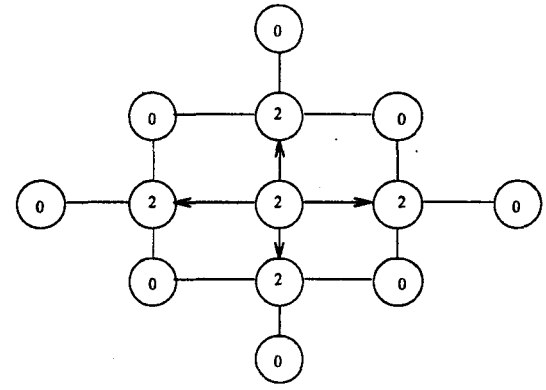


FIG. III.19 - *Étape finale de l'équilibrage Voisinage*

Les PEs ont connaissance de la charge de leurs voisins et avec ces informations comblent les déficits de charge.

L'algorithme *Voisinage* réalise un équilibrage par diffusion en conservant les communications locales. L'algorithme ne nécessite pas de connaissance globale, mais simplement la connaissance des poids des différents processeurs d'un îlot de redistribution. L'équilibrage permet une diffusion des données des zones les plus chargées vers les moins chargées. L'implantation réalisée garantit qu'un processeur détectant un déséquilibre ne créera pas, par le travail transféré un déséquilibre dans le sens inverse.

3.2.4 L'algorithme X-Voisinage

Motivations

Expérimentalement, comme Lüling et *al* [LMR91], nous avons rencontré des anomalies d'exécution dues au parallélisme massif des machines utilisées. De nombreux problèmes sont intrinsèquement déséquilibrés et provoquent une distribution irrégulière répartie de manière uniforme. C'est-à-dire que l'on se trouve confronté à des situations représentées par la figure III.20, où se forment des zones de processeurs surchargés et des zones de processeurs peu occupés.

Lorsque l'équilibrage de charge est effectué avec une stratégie purement locale sur de tels problèmes, on constate que la redistribution des données n'est réalisée qu'aux frontières des différentes zones. Les stratégies par diffusion locale comme l'algorithme précédent sont mal adaptées à de telles situations. La diffusion des données des zones à forte densité de charge vers les surfaces à densité plus faible est lente. De ce fait, le nombre d'appel aux routines d'équilibrage augmente en forte proportion.

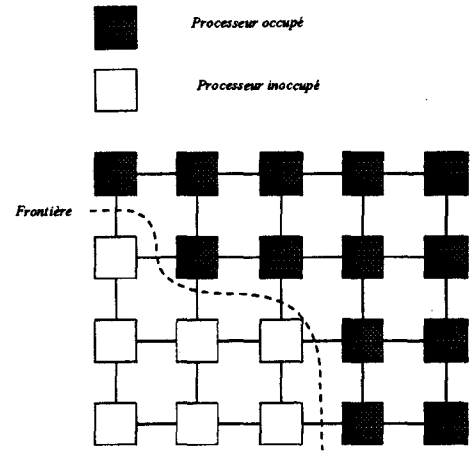


FIG. III.20 - *Exemple de charge irrégulière*

Une solution envisageable pour palier à cet inconvénient est de ne plus limiter un îlot de redistribution au processeurs et ses voisins directs, mais de permettre une évolution dynamique des domaines de redistribution. Il devient nécessaire d'adapter la stratégie d'équilibrage afin de tenir compte de ces modifications. Pour ne pas perdre la vitesse avantageuse offerte par les communications locales, nous utiliserons des communications *uniformes*. Les échanges d'informations s'effectueront non plus entre voisins directs, mais entre processeurs situés à une même distance plus importante mais *dans une même direction*. L'uniformité et la simultanéité des communications garantissent une vitesse de transfert relativement élevée et les performances seront par conséquent meilleures qu'un schéma de communications globales. Afin de mettre en œuvre une telle politique, nous définissons la fonction « chemin » qui indique l'évolution du système. Elle associe à chaque « phase » d'équilibrage, la distance séparant les processeurs appartenant au même domaine de redistribution. La fonction « chemin » ($ch()$) est une fonction de $\mathbb{N}$ dans $\mathbb{N}$ qui associe à chaque équilibrage la taille de l'îlot.

Pour le premier appel à la routine de redistribution, la fonction chemin prendra la valeur 1, le domaine de redistribution sera identique à celui défini dans l'algorithme *Voisinage*, le 1 indiquant que les PEs concernés sont situés à une distance de 1. De même, pour le i^e appel à la routine d'équilibrage, l'îlot de redistribution sera constitué du processeur « centre » ainsi que de ses « voisins » situés à une distance égale à $ch(i)$ dans les directions uniformes. On se trouve donc face à des domaines constitués du même nombre d'éléments mais offrant une dispersion plus grande dans le système. La figure III.21 présente deux domaines de redistribution centrés autour d'un même processeur pour une topologie 2D, les PEs possédant le numéro 1 appartiennent au premier domaine, tandis que les processeurs numérotés 2 font partie du deuxième îlot de redistribution. Le désavantage présenté par l'utilisation de $ch(i)$ est que cette fonction est très sensible à son paramétrage. Nous avons procédé par essai pour trouver des valeurs adéquates de $ch(i)$.

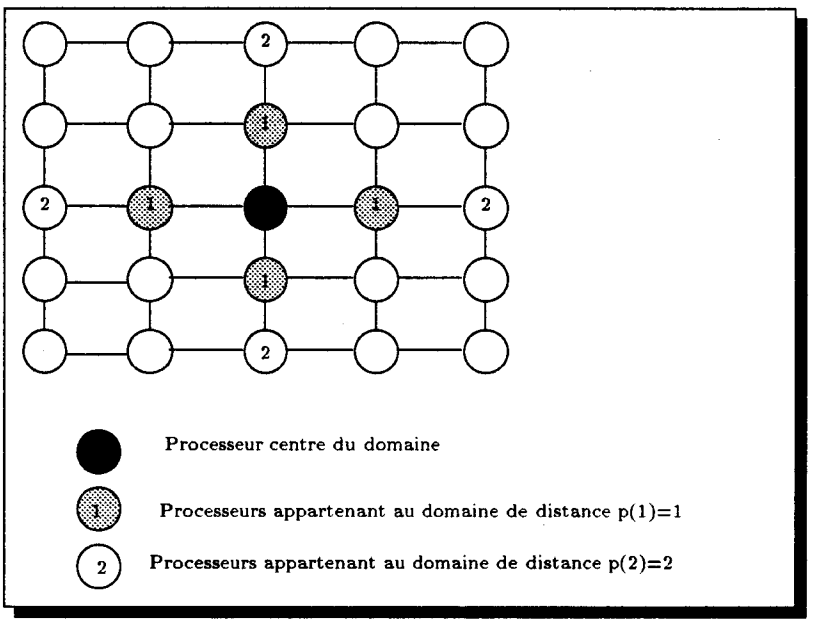


FIG. III.21 - *Présentation de deux domaines de l'algorithme X-Voisinage*

Implantation

Le pseudo-code de l'algorithme *X-Voisinage* se rapproche fortement de celui de l'algorithme *Voisinage*, la principale différence est que la fonction `mon_voisin` est utilisée par des communications uniformes pour accéder à des processeurs distants.

À chaque appel à la routine *X-Voisinage*, le domaine est modifié. Le compte du nombre de phases d'équilibrages est conservé dans la variable scalaire i_{eq}. Cette variable est incrémentée pour chaque nouvel appel à la routine *X-Voisinage*. Elle est utilisée pour le calcul de la fonction $ch(i)$.

Présentation de la fonction chemin

La décision des valeurs à affecter à la fonction chemin est un choix délicat. Une fonction efficace permettra d'améliorer de manière très sensible les performances du système, tandis que des « mauvaises » valeurs provoqueront l'effet inverse. Nous avons testé plusieurs fonctions chemins ; la fonction aléatoire utilisée dans un premier temps pour obtenir une dispersion importante des données dans le système n'a pas donné de bons résultats. Expérimentalement, pour un système à 2 dimensions dont le nombre de processeurs dans la première dimension n_x est égal au nombre de processeurs dans la seconde dimension n_y, la fonction chemin offrant les meilleurs résultats est le cycle suivant :

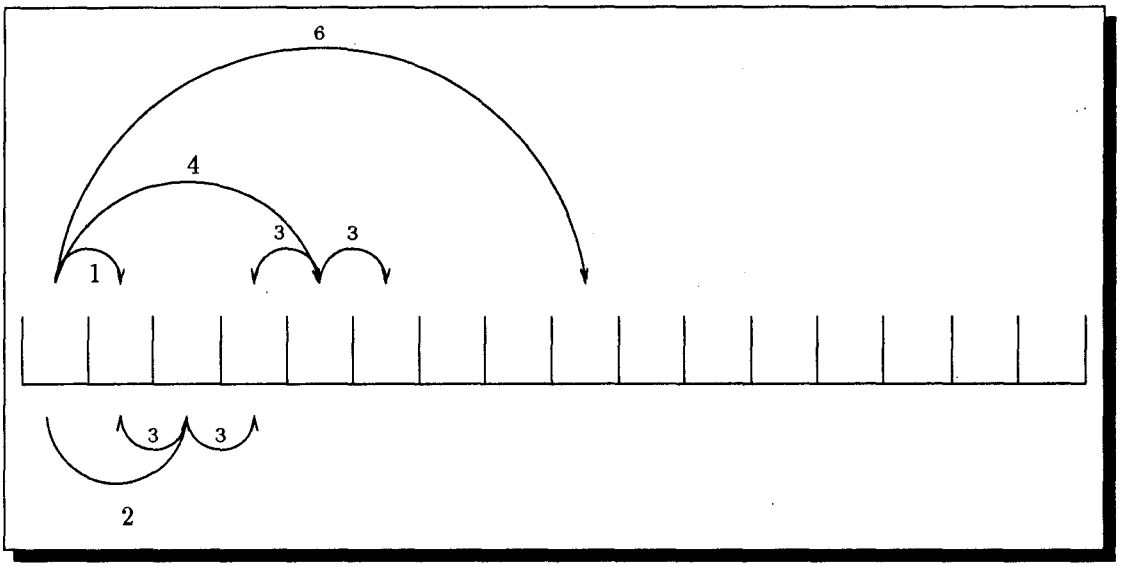


FIG. III.22 - *Action de la fonction chemin sur un ensemble de 16 processeurs. Les numéros au dessus des arcs indiquent le numéro de l'équilibrage*

$$\begin{cases} ch(2i) & = 1 \\ ch(2i+1) & = 2^{i \pmod{\log_2 n_x}+1} \end{cases}$$

Par exemple, sur une grille carrée de taille 32×32, la fonction donnera les valeurs suivantes :

$$ch(i) = \ 1\ ,2\ ,1\ ,4\ ,1\ ,8\ ,1\ ,16\ ,1\ ,\ldots$$

Les itérations paires de la fonction chemin permettent un envoi de points à une distance éloignée, les itérations impaires permettant elles de réaliser une redistribution plus locale. La figure III.22 représente l'action de l'algorithme *X-Voisinage* pour un ensemble monodimensionnel de 16 processeurs élémentaires.

```
for all k,l in parallel do
Phase d'initialisation :
    var charge_voisin[ ] : array[1..4] of integer
    moy[k,l] := 0
    exces[k,l] := 0
    distance := ch(i_eq)
    for j := 1 to 4 do
```

Étape 1 :
$charge\_voisin[k,l][j] := w[\texttt{mon\_voisin}(j, distance)]$
$moy[k,l] := moy[k,l] + charge\_voisin[k,l][j]$
od
$moy[k,l] := (moy[k,l] + w[k,l])/5$
for $j := 1$ to 4 do
Étape 2 :
if $charge\_voisin[k,l][j] > moy[k,l]$ then
$exces[k,l] := exces[k,l] + charge\_voisin[k,l][j] - w[k,l]$
fi
od
Phase de mouvement
for $j := 1$ to 4 do
if $charge\_voisin[k,l][j] > moy[k,l]$ then
Étape 3 :
$p[k,l] := charge\_voisin[k,l][j] - moy[k,l]$
$envoi[k,l] := moy[k,l] \times p[k,l]/exces[k,l]$
receive(mon_voisin$(j, distance), envoi[k,l]$)
fi
od
$i_{eq} := i_{eq} + 1$
od

FIG. III.23 - *Pseudo-code de l'algorithme X-Voisinage*

La figure III.23 présente le pseudo-code de l'algorithme pour un réseau à 4 voisins. La variable n_{eq} est incrémentée à chaque nouvel appel à la phase d'équilibrage. Elle est initialement positionnée à la valeur 1. La variable *distance* prend la valeur de la fonction chemin pour l'équilibrage correspondant. Le pseudo-code de l'algorithme *X-Voisinage* est semblable à celui de la technique *Voisinage*. Seuls les domaines sont évolutifs et leur position change à chaque nouvel appel.

Discussion

L'algorithme *X-Voisinage* réalise un équilibrage par diffusion comme la technique *Voisinage*. L'introduction d'une fonction chemin permet « d'accélérer » la diffusion des données à travers le système et d'éviter les problèmes de concentration de charge. Les schémas de communications réguliers produits par *X-Voisinage* permettent de manière générale une plus grande vitesse de transfert que les schémas de communications globales.

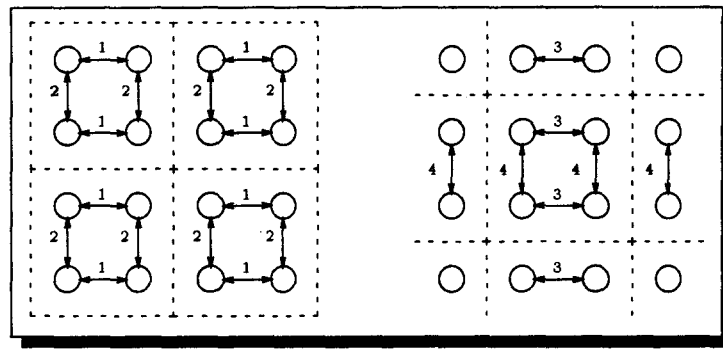


FIG. III.24 - *Exécution de l'algorithme Pavage*

3.2.5 L'algorithme Pavage

Motivations

La construction de l'algorithme *Pavage* est caractérisée par un équilibrage parfait sur de petits domaines disjoints. Ces domaines sont ensuite déplacés et combinés pour propager l'équilibrage.

Dans la présentation de l'algorithme *Pavage*, nous nous limiterons à son étude sur une topologie à une et deux dimensions. Dans ce cas, le domaine d'équilibrage sera appelé « fenêtre » de redistribution. Chaque fenêtre comportera respectivement deux et quatre processeurs élémentaires. L'exécution de l'algorithme *Pavage* se décompose en deux principales phases comme indiqué sur la figure III.24 pour une architecture 2D. Durant la première partie de la redistribution, un équilibrage parfait est réalisé à l'intérieur de chaque fenêtre simultanément. La fenêtre est ensuite légèrement déplacée afin d'être combinée avec trois autres domaines en 2D et un en 1D.

Implantation

Les communications induites par la technique *Pavage* sont limitées à un stricte voisinage. L'algorithme peut être considéré comme étant *purement* local.

Afin de faciliter la description de l'algorithme, le pseudo-code présenté s'applique à un anneau de processeurs. La fenêtre de rééquilibrage est constituée d'un couple de processeurs. L'algorithme se décompose en deux phases principales, une pour l'équilibrage dans un premier domaine et le second pour l'équilibrage après mouvement de la fenêtre de redistribution (*cf.* figure III.25).

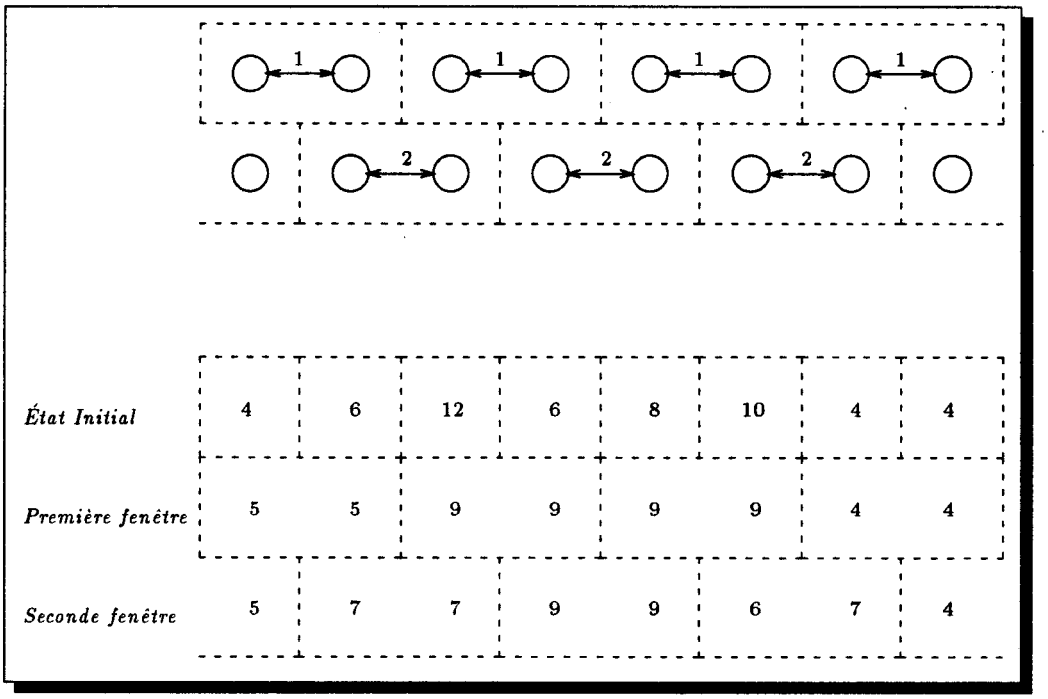


FIG. III.25 - *Algorithme Pavage pour un anneau*

Le but de chaque étape est de réaliser un équilibrage parfait à l'intérieur de chaque domaine de redistribution créé. Durant la première étape, chaque PE possédant un numéro d'identification paire récupère la charge de son voisin situé à droite afin de la cumuler à son propre poids. Cette somme est divisée par deux et comparée à la charge locale. S'il se trouve en excès, le surplus de poids est transmis à son voisin, permettant ainsi d'équilibrer le couple. Si un déficit existe, la charge est récupérée sur son voisin appartenant à la même fenêtre de redistribution. À la fin de cette première étape, chaque fenêtre de deux processeurs est parfaitement équilibrée, il suffit alors de déplacer cette fenêtre et de réitérer ce principe d'équilibrage par couples. Ces opérations sont réalisées par l'étape 2.

Discussion

L'algorithme *Pavage* présente un certain nombre de différences par rapport aux autres techniques étudiées dans ce chapitre :

- la politique de sélection est systématique. Les processeurs participent à l'équilibrage quelque soit leur charge ;
- la localité des communications est toujours conservée. Seuls les processeurs possédant

```
for all k in parallel do
Travail sur la première fenêtre
Étape 1 :
   if ((k mod 2) = 0) then
      moyenne[k] := w[k] + w[k + 1]
      moyenne[k] := moyenne[k]/2
      if w[k] > moyenne[k] then
         send(k + 1, w[k] − moyenne[k])
      else
         receive(k + 1, moyenne[k] − w[k])
      fi
   fi
Travail sur la seconde fenêtre
Étape 2 :
   if ((k mod 2) = 1) then
      moyenne[k] := w[k] + w[k − 1]
      moyenne[k] := moyenne[k]/2
      if w[k] > moyenne[k] then
         sendk − 1, w[k] − moyenne[k])
      else
         receive(k − 1, moyenne[k] − w[k])
      fi
   fi
od
```

FIG. III.26 - *Pseudo-code de l'algorithme Pavage en une dimension*

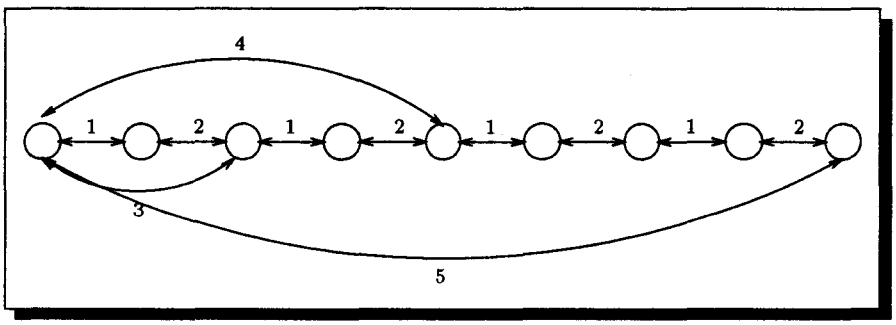


FIG. III.27 - *Représentation de l'algorithme X-Pavage*

un lien physique entre eux peuvent s'échanger une partie de leur charge;

- à la différence de l'algorithme *Voisinage*, un PE à un instant donné n'appartient qu'à une seule fenêtre de redistribution, ce qui permet la réalisation d'un équilibrage parfait à l'intérieur de celle-ci. L'algorithme *Voisinage* ne peut garantir cet état de fait, puisque les PEs appartiennent à des îlots qui se recouvrent en partie.

3.2.6 L'algorithme X-Pavage

Motivations

L'algorithme *Pavage* s'adapte mal à la résolution de problèmes uniformément déséquilibrés. L'algorithme *X-Pavage* en propose une amélioration, en permettant de s'affranchir des communications purement locales. Les domaines de redistribution sont conservés mais les processeurs d'un domaine sont éparpillés à travers le système. L'approche présentée est différente de celle de l'algorithme *X-Voisinage* car une phase complète d'équilibrage comprend un équilibrage à l'intérieur de tous les domaines. La figure III.27 montre l'application de ce principe.

Dans les deux premières phases, les domaines reliés par des liens numérotés 1 et 2, correspondent à la technique *Pavage*. Les étapes suivantes étendent la notion de domaine élémentaire. Les fenêtres de redistribution sont espacées de $2, 4, 8 \ldots$ PEs.

Implantation

Le pseudo-code de l'algorithme *X-Pavage* présenté sur la figure III.28 se décompose en deux parties. La première phase fait appel à la routine de distribution de l'algorithme *Pavage* permettant ainsi de réaliser un équilibrage sur les domaines contigus de longueur 1. La fonction

```
Routine domaine(d) réalisant l'équilibrage par couples à une distance d
for all k in parallel do
if (((k div d) mod 2) = 0) then
    moyenne[k] := w[k] + w[k + d]
    moyenne[k] := moyenne[k]/2
    if moyenne[k] > w[k] then
        send(k + d, w[k] − moyenne[k])
    else
        receive(k + d, moyenne[k] − w[k])
    fi
fi
```

```
Algorithme X-Pavage
for all k in parallel do
    Pavage()
    i := 2
    while (i ≠ n) do
        domaine(i)
        i := i × 2
    od
od
```

FIG. III.28 - *Pseudo-code de l'algorithme X-Pavage en une dimension*

domaine() est ensuite appelée réalisant un équilibrage sur des domaines élémentaires distants. Le paramètre communiqué à la routine **domaine()** représente la « longueur » du domaine élémentaire. Nous avons choisi d'effectuer la redistribution des données suivant les valeurs : $2, 4, 8, \ldots, n/2$. Comme nous l'examinerons dans le chapitre IV, c'est une série de valeurs permettant l'obtention d'une bonne répartition de données.

Discussion

L'évolution dynamique des domaines d'équilibrage permet d'obtenir une meilleure dispersion des données sur un système. Toutefois, à la différence de l'algorithme *X-Voisinage* dont la fonction chemin est définie de manière intuitive, les valeurs fournies à la fonction do-

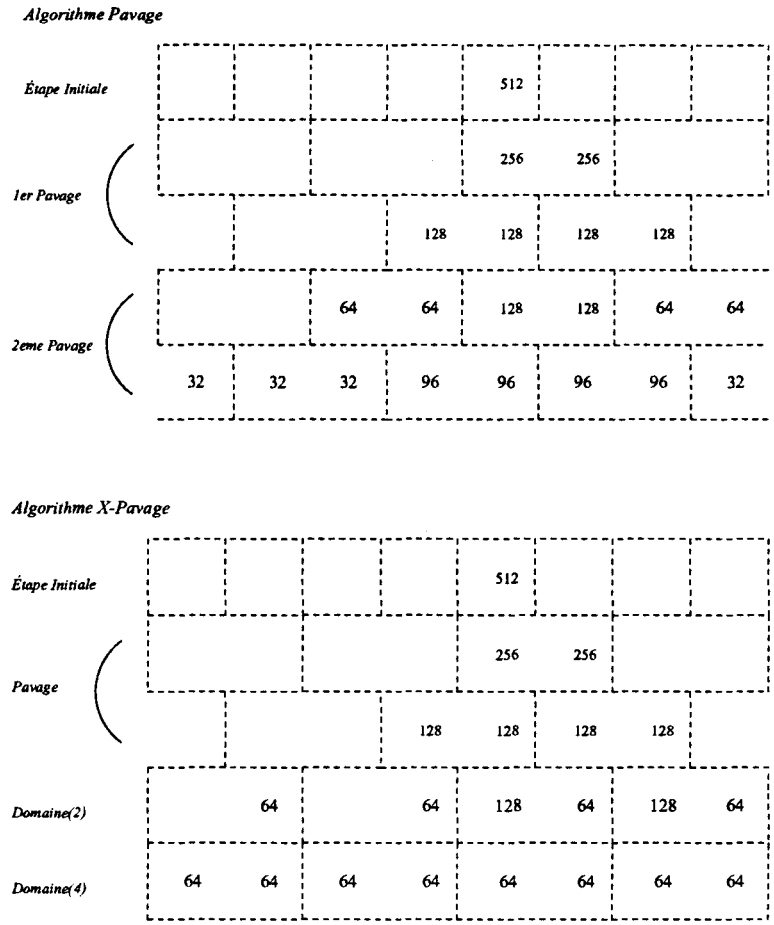


FIG. III.29 - *Comparaison des approches Pavage et X-Pavage pour une chaîne de 8 processeurs*

maine peuvent être appréciées et comparées mathématiquement. Intuitivement, l'algorithme *X-Pavage* semble posséder un meilleur comportement que la technique *Pavage* pour un coût léger en communications. Si la totalité de la charge du système est contenue dans un unique processeur, un nombre conséquent d'appels à l'algorithme *Pavage* seront nécessaires pour lisser la charge. À l'inverse, la technique *X-Pavage* enverra dès les premiers appels une partie de sa charge vers des PEs éloignés. La figure III.29 présente l'évolution conjointe des techniques *Pavage* et *X-Pavage* lorsque la charge initiale est possédée par un unique PE.

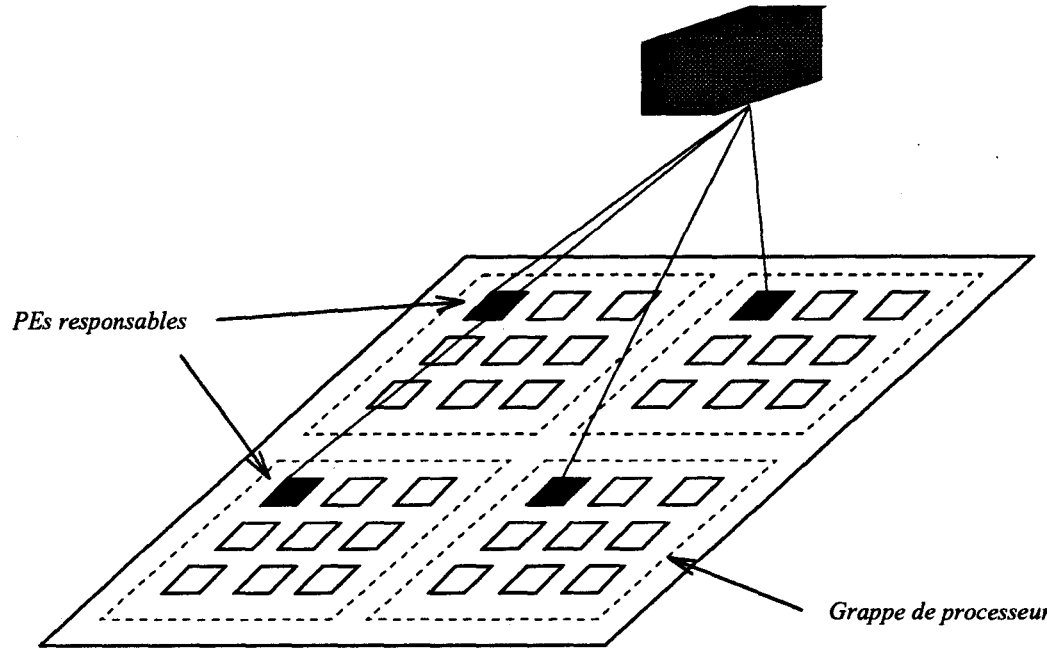


FIG. III.30 - *Exemple de technique Hybride pour une grille composée de 4 grappes*

3.3 Un algorithme hybride

Motivations

Comme dans les méthodes d'équilibrage MIMD, des politiques de domaine mixtes utilisant à la fois des communications locales et globales peuvent être utilisées sur des architectures synchrones. Le but d'une technique mixte ou *Hybride* est de partager le système en grappes de processeurs. Deux niveaux de rééquilibrage sont alors utilisés :

1. Un équilibrage utilisant des communications locales pour réaliser la redistribution des données à l'intérieur des grappes.

2. Un équilibrage à un niveau supérieur permettant l'égalisation des charges entre les grappes de processeurs.

Implantation

La figure III.30 présente le fonctionnement de l'algorithme *Hybride* sur une architecture bi-dimensionnelle.

L'algorithme *Hybride* peut être décomposé en trois étapes :

Remontée des informations La charge globale moyenne peut être rapidement calculée à l'aide des opérations de parallel-prefix. À l'intérieur de chaque grappe en excès, les PEs font « remonter » vers le processeur responsable leur éventuel surcharge. Des communications locales sont utilisées.

Appariement global À l'aide d'un ou plusieurs appels à une technique de réarrangement global comme la technique de *Rendez-vous*, un appariement entre les grappes surchargées et sous-chargées est effectué.

Équilibrage local Un équilibrage en parallèle dans toutes les grappes est accompli à partir des données transférées par l'appariement global.

Discussion

L'algorithme *Hybride* peut être implanté sur les machines SIMD en utilisant le particularisme matériel de ces dernières. Par exemple sur une machine comme la MasPar MP-1, le réseau global est relié par un fil unique à une grappe de 4×4 processeurs appelée cluster. Il est judicieux de choisir les grappes de l'algorithme *Hybride* égales à ces clusters.

On peut remarquer que :

- l'algorithme *Rendez-vous* peut être considéré comme un cas particulier de la technique *Hybride* dans laquelle la taille des grappes est réduite à un processeur ;
- l'utilisation d'une technique *Rendez-vous* sur un ensemble de clusters au lieu de l'ensemble des processeurs permet d'en diminuer le coût. Par exemple, pour une grille torique de processeurs élémentaires, réitérer l'algorithme *Rendez-vous* $O(\log_2(n))$ fois provoque un équilibrage parfait. Pour la technique *Hybride*, le nombre de répétitions de la technique *Rendez-vous* sera en $O(\log_2(\text{nombre de grappes}))$.

3.4 Conclusions sur les algorithmes

Le tableau III.1 présente suivant le modèle d'équilibrage SIMD introduit au paragraphe 1 les différents algorithmes étudiés.

Les méthodes d'équilibrage SIMD présentent une grande diversité. Plusieurs méthodes d'appariement, de sélection et de politiques de communications ont été proposées. L'adéquation de ces différentes techniques à une machine SIMD est fortement dépendantes du système

TAB. III.1 - *Présentation selon le modèle d'équilibrage SIMD des différents algorithmes*

Algorithme	Sélection	Appariement	Domaine/Communication
Central	seuil adaptatif	rendez-vous	globale
Rendez-vous	seuil adaptatif	rendez-vous amélioré	globale
Râteau	seuil	distribuée systématique	voisinage
Glissement Pré-calculé	calculée	distribuée calculée	voisinage
Voisinage	seuil	recherche	voisinage
X-Voisinage	seuil	recherche	uniforme
Pavage	systématique	distribuée systématique	voisinage
X-Pavage	systématique	distribuée systématique	uniforme
Hybride	seuil	hybride	mixte

de communication offert. Sur certaines machines qui ne disposent que d'un réseau de voisinage, les techniques de communications locales seront appliquées. Si des processeurs élémentaires peuvent transmettre des données à d'autres PEs situés à un endroit quelconque pour un coût modique, des approches globales comme celle proposée par l'algorithme *Rendez-vous* seront préférées. Si le réseau de voisinage peut être utilisée pour des communications uniformes peu onéreuses, des techniques comme *X-Pavage* ou *X-Voisinage* pourront être appliquées.

4 Les politiques de déclenchement

Les politiques de déclenchement permettent l'appel aux algorithmes d'équilibrage présentés dans la section 3. La politique de décision constitue une phase capitale de l'équilibrage dynamique sur machines synchrones, car le déclenchement à un moment inopportun ou de façon trop répétée peut réduire à néant le gain offert par la stratégie de redistribution des données.

On peut discerner deux classes de politiques de déclenchement :

les mécanismes périodiques provoquent le rééquilibrage dynamique suivant une fréquence déterminée à l'avance.

les mécanismes adaptatifs utilisent des informations collectées lors des précédentes phases d'équilibrage ou des informations estimées sur le comportement futur du système afin de provoquer ou non une phase d'équilibrage de charge. Si le mécanisme provoque un équilibrage d'après les informations qu'il possède, on parle de mécanisme prédictif *instantané*. S'il tente d'estimer d'après les diverses informations obtenues si l'équilibrage serait profitable, on parle alors de mécanisme prédictif *adaptatif*.

4.1 Notations utilisées

Cette section introduit les différentes notations qui seront utilisées dans la description des politiques de décision. Comme d'habitude, n représente le nombre total de processeurs du système. Le nombre de processeurs actifs au temps t est décrit par une variable discrète $A(t)$.

Définition 8 *Le ratio d'activité $x(t)$ d'un système SIMD est défini comme le rapport du nombre de processeurs actifs sur le nombre total de processeurs.*

$$x(t) = \frac{A(t)}{n}$$

Le nombre d'étapes de calcul ou d'itérations effectuées par l'ensemble des PEs actifs depuis le dernier rééquilibrage est équivalent à un travail.

Définition 9 *Le travail d'un algorithme SIMD depuis la dernière phase d'équilibrage correspond à la quantité totale d'étapes de calcul réalisées par l'ensemble des PEs* actifs. *Le travail est mesuré en PEs actifs×unités de temps.*

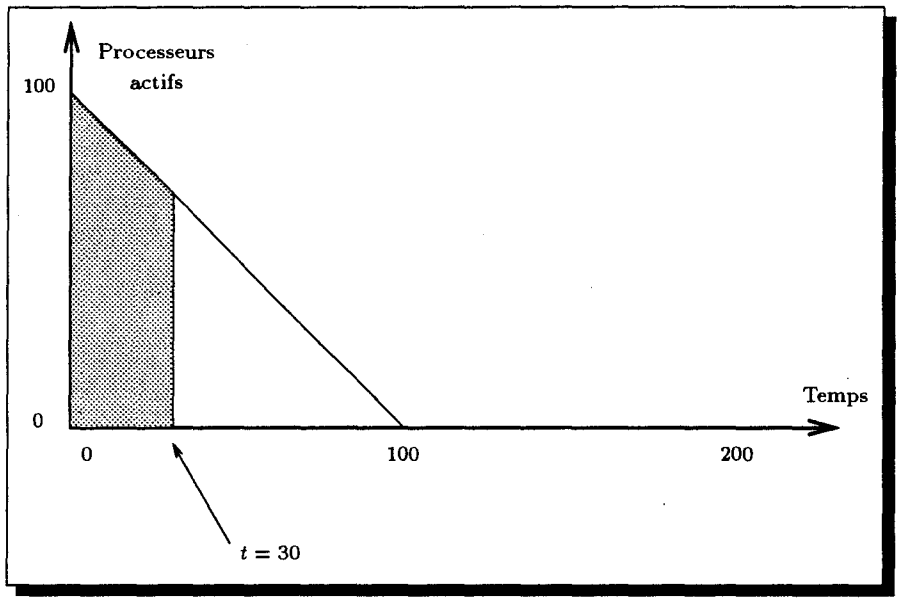


FIG. III.31 - *Exemple d'activité des PEs en fonction du temps*

$$W(t) = \sum_{i=de(t)}^{t} A(t_i)(t_i - t_{i-1})$$

$de(t)$ correspond au moment du déclenchement du dernier rééquilibrage. Le travail est calculé par discrétisation en utilisant une méthode numérique classique comme la méthode de Simpson.

Une notion importante qui sera développée dans les mécanismes adaptatifs prédictifs est l'équivalent PEs au temps t $\sim PE$.

Définition 10 *L'équivalent PEs au temps t $\sim PE$ correspond au nombre de processeurs élémentaires employés à 100% qui auraient été nécessaires depuis le dernier rééquilibrage pour effectuer le même travail.*

$$\sim PE = \frac{W(t)}{t - de(t)}$$

L'équivalent PEs permet de connaître l'utilisation du système.

TAB. III.2 - *Évolution de la fonction $\sim PE(t)$*

Temps	10	20	30	40	50	60	70	80	90	100
$\sim PE$	95	90	85	80	75	70	65	60	55	50

Calculons par exemple, l'équivalent PEs de la figure III.31 pour $t = 30$s. On a $W(30) = 2550$ PEs actifs×s. Ce qui donne $\sim PE(30) = 85$. Le tableau III.2 donne quelques valeurs de l'équivalent PEs pour la courbe de la figure III.31.

4.2 Les politiques périodiques

Lorsque le coût de l'équilibrage de charge est faible et constant, il est possible d'utiliser une politique périodique. Dans un tel cas, la redistribution des données est effectuée régulièrement, soit toutes les T secondes ou encore toutes les T itérations. Cette approche présente le désavantage de nécessiter un paramétrage très sensible.

Une politique périodique et adaptative

Le problème rencontré par la politique périodique est de ne pas autoriser une adaptation du déclencheur aux évolutions du système. Nous proposons de substituer ce déclencheur par un mécanisme modifiant dynamiquement la période. Le principe est le suivant :

En supposant qu'un seuil statique x_{seuil} constitue expérimentalement une bonne valeur pour un système donné et un nombre de processeurs donné, le rééquilibrage se déclenche initialement toutes les T itérations. La variable δ correspond à la sensibilité du déclencheur. Si le ratio de processeurs actifs x est supérieur à $x_{\text{seuil}} + \delta$ indiquant ainsi un rééquilibrage trop précoce, la nouvelle période sera augmentée en conséquence. Réciproquement, la longueur de la période sera diminuée, si le déclenchement a été trop tardif (*cf.* figure III.32).

Si le seuil de référence choisi est de x_{seuil}, si T est la période d'échantillonage :

$$T = \begin{cases} T & \text{si } x_{\text{seuil}} - \delta \leq x \leq x_{\text{seuil}} + \delta \\ (1+\alpha)T & \text{si } x > x_{\text{seuil}} + \delta \\ (1-\beta)T & \text{si } x < x_{\text{seuil}} - \delta \end{cases}$$

avec $\alpha = x - (x_{\text{seuil}} + \delta)$ et $\beta = (x_{\text{seuil}} - \delta) - x$.

Pour éviter des écarts trop importants dans le calcul de la période et afin de prendre en compte des surcharges instantanées de travail, nous imposons la contrainte suivante sur

TAB. III.3 - *Résumé des principales variables utilisées pour les politiques de déclenchement*

Nom	Signification
n	Nombre de processeurs
$w[k]$	Charge du processeur k
N	Charge totale du système $N = \sum_k w[k]$
L	Durée prévisionnelle d'un équilibrage
$A(t)$	Nombre de processeurs actifs à l'instant t
$x(t)$	Ratio d'activité à l'instant t $x(t) = \frac{A(t)}{n}$
x_{seuil}	Ratio seuil pour le déclenchement
$de(t)$	Date du dernier rééquilibrage
$W(t)$	Travail réalisé par les processeurs actifs depuis le dernier rééquilibrage $W(t) = \sum A(i)$
$\sim PE(t)$	Équivalent PE à l'instant t $\sim PE(t) = \frac{W}{t-de(t)}$
$\sim PE_{eq}(t)$	Équivalent d'équilibrage à l'instant t $\sim PE_{eq}(t) = \frac{W}{t-de(t)+L}$
$\sim PE_{gain}(t)$	Équivalent de gain à l'instant t $\sim PE_{gain}(t) = \frac{2\times W}{2t+L}$
POT(t)	Opérateur parallélisme potentiel
$\alpha[k]$	Charge du processeur k susceptible de participer à l'équilibrage

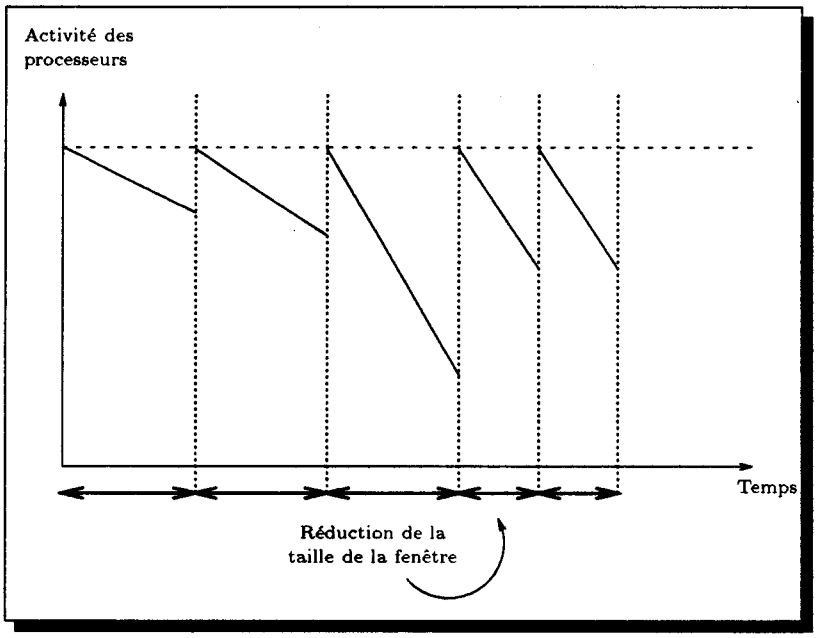


FIG. III.32 - *Déclencheur périodique adaptatif*

les valeurs de α et β qui permet d'éviter une répercussion trop brutale des changements du système sur le mécanisme.

$$\alpha, \beta \leq 0.2$$

4.3 Les politiques adaptatives

Les politiques adaptatives provoquent une phase d'équilibrage d'après les informations rassemblées. Si la décision de déclenchement est prise instantanément, on parle de politiques adaptatives instantanées, si elle prise en estimant l'état futur du système, on parle de politiques prédictives adaptatives.

4.3.1 Une politique adaptative instantanée

Dans une politique adaptative instantanée, la stratégie est déclenchée après une analyse du système à l'instant t. Aucune supposition n'est effectuée sur le comportement futur du système.

Le mécanisme de déclenchement le plus intuitif est de décider d'effectuer le rééquilibrage

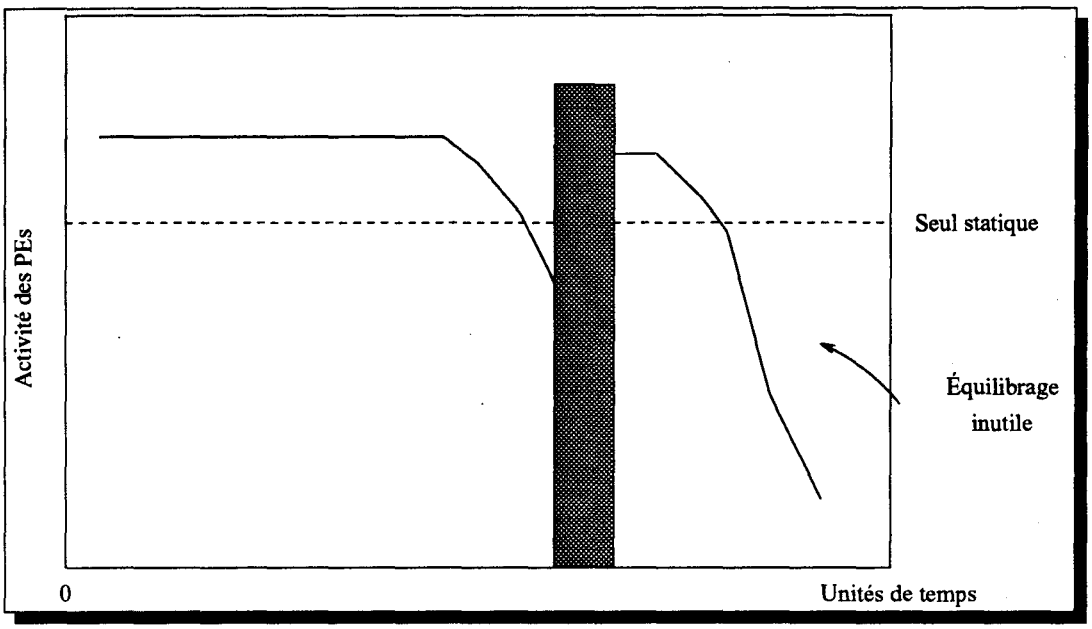


FIG. III.33 - *Évolution du nombre de processeurs actifs en fonction du temps*

lorsque le ratio de processeurs actifs passe en dessous d'un seuil donné x_{seuil}. Formellement, la routine de redistribution est appelée, lorsque :

$$\boxed{A(t) \leq n x_{\text{seuil}} \iff x(t) \leq x_{\text{seuil}}}$$

De plus, l'utilisation de ce déclencheur instantané présente un inconvénient, c'est qu'il ne prend pas en compte les effets de bord. En effet, lorsque l'on arrive à la fin de l'exécution du programme, il y a de fortes probabilités pour qu'un grand nombre de processeurs deviennent inoccupés et que l'appel à l'algorithme de rééquilibrage soit inutile.

Prenons l'exemple de la figure III.33 où le rééquilibrage est inutile en fin d'exécution. Cependant, le mécanisme se déclenche et fait appel à l'algorithme de rééquilibrage.

Pour palier à cet inconvénient, nous proposons l'introduction d'un opérateur qui permet de refléter ces changements et qui évite de déclencher le rééquilibrage lorsque l'on se trouve en fin d'exécution du programme.

L'opérateur Parallélisme pOTentiel

L'opérateur POT(t) (abréviation de Parallélisme pOTentiel) permet de ne pas tenir compte des éventuels effets de bord. On considère qu'il existe un parallélisme potentiel ou in-

exploité, si des processeurs contiennent des objets en attente d'exécution alors qu'ils auraient pu être exécutés sur des processeurs libres. Il est défini de la manière suivante :

Soit $\alpha[k]$, le nombre d'objets présents sur le processeur k qui sont susceptibles de participer à la redistribution des données :

$$\alpha[k] = \max(0, w[k] - 1)$$

Définition 11 *L'opérateur POT(t) parallélisme potentiel exprime le degré potentiel de parallélisme*

On a alors la valeur de l'opérateur POT qui est de :

$$\mathrm{POT}(t) = \begin{cases} 1 & \text{si } A(t) = n \\ \frac{\sum \alpha[k]}{n - A(t)} & \text{sinon} \end{cases}$$

Ce qui correspond à :

$\mathrm{POT}(t) = 1$	Tous les processeurs inactifs peuvent être comblés
$\mathrm{POT}(t) = 0$	La redistribution est inutile.
	Aucun PE inactif ne trouvera de partenaire
$0 < \mathrm{POT}(t) < 1$	Seule une partie des inactifs pourront trouver un partenaire

Par exemple, en utilisant de la figure III.33 avec l'opérateur POT, on obtient le graphique III.34 :

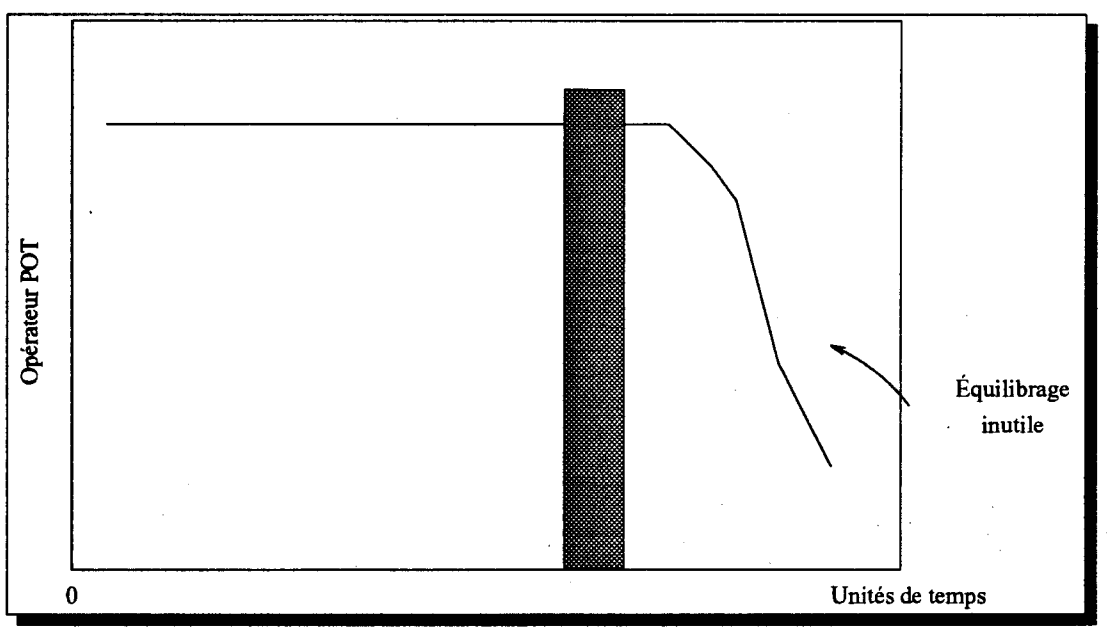


FIG. III.34 - *Action de l'opérateur POT sur l'exemple de la figure III.33*

L'opérateur POT permet l'utilisation d'un seuil statique x_{seuil} en évitant les effets de bord. Il évite par exemple les appels inutiles à la routine de rééquilibrage en fin d'exécution. Il est également utile pour les problèmes dont la charge peut varier dynamique (i.e. des pères créent des fils).

4.3.2 Les politiques adaptatives prédictives

Outre les problèmes du choix de la valeur de x_{seuil} lorsque la quantité de travail décroît, le coût de l'équilibrage reste sensiblement constant. Il est utile de faire intervenir un mécanisme qui prenne en compte l'état actuel et futur du système; un tel mécanisme est appelé adaptatif prédictif.

Il est utile d'estimer le coût de la prochaine phase d'équilibrage. Sa durée sera calculée en moyennant les précédentes durées. Le temps moyen d'un rééquilibrage sera noté L.

Une politique basée sur l'équivalent PEs

Intuitivement, un des moments optimaux pour déclencher une phase de redistribution est l'instant où le déclenchement d'un équilibrage ne peut qu'améliorer le comportement du système. En d'autres mots, si la routine de redistribution n'est pas appelée, le système ne peut que continuer à se dégrader. Cette technique utilise la fonction équivalent PEs qui évalue le nombre de processeurs élémentaires nécessaires utilisés à 100% pour effectuer le travail jusqu'au moment présent.

Nous étendons la définition de la fonction $\sim PE$ afin d'inclure la notion d'équilibrage. Si l'équilibrage est effectuée, la durée sur laquelle la fonction équivalent doit être étudiée n'est pas t, mais t majorée de la durée d'une phase d'équilibrage, soit $t + L$. Ceci nous permet d'introduire la définition suivante :

Définition 12 *L'équivalent PEs à l'instant t avec équilibrage $\sim PE_{eq}$ correspond au calcul de la fonction équivalent PEs incluant la prochaine phase d'équilibrage.*

$$\sim PE_{eq}(t) = \frac{W(t)}{t + L}$$

Le tableau III.4 présente conjointement les fonctions $\sim PE$ et $\sim PE_{eq}$ avec $L = 10$ unités de temps pour la courbe d'activité des processeurs présentée en figure III.31. Les valeurs prises par la fonction $\sim PE_{eq}$ sont toujours inférieures à celles prises par $\sim PE$, ce qui est

TAB. III.4 - *Comparaison de $\sim PE$ avec ou sans équilibrage*

Temps	10	20	30	40	50	60	70	80	90	100
$\sim PE$	95	90	85	80	75	70	65	60	55	50
$\sim PE_{eq}$	47.5	60	63.7	64	62.5	60	56.7	53.3	49.5	45.5

normal puisque la fonction $\sim PE_{eq}$ fonction inclut dans son calcul une « inactivité » de tous les processeurs pendant une période de durée L. Asymptotiquement, on peut remarquer que les limites en l'infini des deux fonctions tendent vers 0, ce qui montre bien qu'attendre un temps infini pour effectuer un équilibrage n'est pas une bonne solution !

La figure III.35 présente l'évolution de la fonction $\sim PE_{eq}$ du tableau III.4. On remarque qu'il existe un extremum, en deçà et au-delà duquel les valeurs prises par $\sim PE_{eq}$ décroissent. Ce point correspond au *meilleur choix possible* pour le mécanisme de déclenchement. En effet, il correspond au gain *maximum* qu'il est possible d'obtenir en effectuant l'équilibrage. Deux remarques s'imposent :

1. La fonction $\sim PE_{eq}$ n'est calculée que de manière discrète, la valeur précise du point optimum ne sera pas connue.
2. La meilleure valeur ne peut être connue que rétroactivement par comparaison avec des valeurs antérieures.

Si l'on suppose que l'échantillonage des valeurs de la fonction $\sim PE_{eq}$ est réalisé à des instants t_i, le critère de décision réagit si :

$$\boxed{\sim PE_{eq}(t_i) < \sim PE_{eq}(t_{i-1})}$$

Une politique de décision basée sur l'estimation du gain éventuel dû à l'équilibrage

Cette méthode autorise la comparaison de la fonction équivalent PEs à l'instant présent, avec cette même fonction si l'équilibrage avait lieu et si la courbe représentant l'activité des processeurs élémentaires possédait sensiblement la même décroissance que celle qu'elle suit actuellement. Cette hypothèse est faite pour pronostiquer l'état futur du système. Ce mécanisme de déclenchement fait donc la supposition que la courbe de décroissance d'activité $A(t)$ se répète de façon plus ou moins identique dans le temps comme indiqué sur la figure III.37. L'équilibrage est déclenché à $t = 50$s, et l'on suppose que la pente de la courbe d'activité après

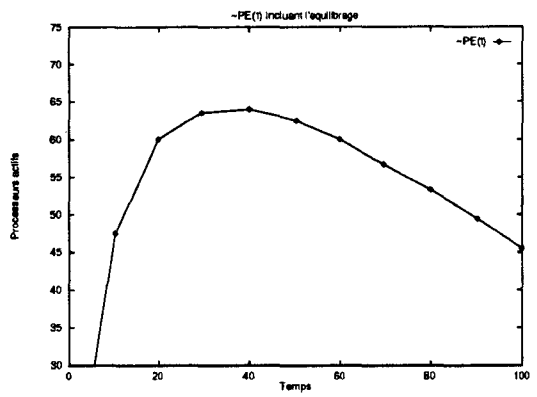


FIG. III.35 - *Évolution de la fonction $\sim PE_{eq}$ incluant le rééquilibrage*

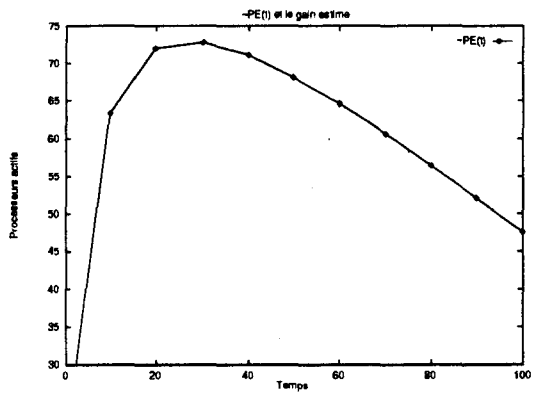


FIG. III.36 - *Évolution de la fonction $\sim PE_{gain}$ estimant le gain possible de l'équilibrage*

TAB. III.5 - *Calcul du gain d'équilibrage*

Temps	10	20	30	40	50	60	70	80	90	100
$\sim PE_{gain}(t)$	63.3	72	72.8	71.1	68.1	64.6	60.6	56.4	52.1	47.6

la répartition des données sera identique à la pente avant la redistribution. Cette contrainte nous permet alors de calculer la fonction équivalent PEs qui estime le gain dû au rééquilibrage.

Définition 13 *L'équivalent PEs de gain, $\sim PE_{gain}$ est le calcul de l'équivalent PE incluant le gain possible dû à l'équilibrage. Formellement:*

$$\sim PE_{gain} = \frac{2 \times W(t)}{2t + L}$$

La figure III.36 indique l'évolution de la fonction $\sim PE_{gain}$ pour une décroissance régulière de l'activité en fonction du temps. La fonction $\sim PE_{gain}$ est échantillonnée à différents instants t_i dont les valeurs sont reprises par le tableau III.5. L'instant optimal de déclenchement de la politique d'équilibrage correspond à l'extremum de la courbe. En effet, ce point particulier correspond au moment où la fonction équivalent PEs prend sa valeur maximale. Si la répartition des données est effectuée à cet instant, le gain calculé sur une longueur de $2t + L$ sera maximum. Par contre, si l'on attend quelques itérations supplémentaires, la valeur de $\sim PE_{gain}$ sera moindre indiquant que la fonction équivalent PEs moyennée sur une durée de $2t + L$ ne pourra être que plus faible.

La valeur du point correspondant à l'extremum de la courbe n'est connue qu'après avoir été dépassée. Le déclenchement de la politique de décision n'est donc réalisé qu'à l'échantillonage

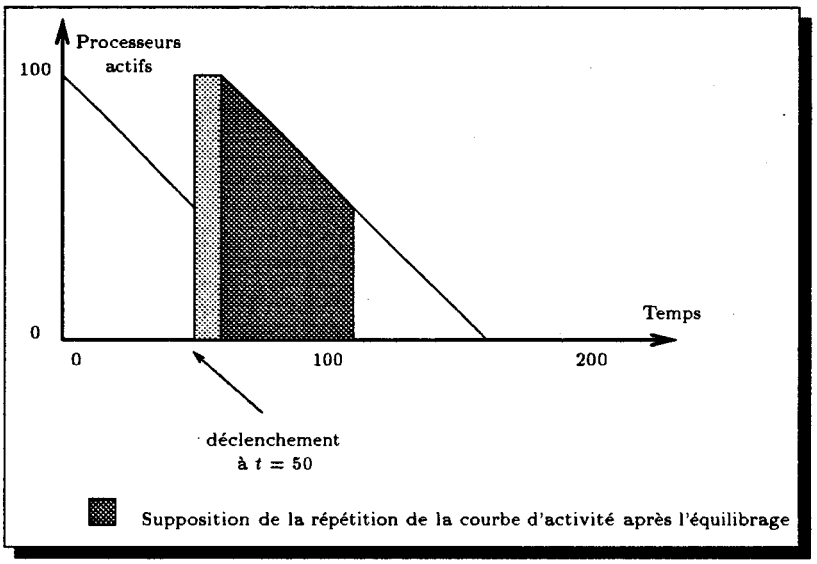


FIG. III.37 - *Présentation de l'équivalent PEs estimant le gain/perte provoqué par l'équilibrage*

suivant ce point particulier. Si l'on suppose que l'échantillonage des valeurs de la fonction $\sim PE_{gain}(t)$ est réalisé à des instants t_i, le critère de décision réagit si :

$$\boxed{\sim PE_{gain}(t_i) < \sim PE_{gain}(t_{i-1})}$$

Pour l'exemple présenté dans la figure III.36 où la durée de l'équilibrage est de 10 unités de temps, le point optimum correspond à $t = 30$. Le début de la répartition des données sera fera lors du calcul de la prochaine valeur, soit à $t = 40$.

Une politique comparant l'inactivité cumulée des PEs à l'équilibrage

Lorsque le nombre d'itérations augmente, l'activité des processeurs élémentaires diminue. Le « travail » non effectué par les processeurs oisifs croît en conséquence. Le but de ce mécanisme de est de minimiser la part de l'inactivité cumulée des PEs et de se déclencher dès que cette part devient supérieure au coût d'un équilibrage.

La figure III.38 représente l'activité des PEs en fonction du temps. Les surfaces grisées indiquent les périodes de répartition des données. L'équilibrage se produit dès que l'inactivité des processeurs représentée par la surface S_1 devient plus importante que le coût de l'équilibrage représenté par S_2 exprimé en PEs actifs×unités de temps. Formellement, le mécanisme se déclenche à l'instant t si :

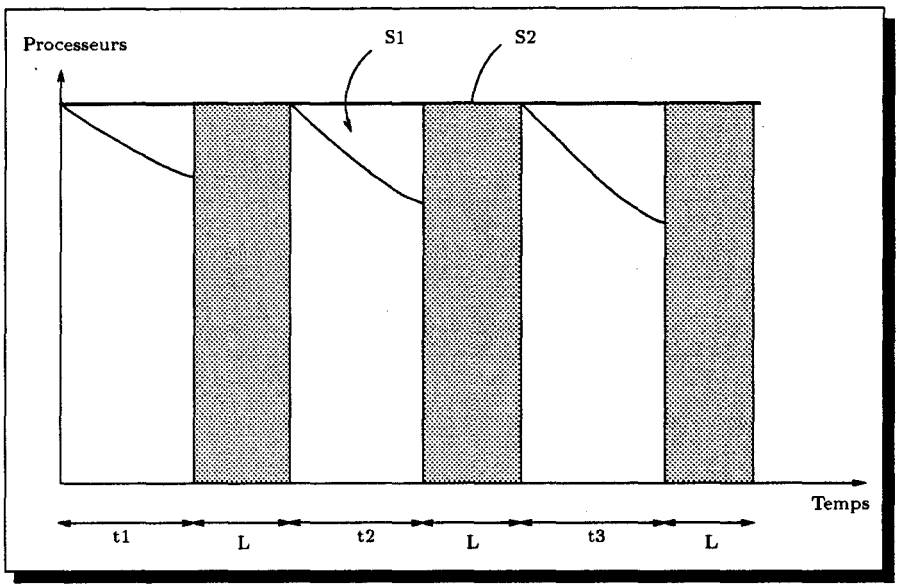


FIG. III.38 - *Présentation du mécanisme basé sur la part des PEs inactifs*

TAB. III.6 - *Inactivité provoquée par les processeurs oisifs*

Temps	10	20	30	40	50	60	70	80	90	100
$W(t)$	50	200	450	800	1250	1800	2450	3200	4050	5000

$$\boxed{n \times t - W(t) \geq L \times n}$$

Le tableau III.6 donne la part du « non travail » des PEs oisifs au cours du temps pour la fonction d'activité présentée à la figure III.31.

Dans cet exemple, le coût en PEs actifs×unités de temps de l'équilibrage est égal à 10×100 ($n = 100$ processeurs), soit donc 1000 PEs actifs×unités de temps. La figure III.39 présente conjointement l'inactivité des processeurs en fonction du temps par rapport au coût moyen d'un équilibrage. Le mécanisme se déclenche dès que les deux courbes se rejoignent indiquant qu'effectuer le rééquilibrage à ce moment compenserait son coût. Dans l'exemple présenté, les deux courbes se coupent peu après $t = 40$ unités de temps, la redistribution a donc lieu pour $t = 50$.

Un mécanisme comparant l'activité effective avec ou sans équilibrage

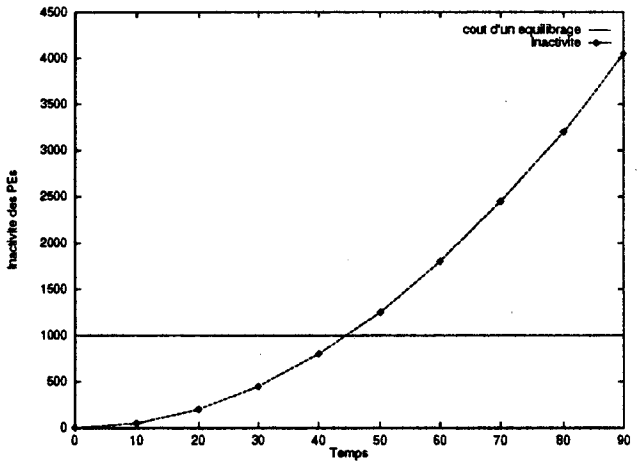


FIG. III.39 - *Déclencheur basé sur la comparaison entre le coût d'un équilibrage et l'inactivité des PEs*

Le dernier mécanisme prend en compte les avantages que procure le rééquilibrage, c'est-à-dire qu'il estime le gain que l'on peut acquérir par ce changement et le compare au travail que l'on aurait obtenu si l'on avait continué sans effectuer de rééquilibrage.

Dans la figure III.40, la surface hachurée représente le gain de travail obtenu par le rééquilibrage s'il est lancé à la fin de la période t_1 , le trait en pointillé indique l'allure de la courbe du nombre de processeurs actifs si l'on continuait sans effectuer le rééquilibrage.

Si l'on continue sans effectuer de rééquilibrage, la travail effectué serait de :

$$W_2 = \sum_{t_1} A(t) + \sum_{L+t'_2} A(t)$$

En utilisant une approximation polynômiale pour évaluer la pente de la courbe $A(t)$ au-delà de t_1, le taux de travail serait donc de :

$$T_{\text{sans}} = \frac{W_2}{t_1 + L + tt_2}$$

Si l'algorithme de redistribution est appelé, on estime le nombre de processeurs actifs en reprenant la courbe du nombre de processeurs du dernier rééquilibrage. Dans ce cas, le travail et le taux de ce dernier sont respectivement de :

$$W_1 = \sum_{t_1} A(t) + \sum_{t'_2} A(t)$$

$$T_{\text{avec}} = \frac{W_1}{t_1 + L + t'_2}$$

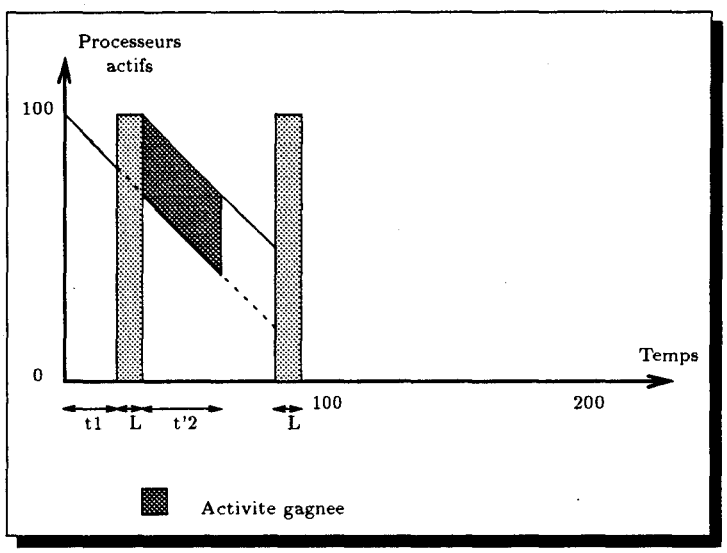


FIG. III.40 - *Mécanisme basé sur le taux d'activité avec ou sans équilibrage*

Le mécanisme se déclenche lorsque l'équation suivante est vérifiée :

$$\boxed{T_{\text{avec}}(t) \geq T_{\text{sans}}(t)}$$

La figure III.41 présente conjointement les taux d'activités sans et avec rééquilibrage. Les deux courbes se croisent aux environs de $t = 40$ unités de temps. Le meilleur choix possible pour déclencher une phase de redistribution des données se situe donc à $t = 50$ unités de temps.

4.4 Conclusion des politiques génériques de déclenchement

Les politiques de déclenchement constituent un élément important de la politique d'équilibrage dans le modèle à parallélisme de données définie dans la section 1.

Les déclencheurs périodiques montrent leur insuffisance. La période choisie est à la fois dépendante de la machine utilisée et du problème à résoudre. Les déclencheurs adaptatifs instantanés s'adaptent à de nombreux problèmes et n'introduisent qu'un faible sur-coût à l'équilibrage. Un problème survient lorsque la quantité de travail total diminue. Le coût du réarrangement dynamique reste sensiblement constant et l'équilibrage peut devenir inutile. Une solution est d'utiliser l'opérateur POT(t) de parallélisme potentiel ou d'utiliser des déclencheurs prédictifs adaptatifs permettant d'appréhender la durée du prochain équilibrage.

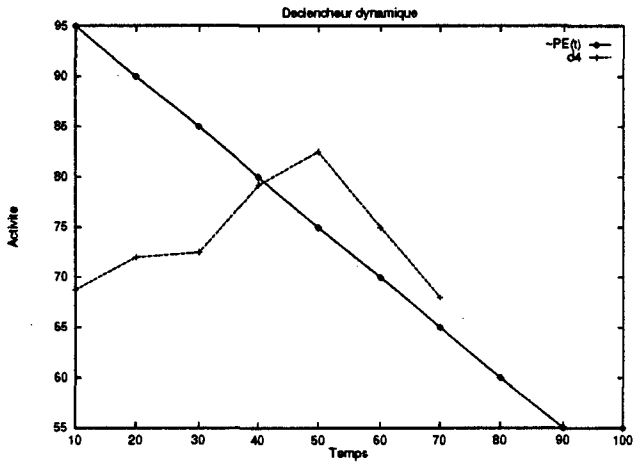


FIG. III.41 - *Exemple du déclencheur basé sur le taux d'activité avec ou sans équilibrage*

Le déclencheur $\sim PE_{eq}$ permet de comparer le taux d'activité avec ou sans équilibrage. Des mécanismes prédictifs plus puissants comme l'équivalent de gain peuvent être utilisés, mais ils reposent sur une hypothèse de reproduction temporelle de la courbe d'activité des processeurs élémentaires.

Chapitre IV

Analyse théorique des politiques de migration

1 Introduction

Dans ce chapitre, nous proposons une étude des algorithmes *Rendez-vous*, *Râteau*, *Pavage*, *X-Pavage* et *Glissement Pré-calculé*.

L'algorithme *Râteau* sert de borne de comparaison pour les autres techniques de réarrangement dynamique. Nous étudions la qualité de la redistribution des données provoquée par cette technique. L'application de l'algorithme *Râteau* sur des architectures de dimension supérieure à 1 est également envisagée.

La vitesse de convergence de l'algorithme *Rendez-vous* est étudiée.

Les techniques *Pavage* et *X-Pavage* sont analysées suivant la méthode du coloriage de graphe [Cyb89, XL92, XL94b] présentée dans le chapitre II. Ces deux techniques d'équilibrage sont comparées afin de savoir si les meilleures performances intuitives de l'algorithme *X-Pavage* par rapport à *Pavage* sont prouvées théoriquement.

Nous montrons enfin la convergence effective de l'algorithme du *Glissement Pré-calculé*.

2 Quelques définitions

Définition 1 *Deux processeurs P_i et P_j sont k-différents si leur différence de charge en valeur absolue est inférieure ou égale à k.*

$$P_i\ P_j\ k\text{-différents} \Longleftrightarrow |w[i] - w[j]| \leq k$$

Définition 2 *Un algorithme est dit k-différent si deux processeurs quelconques en fin d'exécution sont k-différents.*

$$\text{algorithme } k\text{-différent} \Longleftrightarrow \forall i \in \mathcal{P}\, \forall j \in \mathcal{P}\ |w[i] - w[j]| \leq k$$

Une stratégie d'équilibrage produit un équilibrage parfait s'il est 1-différent. Les cas 0-différent est un cas particulier. Il ne peut se produire que si le nombre de données N est un multiple de n, le nombre de PEs.

Plus la valeur de k est faible, meilleure est la qualité de la technique de rééquilibrage.

3 Étude de l'algorithme Rendez-vous

L'algorithme *Rendez-vous* se caractérise par un transfert de données global au système. Nous allons montrer que des appels successifs à cette routine permettent d'obtenir un équilibrage parfait ; chaque PE possédant alors la charge moyenne du système. Les PEs sont triés suivant leur poids. Pour un système comportant n processeurs, le PE dont le rang dans le tri est i, égalisera sa charge avec le PE de rang $n - i$.

Théorème 1 *Sur un système de n processeurs, l'appel à l'algorithme Rendez-vous* $\log_2(n)$ *fois permet d'obtenir un système stable où chaque PE possède la charge moyenne du système.*

Hypothèse : Pour faciliter la démonstration, un système de 2^n processeurs sera choisi. Une étape de rééquilibrage correspond à un appel à la routine *Rendez-vous*.

Preuve : La démonstration s'effectue par récurrence. L'hypothèse de récurrence est la suivante :

À l'étape k, le système peut être partitionné en $\frac{2^n}{2^k}$ ensembles disjoints, chaque ensemble comportant 2^k processeurs élémentaires ayant la même charge.

Montrons que l'hypothèse est vérifiée pour $k = 1$. Pour la première étape, le PE de rang i dans le tri égalise sa charge avec le PE de rang $2^n - i$. L'appariement est réalisé en parallèle

sur tout le système. À la fin de l'appel au premier *Rendez-vous*, il existe dans le système $\frac{2^n}{2}$ ensembles possédant la même charge.

Montrons que l'hypothèse est vérifiée à l'étape $k+1$. Par hypothèse, le système peut être découpé en $\frac{2^n}{2^k}$ ensembles, chaque ensemble de PEs ayant la même charge. Soit $\mathcal{P}_0$, l'ensemble dont la charge est la plus faible. Les 2^k PEs appartenant à $\mathcal{P}_0$ seront associées avec les 2^k PEs les plus chargés appartenant à l'ensemble $\mathcal{P}_{2\frac{2^n}{2^k}}$. Les ensembles possèdent le même nombre de PEs, on assistera à un appariement d'ensembles. Les PEs de l'ensemble i seront appariés avec les PEs de l'ensemble $2\frac{2^n}{2^k} - i$. On obtient donc $\frac{2^n}{2^{k+1}}$ ensembles ayant la même charge. L'hypothèse de récurrence est bien vérifiée.

À l'étape n, on aura donc un unique ensemble de 2^n PEs possédant tous la même charge qui est donc la charge moyenne du système.

□

4 Étude de l'algorithme Râteau

Nous allons examiner le comportement de l'algorithme *Râteau* sur une topologie mono-dimensionnelle et son adaptation sur des systèmes de dimension supérieure à 1.

4.1 L'algorithme Râteau sur une topologie mono-dimensionnelle

Nous savons que l'application d'une phase d'équilibrage de la technique *Râteau* sur un anneau comporte un nombre d'étapes égal au nombre de processeurs. Le théorème suivant permet de définir la qualité de l'équilibrage.

Théorème 2 *Pour un anneau, si la charge totale N est un multiple du nombre de processeurs n, alors, pour l'algorithme Râteau, à la fin de la phase de mouvement, on a:*

$$\forall i \in \mathcal{P}, w[i] = q$$

où q est la charge moyenne globale du système ($q = N/n$).

Hypothèse : Soit un anneau de n processeurs. L'ensemble des processeurs de l'anneau peut être partagé en 3 ensembles disjoints $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2$ représentant respectivement les processeurs ayant une charge supérieure, inférieure ou égale à q.

$\mathcal{P}_0 = \{i/w[i] > q\}$ $\text{Card}(\mathcal{P}_0) = p_0$
$\mathcal{P}_1 = \{i/w[i] < q\}$ $\text{Card}(\mathcal{P}_1) = p_1$
$\mathcal{P}_2 = \{i/w[i] = q\}$ $\text{Card}(\mathcal{P}_2) = p_2$

Une étape de l'algorithme *Râteau* correspond à un envoi de données (i.e. **send**).

Preuve : La démonstration nécessite l'introduction de la distance circulaire $\delta(i,j)$ entre deux processeurs i et j. Elle est définie de la manière suivante :

$$\delta(i,j) = \begin{cases} j - i & \text{si } j \geq i \\ n - (j - i) & \text{sinon} \end{cases}$$

Cette distance δ définit une distance mesurée de la gauche vers la droite de manière circulaire.

Nous allons effectuer une récurrence sur le nombre d'étapes et montrer qu'à l'étape k, la distance circulaire la plus grande entre un PE surchargé ($\in \mathcal{P}_0$) et un PE sous-chargé ($\in \mathcal{P}_1$) est inférieure ou égale à $n - k$.

De façon précise, la distance d est la distance maximale entre les deux ensembles $\mathcal{P}_0$ et $\mathcal{P}_1$

$$d = \max\{\delta(i,j) | \forall i,j \;\; i \in \mathcal{P}_0 \text{ et } j \in \mathcal{P}_1\}$$

Nous faisons l'hypothèse de récurrence suivante : <u>à l'étape k, $d < n - k$.</u>

L'hypothèse de récurrence est bien évidemment vérifiée pour $k = 1$. Dans le cas le plus défavorable où un PE contient l'ensemble de la charge du système, la distance d est égale à $n - 1$.

Supposons l'hypothèse vérifiée pour k, montrons qu'elle est vérifiée pour $k + 1$.

On a $d \leq n - k$. Soit $i_0 \in \mathcal{P}_0$, le processeur le plus « éloigné » des processeurs de P_1. On est assuré de l'existence du processeur i_0, son absence signifierait que tous les processeurs appartiennent à l'ensemble $\mathcal{P}_2$. Le théorème serait alors vérifié.

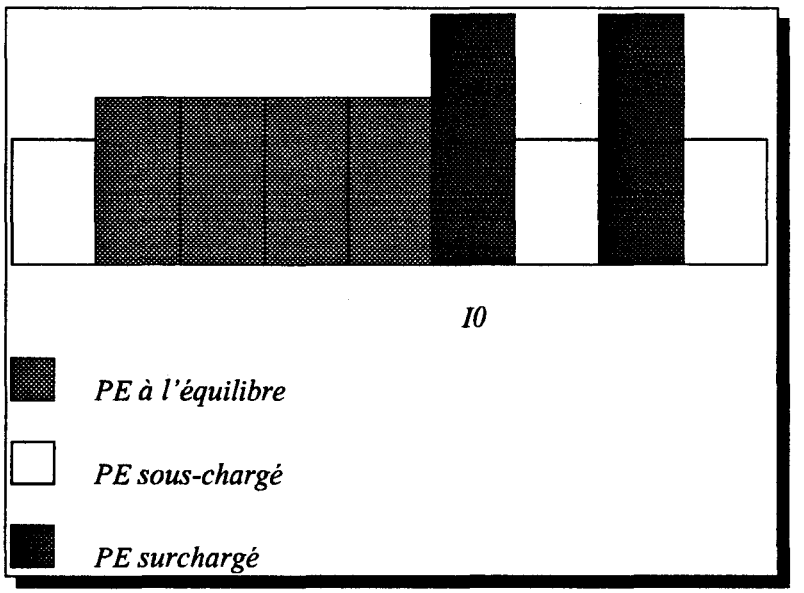


FIG. IV.1 - *Exemple d'exécution de l'algorithme Râteau pour un ensemble de 9 processeurs à la quatrième étape*

Examinons l'état du processeur précédant i_0 (soit $(i_0 - 1) \bmod n$).

- il ne peut pas appartenir à l'état $\mathcal{P}_0$, car cela contredit l'hypothèse que le processeur i_0 est situé à la distance maximum ;
- il ne peut appartenir à $\mathcal{P}_1$, car la distance d serait alors égale à $n - 1$.

Le processeur précédent i_0 appartient donc à $\mathcal{P}_2$. On peut répéter ce processus et remarquer que les k processeurs précédants i_0 appartiennent à l'ensemble $\mathcal{P}_2$. À l'étape k, on a donc $p_2 \geq k$, ces k processeurs dont la charge est q sont contigus. Le $(k+1)$ième PE précédant de façon circulaire i_0 appartiendra soit à $\mathcal{P}_2$ soit à $\mathcal{P}_1$. S'il appartenait à $\mathcal{P}_0$ cela contredirait l'hypothèse qui indique que le processeur i_0 est le plus éloigné. On se trouve dans une situation comme celle présentée à la figure IV.1. Le processeur i_0 est précédé à l'étape k de k processeurs appartennant à $\mathcal{P}_2$.

Le passage à l'étape $k+1$ provoque le passage du processeur i_0 dans l'ensemble $\mathcal{P}_2$.

À l'étape $k+1$, on a $p_2 \geq k+1$, par passage du processeur i_0 de l'ensemble $\mathcal{P}_0$ à $\mathcal{P}_2$. Ce qui entraîne $d \leq n - (k+1)$.

L'hypothèse de récurrence est bien vérifiée. Dans l'algorithme *Râteau* à l'étape k, on a $d \leq n - k$ et $p_2 \geq k$.

Après n itérations, $d = 0$, $p_2 = n$. Tous les PEs sont donc à l'équilibre.

□

Lemme 1 *L'algorithme Râteau pour un anneau de charge quelconque est 1-différent*

Preuve : r désigne le reste de la division euclidienne et q la charge moyenne du système.

D'après le théorème 2, les $n - r$ premières itérations provoquent le passage de $n - r$ PEs contigus dans l'ensemble $\mathcal{P}_2$. On peut appliquer de nouveau le théorème pour les r itérations restantes, elles provoquent le passage de r PEs à la charge $q + 1$.

$$\forall i, j \in \mathcal{P} \ |w(i) - w(j)| \leq 1$$

D'après la définition de la k-différence, on en déduit que l'algorithme *Râteau* est 1-différent.

□

4.2 Extension de l'algorithme Râteau sur une topologie multi-dimensionnelle

L'algorithme *Râteau* sur plusieurs dimensions est une extension de la technique *Râteau* sur une seule dimension

Théorème 3 *L'algorithme Râteau pour α dimensions est α-différent si le nombre de processeurs dans chaque dimension est supérieur au nombre total de dimension.*

$$\forall i \in \mathcal{P}, \ q \leq w[i] \leq q + \alpha$$

Preuve : Nous allons résonner par récurrence en supposant que la propriété énoncée est vraie sur une machine de dimension k.

Montrons que notre hypothèse est vérifiée lorsque $k = 1$:

On a vu que pour un anneau :

$$\forall i \in \mathcal{P}, \ q \leq w[i] \leq q + 1$$

L'hypothèse de récurrence est vérifiée pour $k = 1$.

Supposons que l'algorithme *Râteau* entraîne une k-différence pour un système à k dimensions. Montrons qu'il provoque une $(k+1)$-différence pour une topologie à $k+1$ dimensions.

Dans un système à $k+1$ dimensions, un PE est repéré de façon unique par $k+1$ coordonnées, chaque coordonnée représentant une dimension. On suppose pour simplifier la présence de n processeurs dans chaque dimension ($n > k$).

Après application de l'algorithme *Râteau* sur les k premières dimensions, on a pour toute dimension $i \leq k$ et pour tous les processeurs élémentaires de cette dimension :

$$\text{pour } i \leq k,\ \forall y \in n\ , |w[x_1, x_2, \ldots, x_y, \ldots, x_k, x_{k+1}] - w[x_1, x_2, \ldots, x_y, \ldots, x_k, x_{k+1}]| \leq 1 \tag{IV.1}$$

puisque l'algorithme *Râteau* est 1-différent sur un anneau.

La formule IV.1 explique qu'en se promenant sur une dimension inférieure ou égale à k (en fixant tous les autres coordonnées), la différence de charges entre deux processeurs quelconques est au maximum de 1.

Les k premiers équilibrages ont provoqué d'après l'hypothèse de récurrence, une k-différence dans les k premières dimensions :

$$\forall x_1, x_2, \ldots, x_k\ |w[x_1, x_2, \ldots, x_k, x_{k+1}] - w[x_1, x_2, \ldots, x_k, x_{k+1}]| \leq k \tag{IV.2}$$

Réaliser le rééquilibrage dans la $k+1^{\text{e}}$ dimension revient à effectuer un équilibrage *Râteau* sur plusieurs anneaux dont la charge pour chaque PE est k-différente. Dans le cas le plus défavorable, un anneau possédera une charge supérieure de $k \times n$ objets[1] que l'anneau le moins chargé. Comme $k < n$, la charge de l'anneau le plus lourd sera « lissé » sur les n processeurs de la $(k+1)^{\text{e}}$ dimension et entraînera une $(k+1)$-différence entre les processeurs.

L'application de l'algorithme *Râteau* sur un système à α dimensions permet d'obtenir une α-différence pour un coût de $O(\alpha \times n)$ où n est la nombre total de processeurs dans chaque dimension.

□

1. Par définition de la k-différence

5 Étude des algorithmes Pavage et X-Pavage sur une grille torique

5.1 Introduction

Le but de cette section est d'analyser le comportement des techniques *Pavage* et *X-Pavage* sur une grille théorique par la méthode itérative matricielle présentée dans le chapitre II. Les principaux résultats de la méthode itérative matricielle sont rappelés. Nous examinerons ensuite la convergence effective de ces deux algorithmes ainsi que leur vitesse de convergence. Les deux approches seront ensuite comparées.

5.2 Présentation de la méthode itérative matricielle

Afin d'analyser le comportement asymptotique de cet algorithme, nous adoptons la démarche proposée par Cybenko [Cyb89] et reprise par Xu et Lau [XL92]. À partir du système, nous considérons un graphe $G = (N, A)$ où l'ensemble des nœuds N représente l'ensemble des processeurs et une arête (i, j) représente un lien physique entre les processeur P_i et P_j.

Le graphe G est transformé en un graphe coloré $G_k = (N, A_k)$. L'ensemble N représente toujours l'ensemble des processeurs du système, une arête est définie par un triplet (i,j;c) qui indique une liaison entre les processeurs P_i et P_j d'une couleur c. Le nombre de couleurs pour le graphe G_k est le nombre *minimal* de couleurs tel que deux arêtes adjacentes n'aient pas la même couleur. $\delta(G_k) = k$ est le nombre minimum de couleurs pour décrire le graphe G_k, $\delta(i)$ est le nombre de couleurs différentes sur les arêtes du nœud (processeur) i.

La figure IV.2 est un exemple de coloriage pour une grille torique de dimension 4×4. Les entiers sur les arêtes représentent la couleur du lien. Les nœuds sont numérotés de 1 à n. La coloration des nœuds sert à décrire la séquentialité des échanges, une phase de rééquilibrage dynamique se compose dans le cas de la grille de la figure IV.2 de 4 étapes, chaque étape correspondant à un échange de travail simultané entre tous les processeurs reliés par un lien de même couleur. Si deux processeurs ne sont pas reliés, on aura dans ce cas, un échange de travail nul.

À l'étape t avec $c = (t \bmod k) + 1$, si w_i^t indique la charge du processeur à l'instant t, λ le paramètre définissant le travail échangé entre deux processeurs. Lorsque $\lambda = 1/2$, l'algorithme est appelé *DE* (« Dimension Exchange »).

$$\begin{cases} w_i^{t+1} = (1-\lambda)w_i^t + \lambda w_j^t & \text{si}(i,j;c) \text{ existe} \\ w_i^{t+1} = w_i^t & \text{sinon} \end{cases}$$

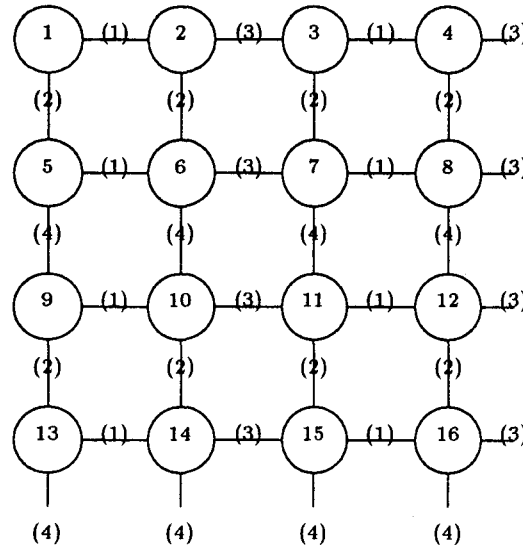


FIG. IV.2 - *Grille* 4×4

En définissant le vecteur distribution de charges sous forme vectorielle $W^t = (w_1^t, w_1^t, \ldots, w_{n-1}^t)$, on peut écrire sous forme matricielle l'étape de rééquilibrage de la manière suivante :

$$W^{t+1} = M_c(\lambda)W^t$$

Les coefficients de la matrice s'expriment simplement de la manière suivante :

$$\begin{cases} (M_c(\lambda))_{ii} = (1-\lambda) & \\ (M_c(\lambda))_{ij} = (M_c(\lambda))_{ji} = \lambda & \text{si } (i,j;c) \text{ existe} \\ (M_c(\lambda))_{ij} = 0 & \text{sinon} \end{cases}$$

où $M_c(\lambda)_{ij}$ indique l'entrée de la matrice $M_c(\lambda)$ située à la ligne i et à la colonne j.

Une phase complète du rééquilibrage est composée de la multiplication du vecteur W^t par l'ensemble des 4 matrices de couleurs.

$$W' = M(\lambda)W^t \text{ avec } M(\lambda) = M_4(\lambda) \times M_3(\lambda) \times M_2(\lambda) \times M_1(\lambda)$$

Les définitions suivantes permettent d'étudier l'éventuelle convergence de la matrice $M(\lambda)$.

Définition 3 *Une matrice est dite non négative si l'ensemble de ses coefficients ne sont pas négatifs*

Définition 4 *Une matrice M est dite doublement stochastique si la somme de ces coefficients pour chaque ligne et pour chaque colonne est égale à 1, c'est-à-dire :*

$$\forall(i,j) \sum_{1\leq k\leq n} a_{ik} = \sum_{1\leq k\leq n} a_{kj} = 1$$

Définition 5 *Une matrice est dite primitive s'il existe un entier positif s tel que $M^s > 0$*

Lemme 2 *Le produit d'une matrice non négative, primitive et doublement stochastique par elle-même est non négative, primitive et doublement stochastique [BP79, XL92]*

Définition 6 *Les matrices qui sont à la fois primitives et doublement stochastiques seront appelés matrices PDS.*

Les théorèmes de [BP79] repris par [XL92] permettent d'affirmer la propositions suivante.

Théorème 4 *Soit $M(\lambda)$ une matrice PDS d'un graphe de k couleurs G_k, $\bar{M}$ une matrice $n \times n$ avec tous les éléments égaux à $1/n$, et $\bar{W}$ le vecteur distribution uniforme avec tous les éléments égaux à 1. Alors,*

1. $\lim_{t\to\infty} M^t(\lambda) = \bar{M}$.
2. *Pour toute distribution de charge W^0, la suite $\{W^k\}$ converge vers $b\bar{W}$, avec $b = \sum_{1\leq i\leq n}(W_i^0)/n$.*

La vitesse de convergence de la suite $\{W^k\}$ peut être connue de la manière suivante. Soit $\mu(M(\lambda))$ l'ensemble des valeurs propres de la matrice $M(\lambda)$. 1 est la plus grande des valeurs propres. $\gamma(M(\lambda))$ est la valeur propre sous-dominante (la deuxième plus grande valeur propre) de l'ensemble des valeurs propres, soit :

$$\gamma(M(\lambda)) = \max\{|\mu(M(\lambda))| : \mu(M(\lambda)) \neq 1\}$$

alors la vitesse de convergence asymptotique de la suite $\{W^k\}$ est égale à $-\ln\gamma(M(\lambda))$.

5.3 Étude de l'algorithme Pavage sur une grille torique

Nous montrons dans un premier temps que la technique d'équilibrage *Pavage* est équivalent à l'algorithme *DE*. Nous prouvons sa convergence et la qualité de l'équilibrage sur une grille torique de taille 4×4.

5.3.1 Convergence de la méthode Pavage

L'algorithme *Pavage* travaille sur la grille de la machine en partitionnant le système en carré de 4 éléments. Un rééquilibrage est effectué sur ce pavage afin d'égaliser la charge puis la fenêtre de redistribution est décalée comme indiquée sur la figure IV.3.

Théorème 5 *L'algorithme Pavage est équivalent à la méthode DE de (paramètre d'échange* $\lambda = 1/2$*).*

Preuve : Par construction de l'algorithme *Pavage* sur une grille, les processeurs appartenant à un même carré de rééquilibrage égalisent leur charge suivant une direction ouest-est dans un premier temps. On peut donc numéroter ce lien 1. De la même manière, à l'intérieur de ce carré de rééquilibrage, les processeurs partagent leur travail suivant une direction nord-sud comme indiqué sur la figure IV.3 ; le lien est numéroté 2. Ensuite, le carré de rééquilibrage est déplacé d'une façon torique d'un processeur dans la direction est-ouest et d'un processeur dans la direction nord-sud. Le rééquilibrage effectué alors dans ce carré de rééquilibrage peut être numéroté 3 et 4. On est sur qu'il n'y aura pas de redondance entre ces numérotations et les précédentes puisque le carré de rééquilibrage a été déplacé. Chaque processeur aura donc 4 liens.

□

Théorème 6 (Convergence de l'algorithme Pavage) *Pour tout vecteur de distribution initiale* W^0*, il y a convergence de la méthode de rééquilibrage Pavage pour une grille torique de taille* $2n \times 2n$*.*

Preuve : L'algorithme *Pavage* correspond d'après le théorème 5 à l'algorithme *DE* ($\lambda = 1/2$). D'après le théorème 5 est équivalent à la méthode *DE* (*lambda* $= 1/2$). Une matrice d'équilibrage $M(\lambda)$ peut être construite.

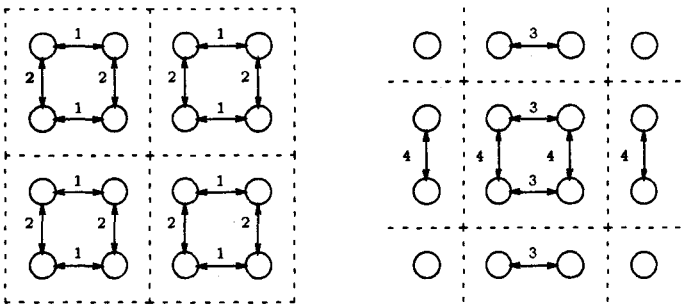


FIG. IV.3 - *Algorithme Pavage*

Le graphe coloré G_k associé à la grille est une extension du graphe coloré présenté à la figure IV.2. On a $\delta(i) = 4$ pour tout i. Par construction, les matrices $M_1(\lambda)$, $M_2(\lambda)$, $M_3(\lambda)$ et $M_4(\lambda)$ sont PDS. La matrice résultante $M(\lambda) = M_1(\lambda) \times M_2(\lambda) \times M_3(\lambda) \times M_4(\lambda)$ est PDS. D'après le théorème 4 , $\lim_{t\to\infty} M^t(\lambda)$ existe et $\lim_{t\to\infty} M^t(\lambda) = \bar{M}$.

□

5.3.2 Étude la converge de l'algorithme Pavage sur une grille torique 4×4

Nous allons montrer que l'application de l'algorithme *Pavage* sur une grille torique de taille 4×4 permet d'obtenir un équilibrage parfait. Pour cette démonstration, nous allons construire élément par élément la matrice $M(\lambda)$ et nous montrerons que tous les coefficients de la matrice $M(\lambda)$ sont égaux. Les différents coefficients de la matrice seront construits par l'intermédiaire du chemin coloré.

Définition 7 *Soit G_k un graphe de k-couleurs. Un chemin coloré de i à j est de la forme :*

$$(i = i_0, i_1; c_1), (i_1, i_2; c_2), \ldots, (i_{l-1}, i_l = j; c_l)$$

où tous les nœuds intermédiaires $i_s (1 \leq s \leq l-1)$ sont distincts et $k \geq c_1 > c_2 > \ldots > c_l \geq 1$

Cette définition exprime qu'un chemin doit être suivi dans l'ordre des couleurs décroissante.

Le lemme suivant démontré par Xu et Lau [XL92] permet de construire les entrées de la matrice $M(\lambda)$.

Lemme 3 *Soit $M(\lambda)$ une matrice PDS d'une graphe k-coloré G_k. Si $0 < \lambda < 1$, alors pour tout $i \geq 1, j \leq n$*

$$m_{ij}(\lambda) = \sum_{p \in \mathcal{P}_{ij}} (1-\lambda)^{r_p} \lambda^{l_p}$$
$$m_{ii}(\lambda) = (1-\lambda)^{\delta(i)} + \sum_{p \in \mathcal{P}_{ii}} (1-\lambda)^{r_p} \lambda^{l_p})$$

où l_p est la longueur du chemin coloré $p \in \mathcal{P}_{ij}$ de la forme :

$$(i = i_0, i_1; c_1), (i, i_2; c_2), \ldots, (i_{l_{p-1}}, i_{l_p} = j; c_{l_p})$$

avec tous les nœuds intermédiaires distincts et $c_1 > c_2 > \ldots > c_{l_p}$

et $r_p = \sum_{s=0}^{l_p}(n_s)$, où n_0 est le nombre d'arêtes incidentes du nœud i dont la couleur est plus grande que c_1; n_{l_p} est le nombre d'arêtes incidentes sur le nœud j dont la couleur est plus petite que c_{l_p}; et $n_s (1 \leq s \leq l_{p-1})$ est le nombre d'arêtes incidentes du nœud i_s dont la couleur est plus grande que c_{s+1} et plus petite que c_s.

Théorème 7 (Convergence sur une grille 4×4) *Pour une grille torique 4×4, l'application d'un équilibrage Pavage permet l'obtention d'une grille équilibrée.*

Nous allons calculer chaque entrée de la matrice à l'aide de la technique du chemin coloré. Nous allons successivement montrer que :

1. Dans une grille de taille 4×4, le chemin d'un processeur vers un autre est unique.
2. Toutes les entrées de la matrice sont égales.

Preuve :

1. Soit un PE i_0, soit c la couleur empruntée pour aller vers un processeur j. Il suffit de remarquer qu'une couleur partitionne respectivement la grille de processeurs suivant une ligne ou une colonne. Revenir vers ce processeur implique un nouvel emprunt de cette couleur, ce qui, par la définition du chemin coloré est impossible.
2. Comme il n'existe pas de chemin clos, la seconde partie de la formule donnant $m_{ii}(\lambda)$ est donc nulle ($\text{Card}\mathcal{P}_{ii} = 0$).

On a donc, pour tout i :

$$m_{ii}(\lambda) = (1-\lambda)^4$$

Il existe un unique chemin entre i et j, soit $\mathrm{Card}(\mathcal{P}_{ij}) = 1$. Il n'existe que 4 possibilités pour choisir un PE et tous ses voisins qui subiront durant les 4 étapes une partie de la redistribution. Les quatre choix possibles sont représentés dans les figures de la page 152. Le PE échange sa charge avec ses voisins situés dans un carré de 4 processeurs centré autour de lui. Nous allons étudier tous les échanges possibles avec un processeur i_0.

- Pour les liens de longueur 1 :

  Les PEs reliés par un chemin de longueur 1 sont ces quatre voisins directs possédant un lien de communications. On a $l_p = 1$. Comme il n'existe qu'un chemin entre chaque PE voisin et le processeur i_0, la formule donnant l'entrée $m_{ij}(\lambda)$ se simplifie en :

  $$m_{ij}(\lambda) = \lambda(1-\lambda)^{r_p}$$

  On peut calculer r_p en remarquant que le chemin emprunté possède une couleur c_1, et que n_0 le nombre d'arêtes incidentes du nœud i_0 dont la couleur est plus grande que c_1 est égal à $4 - c_1$. De même, n_{l_p} le nombre d'arêtes incidentes sur le nœud cible dont la couleur est plus petite que c_1 est égal à $c_1 - 1$.
  Ce qui donne :

  $$\begin{aligned} m_{ij}(\lambda) &= \textstyle\sum_{p\in\mathcal{P}_{ij}} (1-\lambda)^{r_p}\lambda^{l_p} \\ &= (1-\lambda)^{r_p}\lambda \\ &= (1-\lambda)^{4-c_1+c_1-1}\lambda \\ &= (1-\lambda)^3\lambda \end{aligned}$$

- Pour les liens de longueur 4 :

  Comme $l_p = 4$, on peut en déduire que le seul chemin existant est celui empruntant l'ensemble des couleurs de 4 à 1. Par conséquent $n_0 = n_i = n_{l_p} = 0$, la valeur de r_p est donc nulle.

  $$\begin{aligned} m_{ij}(\lambda) &= \textstyle\sum_{p\in\mathcal{P}_{ij}} (1-\lambda)^{r_p}\lambda^{l_p} \\ &= \lambda^4 \end{aligned}$$

- Pour les liens de longueur 2 :

  Pour un chemin de longueur 2, il n'est pas possible à partir du nœud i_0 de commencer à choisir la couleur 1. Seules les arêtes $4, 3, 2$ sont autorisées. On a $l_p = 2$.

Le chemin emprunté sera de la forme $(i = i_0, i_1; c_1), (i_1, i_{l_p}; c_{l_p})$. On peut en déduire que :

$$\begin{aligned} r_p &= n_0 + n_1 + n_{l_p} \\ &= (4 - c_1) + (c_1 - c_{l_p} - 1) + (c_{l_p} - 1) \\ &= 2 \end{aligned}$$

On a donc :

$$\begin{aligned} m_{ij}(\lambda) &= \sum_{p \in \mathcal{P}_{ij}} (1-\lambda)^{r_p} \lambda^{l_p} \\ &= (1-\lambda)^2 \lambda^2 \end{aligned}$$

Par un raisonnement similaire, on démontre qu'un chemin de longueur 3 est égal à $m_{ij}(\lambda) = (1-\lambda)\lambda^3$.

On a donc pour tout chemin, une entrée de la matrice correspondante égale à :

Longueur du chemin	valeur de m_{ij}	$\lambda = 1/2$
1	$(1-\lambda)^3\lambda$	$1/2^4$
2	$(1-\lambda)^2\lambda^2$	$1/2^4$
3	$(1-\lambda)\lambda^3$	$1/2^4$
4	λ^4	$1/2^4$

À partir du graphe de la figure IV.2, on obtient la matrice PDS suivante :

$$M(1/2) = 1/2^4 \times \mathbf{1}_{16\times 16}$$

où $\mathbf{1}_{16\times 16}$ est la matrice de taille 16×16 dont toutes les entrées sont égales à 1.

Le rang de la matrice est donc de 1, par conséquent $\gamma(M(1/2) = 0$, ce qui maximise la vitesse de convergence.

Soit $N = \sum_{i=1}^{n} w_i(t)$ (la charge totale), et $\bar{m} = N/n$ (la charge moyenne de la grille), pour toute charge initiale de vecteur W_0, on a :

$$MW_0 = \bar{m}W$$

□

5.4 Étude de l'algorithme X-Pavage sur une grille torique

Nous allons montrer dans cette partie que l'on peut définir des distances qui permettent dans le cas des grilles de taille $2^n \times 2^n$ d'obtenir une vitesse de convergence *optimale*.

L'algorithme *X-Pavage* se déroule de la façon suivante; dans un premier temps, l'algorithme se déroule comme l'algorithme *Pavage*, il y a partage de travail avec les processeurs voisins puis décalage du carré de rééquilibrage, dans un deuxième temps les processeurs échangent leur charge sur une fenêtre de taille plus importante. La figure IV.4 présente un exemple du rééquilibrage X-Pavage pour une grille torique de taille 8×8, les 4 premiers liens numérotés $1, 2, 3$ et 4 représentent le pavage classique, les liens 5 et 6 sont représentatifs du pavage étendu.

5.4.1 Définitions

Afin de qualifier la distance à laquelle l'échange de travail a lieu dans le cas du pavage étendu, nous introduisons la notion de distance.

Définition 8 *La distance entre deux processeurs i et j notée $d(i,j)$ est donnée dans une grille par la distance de Manhattan*

La distance (que l'on mesure dans notre cas soit horizontalement, soit verticalement) est le nombre de liens qu'il faut traverser en partant du processeur i pour atteindre le processeur j. Dans l'exemple de la figure IV.4, on a $d(18, 36) = 4$.

Les 4 premières couleurs (*cf.* algorithme *Pavage*) sont réservées pour les échanges de charge avec les processeurs immédiatement voisins, les couleurs suivantes sont utilisées pour les échanges avec les processeurs distants. On appellera par la suite les 4 premières couleurs utilisées pour les communications de voisinages, les couleurs **primaires**, les couleurs dont le numéro d'ordre est supérieur à 4 seront appelées les couleurs **étendues**. Les couleurs étendues provoquent des communications uniformes.

Le théorème suivant nous permet de construire pour les grille de taille $2^n \times 2^n$, un nombre de couleurs tels que l'on obtienne une vitesse de convergence optimale.

Définition 9 *Pour associer à une couleur une distance et les paires de processeurs correspondantes, nous construisons deux suite (H_n) et (V_n) qui indiqueront respectivement un échange dans la direction horizontale et dans la direction verticale à la distance (H_n) ou (V_n).*

Les suites ne sont construites que pour les valeurs de couleur strictement supérieures à 4.

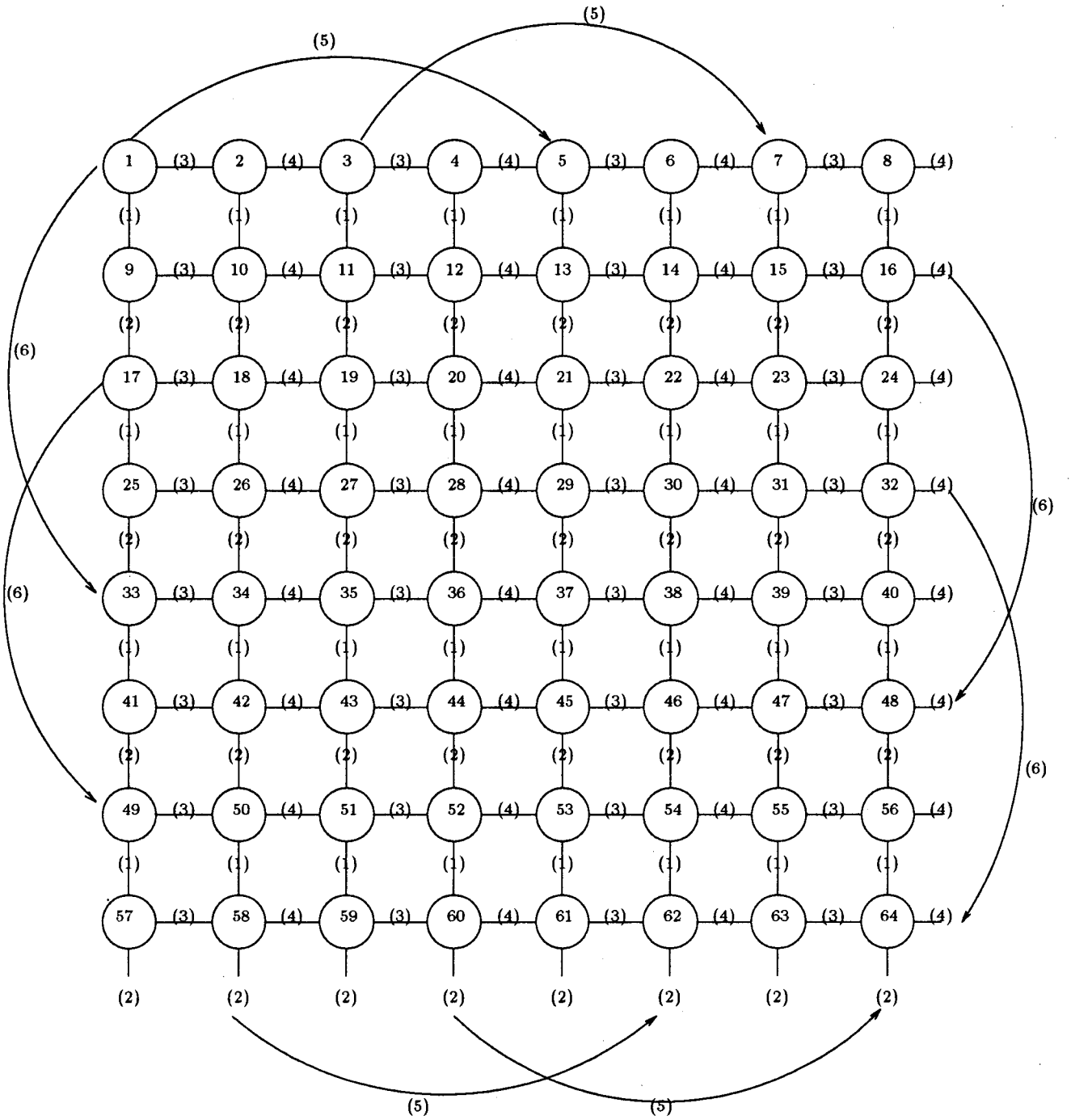


FIG. IV.4 - *Grille* 8×8

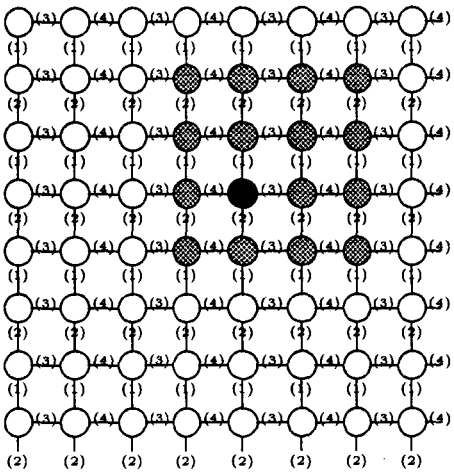


FIG. IV.5 - *Premier cas*

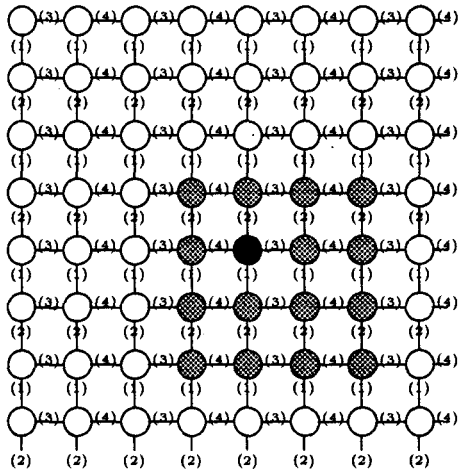


FIG. IV.6 - *Second cas*

$$\begin{cases} H_n = 2^{\frac{n-1}{2}} & \text{si } 4 < n \leq n(G_k) \text{ et } n \text{ impaire} \\ V_n = 2^{\frac{n-2}{2}} & \text{si } 4 < n \leq n(G_k) \text{ et } n \text{ paire} \\ H_n = 0 & \text{sinon} \\ V_n = 0 & \text{sinon} \end{cases}$$

5.4.2 Construction de la matrice d'équilibrage pour l'algorithme X-Pavage

Pour étudier la vitesse de convergence de la technique *X-Pavage*, nous allons, comme dans le cas de l'analyse de *Pavage* construire élément par élément la matrice $M(\lambda)$. Nous utiliserons la technique du chemin coloré pour calculer les différentes entrées de la matrice. Nous procéderons en deux étapes: nous montrerons l'unicité et l'existence du chemin entre deux processeurs quelconques du système puis nous calculerons l'entrée correspondante dans la matrice.

Lemme 4 *Avec la suite de couleurs définie dans la définition 9, il existe un chemin coloré entre deux processeurs quelconques.*

Preuve :

1. Si le processeur i appartient au voisinage du processeur j, c'est-à-dire que le processeur i est dans le carré 4×4 centré sur j, on sait qu'il existe un tel chemin et de plus qu'il est unique. Il existe en tout et pour tout que quatre cas différents (*cf.* figures IV.5 à IV.8).

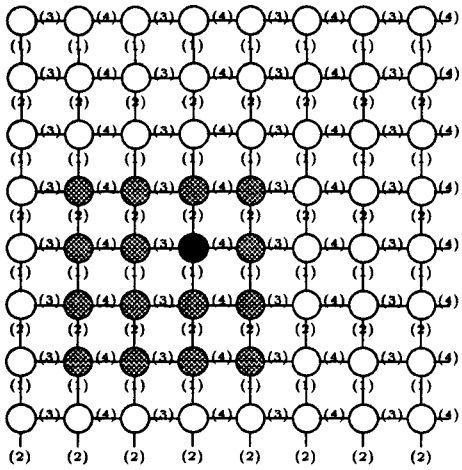


FIG. IV.7 - *Troisième cas*

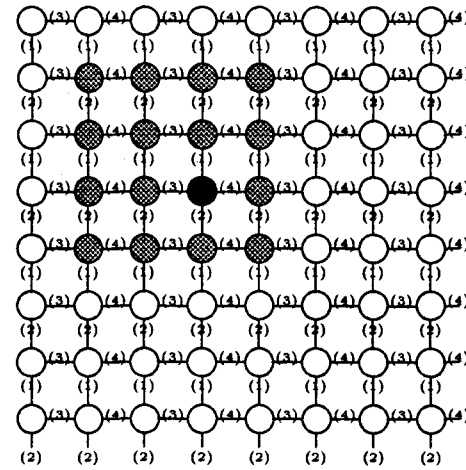


FIG. IV.8 - *Quatrième cas*

2. Si le processeur i n'appartient pas au voisinage du processeur j,
   alors on peut construire un chemin entre les processeurs i et j de la façon suivante: on partitionne la grille de processeurs en choisissant le carré de taille 4×4 (c'est l'un des quatre carrés dont j est le centre), ce qui est possible puisque l'on travaille sur des grilles carrées de taille $2^n \times 2^n$.

   À partir de tout processeur, en utilisant les suites H_n et V_n: $2^{n-1}, 2^{n-2}, \ldots, 4$ on peut se retrouver dans tout carré 4×4 et donc dans le carré du processeur cible, où l'on sait d'après la première partie de ce lemme que l'on atteint tout processeur avec une combinaison des couleurs primaires.

□

Lemme 5 *Si m_{ij} représente l'entrée de ligne i et de colonne j de la matrice $M(\lambda)$ de couleur, on a pour une grille de taille $2^n \times 2^n$:*

$$\forall i, j \quad m_{ij} = \lambda^{\delta(G_k)}$$

Preuve : D'après le lemme 3, on a pour tout i, $\delta(i) = delta(G_k)$ (le nombre d'arêtes incidentes est le même pour tout nœud), comme il n'existe pas de chemin de rebouclage pour les quatre couleurs primaires et que les couleurs étendues mènent à d'autres quadrants. On a donc :

$$m_{ii} = \lambda^{\delta(G_k)}$$

De même, on sait que $m_{ij} = \sum((1-\lambda)^{r_p}\lambda^{l_p})$, nous allons montrer qu'il existe un unique chemin entre deux nœuds i et j. L'existence d'un tel chemin a été prouvé dans le lemme précédent.

Par récurrence, supposons que le résultat soit vrai pour une machine de taille $2^n \times 2^n$, montrons qu'il est vrai pour une machine de taille $2^{n+1} \times 2^{n+1}$.

Travailler sur une machine $2^{n+1} \times 2^{n+1}$ revient à quadrupler la taille de la machine, on a donc affaire à 4 systèmes de taille $2n$. Les deux couleurs rajoutées permet de passer d'une machine $2^n \times 2^n$ à une des trois autres machines. On remarque que l'utilisation des deux couleurs rajoutées ne permettent pas de rebouclage. Si pour n il existe un unique chemin d'un processeur à un autre, il en est de même pour $n+1$.

Or, on a vu que pour une machine 4×4, soit $n = 2$, le chemin d'un processeur i quelconque à un processeur j est unique.

Par récurrence, on en déduit que pour tout n le chemin d'un processeur à un autre est unique. De plus, comme pour tout processeur $\delta(i) = \delta(G_k)$, on peut donc écrire:

$$\begin{aligned} m_{ij} &= \sum_{p \in \mathcal{P}_{ij}}((1-\lambda)^{r_p}\lambda^{l_p} \\ &= (1-\lambda)^{r_p}\lambda^{l_p} \\ &= \lambda^{r_p+l_p} \end{aligned}$$

Montrons que pour tout chemin, $m_{ij} = \lambda^{r_p+l_p} = \lambda^{\delta(G_k)}$.

Tout chemin coloré entre deux processeurs i et j est de la forme
$(i = i_0, i_1; c_1), (i, i_2; c_2), \ldots, (i_{l_p-1}, i_{l_p} = j; c_{l_p})$;
avec tous les nœuds intermédiaires distincts et $c_1 > c_2 > \ldots > c_{l_p}$.

Par définition, on a bien évidemment $l_p \leq \delta(G_k)$, de plus on $r_p = \sum_{s=0}^{l_p}(n_s)$.

On a:

$$\begin{aligned} n_0 &= \delta(G_k) - c_1 \\ n_{l_p} &= c_{l_p} - 1 \\ n_s &= c_s - c_{s+1} - 1 \end{aligned}$$

d'où on en déduit :

$$\begin{aligned} r_p &= \delta(G_k) - c_1 \\ &+ c_1 - c_2 - 1 \\ &+ c_2 - c_3 - 1 \\ &\vdots \\ &+ c_{l_{p-1}} - c_{l_p} - 1 \\ &+ c_{l_p} - 1 \\ &= \delta(G_k) + l_p \times (-1) \\ &= \delta(G_k) - l_p \end{aligned}$$

On en déduit facilement que :

$$m_{ij} = \lambda^{\delta(G_k)} = \lambda^{r_p + l_p}$$

□

5.4.3 Convergence et équilibrage parfait de X-Pavage

Nous avons calculé toutes les entrées de la matrice $M(\lambda)$, le théorème suivant permet de conclure sur la convergence de la méthode *X-Pavage*

Théorème 8 (Convergence de l'algorithme *X-Pavage***)** *Pour une grille de taille* $2^n \times 2^n$ *l'application d'un algorithme X-Pavage (c'est-à-dire l'utilisation des suites de couleurs* (H_n) *et* (V_n) *et des couleurs primaires) permet d'obtenir un équilibrage parfait.*

Ce théorème nous assure de l'existence de communications optimales entre les processeurs faisant converger le système vers un état stationnaire, mais il permet également de construire ce schéma de communications.

Par exemple, pour la figure IV.4, on utilise 6 couleurs ($n(G_k) = 6$), dont deux couleurs étendues. Les quatre premières couleurs sont réservées aux échanges de voisinage, les couleurs 5 et 6 sont définies par les suites H_n et V_n. On a $H_5 = 2^2$, soit un échange de données à une distance de 4 processeurs, $V_6 = 4$ qui se fera à la même distance que H_5 suivant la verticale.

On peut donc conclure le théorème, comme il existe un chemin entre deux processeurs quelconques i et j et tel que $m_{ij} = m_{ii} = \lambda^{\delta(G_k)}$, la matrice de couleur M sera donc égale à :

$$M = 1/2^n \times 1_{2^n \times 2^n}$$

avec $1_{2^n \times 2^n}$ la matrice de taille $2^n \times 2^n$ dont toutes les entrées sont égales à 1. De plus, d'après le théorème 4 la vitesse de converge dépend de la valeur propre sous-dominante. Le rang de la matrice étant de 1, on a donc $\gamma(M) = 0$, ce qui maximise la vitesse de convergence.

Ainsi, pour une grille torique de taille $2^n \times 2^n$, l'application d'une égalisation de travail aux distances définies par les suites (H_n) et (V_n) puis de la technique *Pavage* provoque une égalisation de la charge des PEs sur l'ensemble du système. Pour une telle grille $2 \times n$ échanges de charge sont nécessaires, ce qui correspond à $2\log_2(n)$ échanges pour une grille $n \times n$.

5.5 Comparaison entre l'équilibrage Pavage et X-Pavage

Dans cette section, nous allons nous attacher à déterminer les améliorations apportées par l'algorithme *X-Pavage* sur l'algorithme *Pavage*.

Théorème 9 *Dans une grille torique de taille $2n \times 2n$, le nombre de répétitions de l'algorithme Pavage afin de provoquer l'échange de travail entre deux processeurs quelconques est de $n/2$.*

Preuve : On a vu (*cf.* lemme 4) que pour un processeur donné, l'échange de travail est réalisé à l'intérieur d'un carré de taille 4×4 dont il est le centre. Si l'on suppose qu'un PE contient toute la charge de la grille, chaque PE appartenant à son carré de redistribution possédera une partie de sa charge. Si le processus de redistribution des données est réitéré, chaque processeur créera un nouveau carré recouvrant en partie de l'ancien zone d'échange. À chaque nouvel équilibrage, la surface couverte par la zone de distribution des données s'élargit de 2 processeurs dans les directions cardinales comme indiqué sur la figure IV.9.

Si un processeur contient toute la charge d'une grille torique de dimension $2n \times 2n$, comme la zone d'équilibrage est un carré de taille initiale 4×4 qui augmente sa taille de 2 PEs dans les directions cardinales, le nombre totale de phase d'équilibrage pour couvrir la grille est de :

$$\frac{2n}{4} = \frac{n}{2}$$

□

Ces résultats permettent de comparer les approches *Pavage* et *X-Pavage* sur des bases théoriques. Deux processeurs quelconques de l'algorithme *X-Pavage* échangent leur travail sur

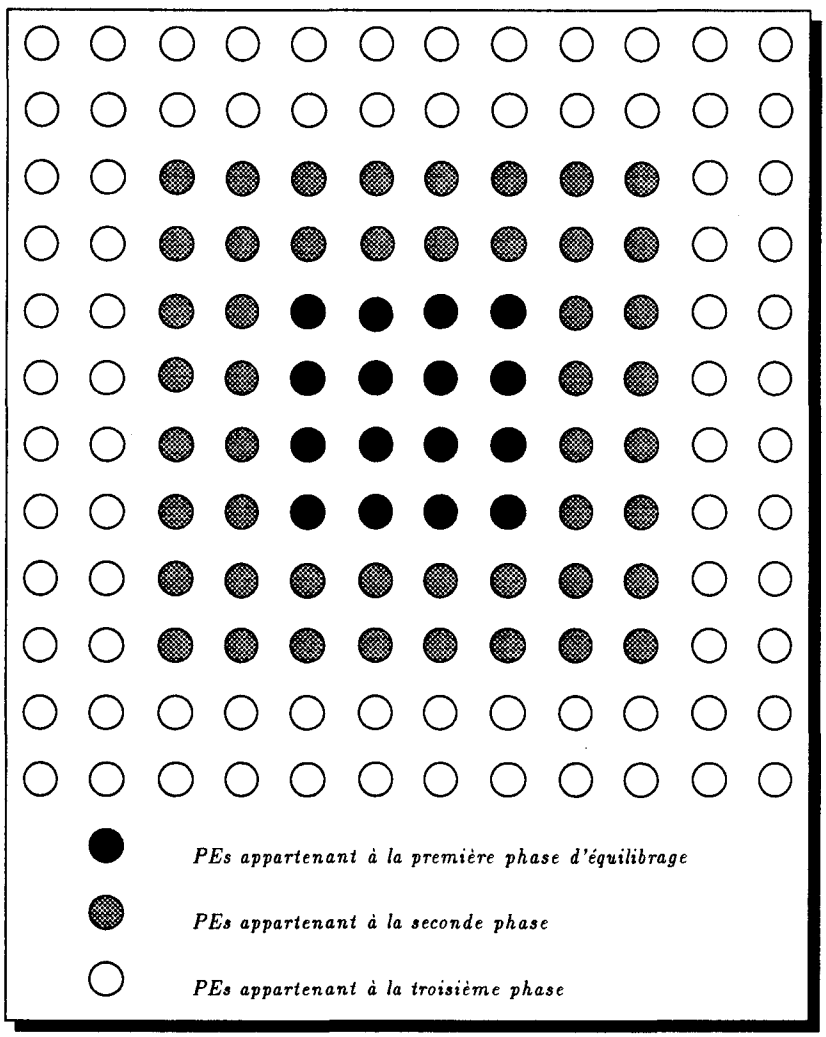


FIG. IV.9 - *Accroissement de la zone d'équilibrage pour l'algorithme Pavage*

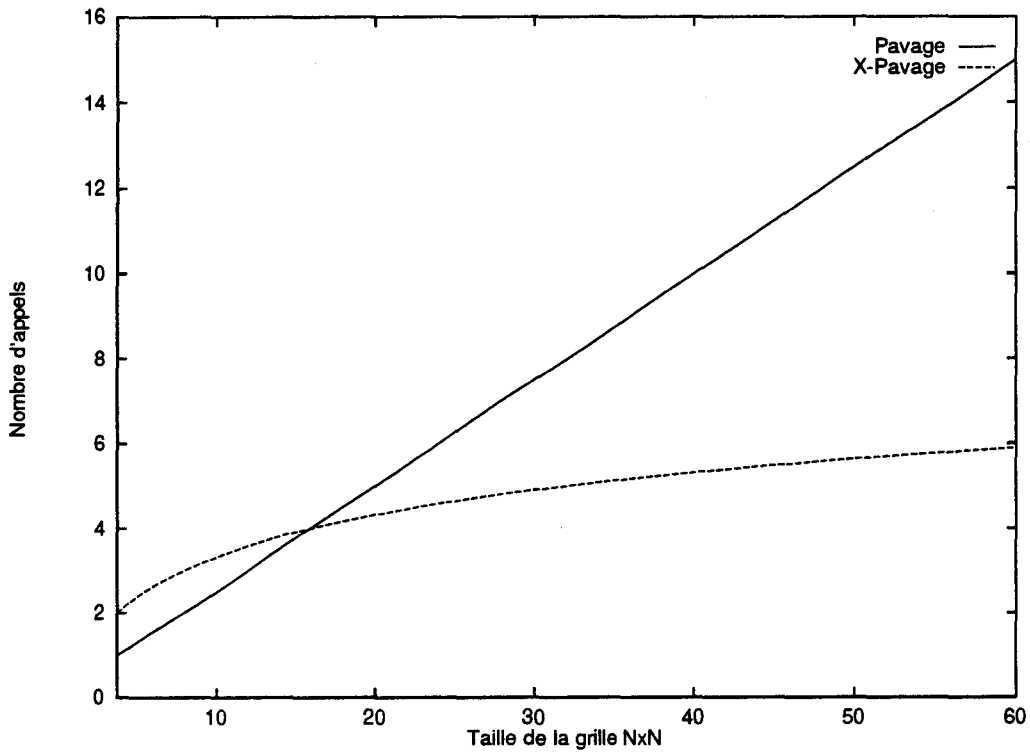


FIG. IV.10 - *Comparaison Pavage/X-Pavage*

une grille torique de dimension $n \times n$ en au maximum $2\log_2 n$ itérations d'après le théorème 8. Par contre, l'application de l'algorithme *Pavage* nécessite un nombre d'équilibrage au moins égal à $n/4$. La figure IV.10 présente conjointement le nombre d'itérations nécessaires pour l'algorithme *X-Pavage* et *Pavage*.

L'algorithme *X-Pavage* présente pour une grille torique de processeurs des performances théoriques supérieures à l'algorithme *Pavage* pour le nombre de communications effectuées. Il faut cependant que les communications uniformes ne possèdent pas un coût très supérieur aux communications de voisinage.

6 Algorithme du Glissement Pré-calculé

L'algorithme du *Glissement Pré-calculé* effectue un équilibrage en ne déplaçant que les données nécessaires pour atteindre l'équilibre sur chaque PE.

Théorème 10 *Si la charge totale N de la chaîne de PEs est un multiple du nombre de processeurs élémentaires n, alors l'algorithme du Glissement Pré-calculé effectue un équilibrage idéal.*

Preuve : L'algorithme du *Glissement Pré-calculé* calcule le nombre optimal de communications à travers la variable $transfert[k]$. Elle correspond au nombre *minimal* de communications qui passeront entre le lien des processeurs P_k et P_{k+1}. La différence entre les variables *but*[ ] et *somme*[ ] correspond juste au nombre à récupérer ou à envoyer afin d'atteindre l'équilibre.

Montrons que l'équilibre est atteint :

Si $transfert[k] > 0$, un transfert de données est réalisé des processeurs P_k vers P_{k+1}. Réciproquement, si $transfert[k] < 0$, l'échange des données se fera dans le de P_{k+1} vers P_k.

La charge du processeur P_k est égal à la différence des données reçues $transfert[k+1]$ et des données émises $transfert[k]$.

$$w'[k] = w[k] - transfert[k] + transfert[k+1]$$

Par définition, la valeur $transfert[k+1]$ est égale à :

$$\begin{aligned} transfert[k+1] &= (k+2)q - somme[k+1] \\ &= (k+1)q + q - somme[k] - w[k+1] \\ &= transfert[k] + q - w[k+1] \end{aligned}$$

En reportant la valeur de $transfert[k+1]$, dans l'équation donnant la charge du processeur P_k, on obtient :

$$\begin{aligned} w'[k] &= w[k] - transfert[k] + transfert[k+1] \\ &= w[k] - transfert[k] + transfert[k] + q - w[k+1] \\ &= q \end{aligned}$$

□

Sur un anneau, dans le cas le plus défavorable, toutes les données sont contenues par un unique PE. Le nombre d'étapes nécessaire est alors égal au nombre de processeurs comme la technique *Râteau*.

TAB. IV.1 - *Paramètres nécessaires pour l'estimation du coût des algorithmes*

Paramètres	Significations
t_v	Coût d'une communication de voisinages
t_u	Coût d'une communication uniforme
t_g	Coût d'une communication globale (permutation)
t_r	Coût d'une réduction ($O(\log(n))t_u$)
t_s	Coût pour un tri ($O(\log^2(n)t_u$)

TAB. IV.2 - *Coût en communications de différentes techniques d'équilibrage sur une grille* $n \times n$

Algorithme	Coût d'une exécution	Coût pour la convergence
Central	$3t_r + 2t_g + \alpha t_g$	Ne converge pas
Rendez-vous	$t_r + 2t_s + 2t_g + \alpha t_g$	$2\log_2(n)(t_r + 2t_s + 2t_g + \alpha t_g)$
Râteau	$t_r + 2\alpha n t_v$	$(t_r + 2\alpha n t_v)$
Pavage	$4t_v + 4\alpha t_v$	$> n(2t_v + 2\alpha t_v)$
X-Pavage	$4t_v + 4\alpha t_v + 2(\log_2(n) - 2)(t_u + \alpha t_u)$	$4t_v + 4\alpha t_v + 2\log_2(n)(t_u + \alpha t_u)$
Glissement	$< t_r + 2\alpha n t_v$	$< (t_r + 2\alpha n t_v)$

7 Calcul du coût en communications des algorithmes

Cette section s'attache à estimer le coût en communications des différents algorithmes présentés dans ce chapitre. Le tableau IV.1 présente les différents paramètres. Le paramètre α représente la quantité de charge échangé entre deux processeurs. Cette valeur n'est pas fixe.

Le tableau présente le coût des techniques en communications et conjointement le nombre total pour atteindre la convergence.

8 Conclusion

D'après l'analyse théorique menée pour les algorithmes *Rendez-vous*, *Râteau*, *Pavage* et *X-Pavage*, on peut en déduire les faits suivants :

- l'algorithme *Rendez-vous* réalise un équilibrage idéal pour un coût en communications en $O(\log_2(n))$
- l'algorithme *Râteau* a un coût qui dépend directement du nombre de processeurs dans chaque dimension. Sur une grille 2D, un rééquilibrage *Râteau* est réalisé en effectuant

TAB. IV.3 - *Nombre d'échanges de charge nécessaires pour atteindre l'équilibre sur une grille de taille $n \times n$*

Algorithme	Nombre d'échanges	Type
Rendez-vous	$\sim 2 \times \log_2(n)$	Globale
Râteau	$\sim 2 \times n$	Locale
Pavage	$> n/2$	Locale
X-Pavage	$\sim 2 \times \log_2(n)$	Uniforme
Glissement	$< 2 \times n$	Locale

deux fois la technique de base, une fois dans chaque dimension. Son coût est donc proportionnel au nombre moyen de processeurs dans chaque dimension.

Pour un nombre de processeurs donné, on peut considérer que l'accroissement de la charge des PEs ne provoquera qu'une faible augmentation du sur-coût de l'algorithme de rééquilibrage. Cette particularité s'explique par la dépendance directe du coût du rééquilibrage au nombre de processeurs.

L'algorithme *Râteau* est une borne supérieure de l'algorithme du *Glissement Précalculé* ;

- les algorithmes *Pavage* et *X-Pavage*

  D'après l'analyse effectuée, on peut de manière intuitive supposer que l'extensibilité de l'algorithme *Pavage* sera plus faible que la technique *X-Pavage*. En effet, le nombre de phases nécessaires pour réaliser un équilibrage parfait sur une grille de taille $n \times n$ est en $O(\log_2(n))$ pour *X-Pavage* et supérieur à $O(n)$ pour la technique *Pavage*. Si la taille de la grille augmente, le sur-coût provoqué par la technique *Pavage* augmentera de façon plus importante que celui provoqué par *X-Pavage*.

  On peut déduire également de l'analyse que la fréquence d'équilibrage de *Pavage* sera supérieure à celle de *X-Pavage*.

Le tableau IV.3 présente le nombre de communications générées (nombre total d'appel aux fonctions **send** et **receive**) pour atteindre l'équilibrage sur une grille de taille $n \times n$. On constate que les techniques *Rendez-vous* et *X-Pavage* ont des coûts logarithmiques. À noter que *Rendez-vous* permet d'obtenir une convergence plus rapide que *X-Pavage* lorsque seul un petit nombre de processeurs sont inactifs et espacés d'une distance impaire.

Chapitre V

Analyse de performances

1 Présentation

Nous présentons dans ce chapitre les diverses expérimentations qui ont été réalisées afin d'estimer les performances respectives des différentes techniques d'équilibrage dynamique. La machine à notre disposition pour tester les diverses méthodes est une MasPar de chez Digital Equipment Corporation que nous décrivons brièvement dans les paragraphes suivants. Nous avons testé les divers algorithmes sur l'ensemble de Mandelbrot. C'est un problème qui possède de bonnes propriétés qui en font un bon candidat pour l'analyse de performances.

Afin d'apprécier les nombreuses mesures obtenues, nous utilisons certaines notions classiques comme le gain , d'autres dédiées à l'estimation de l'équilibrage dynamique : la qualité et la fréquence d'équilibrage. Nous visualiserons les expériences effectuées par une série de courbes 3D présentant l'évolution temporelle des différentes techniques de redistribution des données. Nous présenterons ensuite certains résultats expérimentaux afin de mieux analyser le rééquilibrage et son comportement asymptotique et d'affiner les résultats présentés par les courbes.

2 Présentation de l'architecture cible

La machine sur laquelle nous avons portée nos différentes techniques d'équilibrage est une MasPar MP-1 [Bla90] construite par Digital Equipment Corporation. C'est une représentante typique des machines SIMD, l'ensemble des processeurs élémentaires est organisé sous la forme

d'une grille torique. Elle est composée des éléments suivants :

Une machine hôte ou frontal dont le rôle principal est de servir d'interface entre l'utilisateur et la MasPar proprement dite. La machine est une station de travail pilotée par un système d'exploitation UNIX. L'utilisateur écrit et compile ses programmes sur cette machine, les binaires sont ensuite envoyés vers le séquenceur.

Un séquenceur dont la dénomination exacte est ACU (Array Control Unit) exécute séquentiellement le programme reçu du frontal. L'assembleur de la MasPar distingue deux type d'instructions : les instructions scalaires qui sont exécutées uniquement par l'ACU et les instructions parallèles qui sont transmises à la grille de processeurs élémentaires.

Un tableau de processeurs élémentaires organisé sous la forme d'une grille torique. Il est possible de disposer d'un nombre de PEs variant entre 1024 et 16384. Le séquenceur a le contrôle d'un masque de bits (1 par PE) qui peut placer les PEs dans 2 états : soit oisif, soit actif.

Deux réseaux de communications composés d'un réseau spécialisé dans les communications de voisinage et dans les communications uniformes et d'un cross-bar permettant une communication entre deux PEs quelconques.

La MasPar MP-1 supporte deux langages de programmation : un langage dérivé de FORTRAN appelé MP-FORTRAN [Dig92a] dont le parallélisme est exprimé de manière implicite. L'utilisateur n'a pas à se soucier de l'allocation des données sur les processeurs élémentaires ni des communications induites par les affectations et les calculs. L'autre langage est un dérivé du langage C, MPL [Dig92b] dont le parallélisme est décrit explicitement. MPL reprend les mots-clefs du langage C et incorpore un certain nombre de déclarations, commandes et variables dédiées à l'expression du parallélisme de données. Ainsi, le mot-clef `plural` permet la déclaration d'une variable contenue dans chacun des PEs. Nos différents algorithmes ont tous été développés en langage MPL.

La machine à 16k processeurs permet d'obtenir une puissance de 473 MégaFLOPS pour le calcul en double précision [RO93].

Vue l'importance des communications dans nos expériences, nous décrivons de manière plus détaillée dans les paragraphes suivants les principales caractéristiques de ces deux systèmes de communication.

2.1 Le réseau de voisinage

Chaque processeur élémentaire est relié de manière directe à ses 8 voisins situés dans les directions cardinales (N,S,E,O,NE,NO,SE,SO). Le réseau de voisinage appelé Xnet permet à tout PE de communiquer avec tout PE situé dans une des directions cardinales.

2.2 Le réseau global

Le réseau global ou Global Router permet à travers un cross-bar de trois étages d'effectuer une communication entre deux processeurs situés à tout endroit sur la grille. Chaque ensemble de 16 PEs ou grappe groupé sous une forme 4×4, ne possède qu'un lien vers le réseau multi-étagé. Cette conception du réseau global entraîne des conflits si un envoi de données est effectué sur ou par deux PEs appartenant à la même grappe.

2.3 Le réseau de voisinage est il plus performant que le réseau global?

La vitesse des communications pour le réseau de voisinage et le réseau global dépend d'un certains nombre de paramètres [Gav92, Pre92].

Pour le réseau global:

- la taille des données à transmettre;
- le nombre de conflits. Le routeur permet une communication globale en parallèle avec plusieurs PEs. Une partie des communications est réalisée séquentiellement dépendant du nombre de *conflits* engendrés. Les conflits étant proportionnels au nombre de PEs d'un même cluster souhaitant envoyer des données dans ce même cluster ou aux PEs d'un autre cluster. Lire ou écrire k fois dans le même PE provoquera également k conflits. Les techniques de rééquilibrage n'engendrent pas de conflits d'accès sur le même PE, mais des conflits provoqués par l'émission ou la réception sur un même cluster sont possibles.

Pour le réseau de voisinage:

- la taille des données à transmettre;
- la distance à laquelle les données sont envoyées.

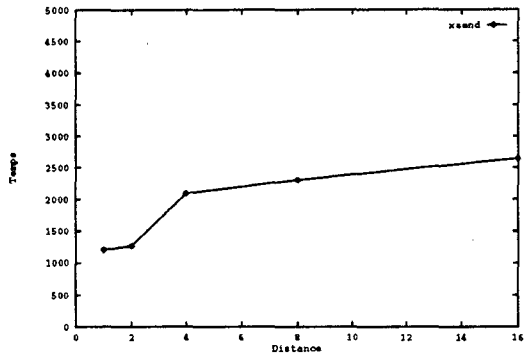


FIG. V.1 - *Coût d'une communication par le réseau de voisinage*

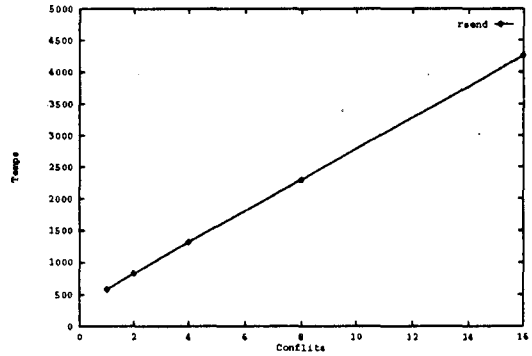


FIG. V.2 - *Coût d'une communication par le Global Router*

Il est difficile de comparer véritablement les performances des deux systèmes de communications, puisque leurs performances respectives sont dépendantes de critères différents. On peut cependant tenter de les comparer pour un envoi de données de 8 octets. Les figures V.1 et V.2 présente le temps nécessaire pour transmettre 8 octets par le réseau de voisinage et le réseau global. On voit que les résultats ne sont pas tranchés et qu'aucune des deux approches n'est vraiment supérieure à l'autre. On peut cependant remarquer qu'une communication *uniforme* effectuée à une distance supérieure à 4 n'est guère plus coûteuse qu'une communication effectuée à une distance de 4.

3 Présentation du problème de l'ensemble de Mandelbrot

3.1 L'ensemble de Mandelbrot

L'ensemble de Mandelbrot est un bon exemple pédagogique pour l'introduction à la théorie du chaos [Dew85]. Lorsque l'on soumet des nombres du plan complexe à une certaine opération indéfiniment répétée, on obtient pour certaines parties du plan des comportements totalement imprévisibles.

La construction de l'ensemble de Mandelbrot est la suivante :

On choisit un nombre complexe c, et on calcule l'expression $z^2 + c$, où z est un nombre complexe variable. Donnons à z la valeur initiale 0 : $z^2 + c$ se réduit à c. Remplaçons alors z par c dans z^2 : nous obtenons $c^2 + c$. Remplaçons z par cette nouvelle valeur pour obtenir $(c^2 + c)^2 + c$. On poursuit alors ce processus, chaque nouvelle valeur de la somme $z^2 + c$ étant prise comme nouvelle valeur de z.

$$z = z^2 + c \tag{V.1}$$

Cette opération est répétée tant que le module de z^2+c reste inférieur à 2. Pour

obtenir, l'ensemble de Mandelbrot, il suffit d'effectuer ces opérations sur le plan complexe. On affecte alors une couleur à chaque point fonction du nombre d'itérations nécessaires pour atteindre la convergence. On obtient alors un structure étonnante de complexité, de variété et de beauté (*cf.* figure V.3).

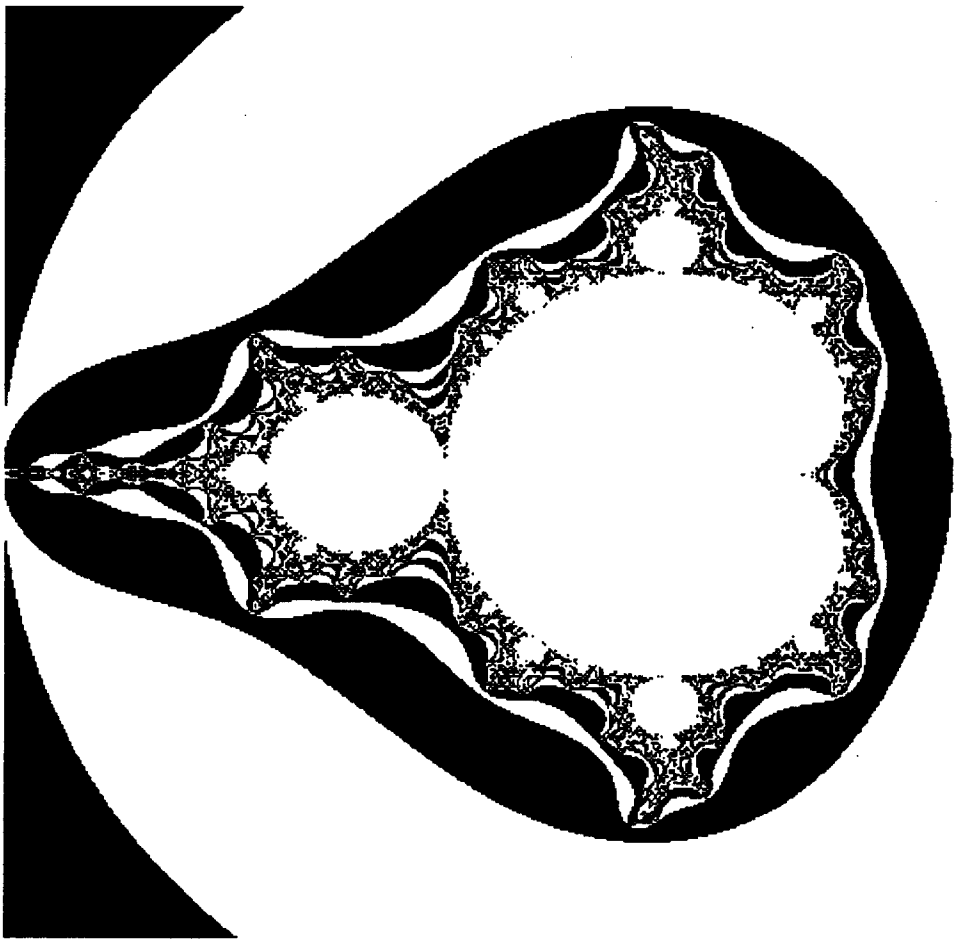


FIG. V.3 - *Ensemble de Mandelbrot*

3.2 Motivations

Le problème de l'ensemble de Mandelbrot a été sélectionné afin de comparer les performances des algorithmes de rééquilibrage pour plusieurs raisons :

1. Le test de nos différents algorithmes requiert un exemple dont l'exécution soit rapide et *reproductible* temporellement. L'ensemble de Mandelbrot satisfait ces deux propriétés.
2. L'algorithme de Mandelbrot est représentatif de nombreux problèmes impliquant le même calcul sur différents points possédant des taux de convergence différents, comme

la résolution d'équations différentielles ou le calcul par éléments finis.

3. Des algorithmes plus « classiques » comme la recherche en parallèle n'ont pas été choisis, car ils présentent la particularité d'être très sensibles aux mécanismes de redistribution. Il devient alors malaisé de comparer des techniques d'équilibrage qui provoquent des anomalies comme la super-linéarité. La super-linéarité se rencontre par exemple dans le parcours d'arbre en parallèle. L'algorithme séquentiel parcourt l'arbre par exemple suivant la branche la plus gauche. Si la solution se trouve dans une branche de droite, la solution n'est trouvée qu'au bout d'un nombre important d'itérations. La parallélisation du parcours d'arbre peut provoquer une exploration beaucoup plus rapide suivant le mécanisme de projection des nœuds sur les processeurs. On obtient un gain plus important que le nombre de processeurs qui n'est pas représentatif du gain dû à la parallélisation.

4. L'ensemble de Mandelbrot présente l'immense avantage d'être facilement extensible. Lorsque l'on souhaite comparer un algorithme d'équilibrage dynamique sur des machines possédant n puis $\alpha \times n$ processeurs, il est nécessaire de multiplier le travail par le coefficient α. Cet accroissement du travail n'est pas toujours facile à réaliser du fait de la méconnaissance du travail engendré. Par contre, l'ensemble de Mandelbrot peut être facilement adapté sur des machines possédant un nombre plus ou moins important de processeurs en modifiant la définition de l'image souhaitée. Par exemple sur une machine de n processeurs, une image dont la précision souhaitée est de n_x sur n_y sera transformée en une image $\frac{\alpha}{2}n_x \times \frac{\alpha}{2}n_y$ pour un ensemble de α PEs. On conserve sensiblement un travail « identique » multiplié par un coefficient α.

5. L'ensemble de Mandelbrot est un exemple typique de calcul dont le comportement de chaque point est imprévisible et dont le taux de convergence peut varier de une itération à plusieurs centaines pour deux données contigües.

6. la partie calculatoire de résolution d'un tel problème est très faible en regard du coût des communications. Chaque cycle de calcul ne comporte que le calcul d'une élévation au carré et d'une addition. La plupart des autres algorithmes data-parallèles ont une partie calculatoire plus importante. Tout gain obtenu pourra ainsi être répercuté avec avantage sur d'autres problèmes nécessitant l'utilisation de techniques de rééquilibrage dynamique.

3.3 Parallélisation du calcul de l'ensemble de Mandelbrot

La méthode la plus simple pour paralléliser la résolution de l'ensemble de Mandelbrot sur une grille de processeurs est d'effectuer une partition du plan de données comme indiquée sur la figure V.4. Chaque processeur élémentaire aura la responsabilité du traitement d'un nombre plus ou moins important de points suivant la taille du domaine et la définition souhaitées.

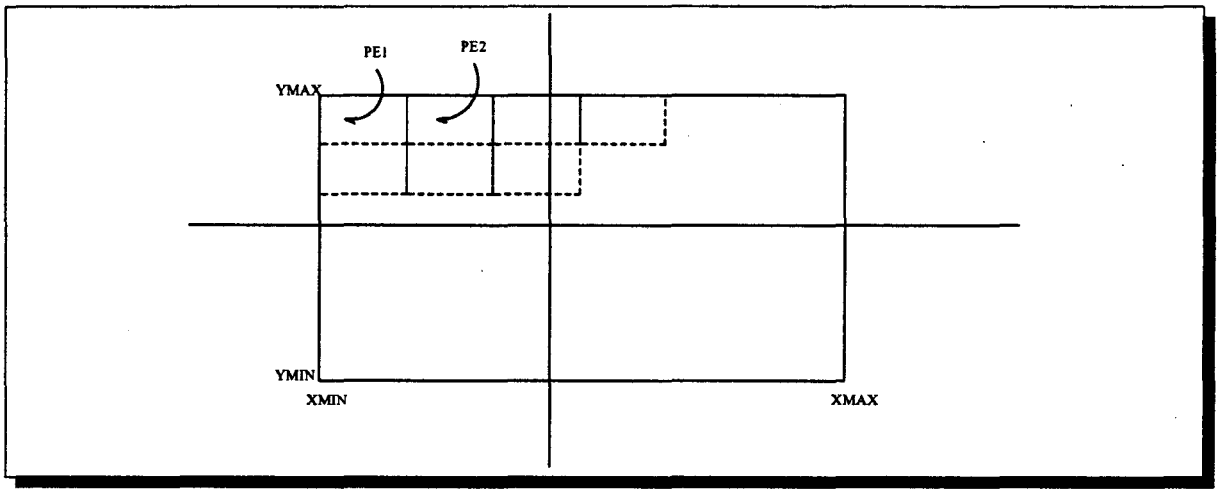


FIG. V.4 - *Découpage du plan pour la parallélisation de l'ensemble de Mandelbrot*

L'algorithme utilisé réalise le calcul indiqué par l'équation V.1 sur l'ensemble des PEs. Un test de convergence globale est effectué suivant une fréquence fixée à la compilation. Les points ayant effectivement convergé sont remplacés par d'autres données, les processeurs dont les points n'ont pas atteint la convergence continuent le même calcul. Comme certains points du plan complexe ne convergent jamais, une borne supérieure est imposée. Ce maximum dépassé, le point suivant contenu par le processeur est considéré. On constate que cette parallélisation « naïve » amène un gain peu élevé. En effet, le nombre d'itérations nécessaires à sa terminaison est pratiquement toujours égal au nombre maximum d'itérations multiplié par le nombre de points contenu par un processeur. Ceci s'explique par la mauvaise distribution spatiale des taux de convergence, certains processeurs ayant la charge des points dont la convergence effective n'est jamais atteinte. Au bout de quelques itérations, la majorité des processeurs ont épuisé toute leur charge et deviennent inactifs, attendant que les quelques PEs actifs finissent leur travail.

3.4 Paramétrisation du problème

Afin d'éviter l'intervention d'un nombre trop important de paramètres, nous avons pour notre étude fixé certaines variables. Comme certains points du plan complexe ne convergent jamais, une limite au nombre d'itérations appelée *Maxiter* est choisie. Empiriquement, il semble qu'imposer une limite de 100 permet d'obtenir des résultats corrects, car si la convergence n'a pas été obtenue après 100 cycles, les probabilités sont très élevées qu'elle ne soit jamais atteinte.

Le test de convergence effective de l'ensemble de la grille de processeurs élémentaires et le chargement de nouveaux points pour les PEs ayant convergé ne peuvent être réalisés

toutes les itérations. Cela nécessiterait en effet un sur-coût trop important. Nous avons fixé arbitrairement ce test global de convergence au cinquième du nombre maximum d'itérations (*Maxiter*).

L'estimation de l'état du système en vue de provoquer un réarrangement des données ne peut être réalisée à tous les cycles de calcul. La politique de déclenchement utilisée est périodique à seuil. Nous avons fixé de manière arbitraire une possibilité de déclenchement de l'équilibrage toutes les 50 itérations. Cette valeur relativement élevée a été choisie pour ne pas pénaliser le système dans des mécanismes de déclenchement appelés trop souvent. Pour l'ensemble des tests sur les algorithmes, le mécanisme réalise une redistribution des données si le nombre de processeurs élémentaires devient inférieur à 90%.

Le réseau de communications locales de la MasPar MP-1 peut pour les techniques *Voisinage* ou *X-Voisinage* être configuré pour un ensemble de 4 ou 8 voisins. Notre choix s'est limité aux échanges pour une topologie à 4 voisins qui est un bon compromis entre l'amélioration de la diffusion de l'équilibrage et la séquentialité des échanges sur chacun des voisins.

4 Analyse de l'équilibrage dynamique

Mesurer les performances effectives d'une application parallèle n'est pas un problème trivial. Divers paramètres entrent en ligne de compte, le temps de calcul, le nombre d'itérations, le temps passé durant les phases de redistribution des données, l'utilisation des processeurs... Pour estimer le comportement de nos techniques d'équilibrage, nous avons retenu plusieurs critères.

Le gain algorithmique Il représente le gain *brut* obtenu après parallélisation d'un algorithme. Il correspond au rapport du nombre d'itérations en séquentiel au nombre d'itérations en parallèle.

Le gain d'exécution Il modère les valeurs du gain « algorithmique » en tenant compte du sur-coût provoqué par le rééquilibrage. Un algorithme peut présenter un gain important en diminuant le nombre d'itérations nécessaires à sa terminaison tout en étant gâché par un coût important d'équilibrage.

La qualité Elle indique le pourcentage de temps ou d'itérations gagné par rapport au même algorithme réalisé sur la même machine et dans les mêmes conditions mais qui ne bénéficie pas de l'équilibrage.

la fréquence de l'équilibrage Cette mesure permet de connaître le nombre d'appels aux routines de réarrangement dynamique.

4.1 Le gain algorithmique

Afin d'estimer l'accroissement de performance dû à la parallélisation d'un algorithme séquentiel, on calcule généralement le rapport de la durée d'exécution du *meilleur* algorithme séquentiel sur le temps d'exécution de l'algorithme parallèle. Cette mesure permet de comparer le gain effectif du parallélisme au nombre de processeurs. Si le nombre de processeurs élémentaires disponibles sur la machine est n, le gain théorique maximum est de n. Malheureusement comme l'explique [HB84], les performances du système se dégradent avec le nombre de PEs.

L'unité de base dans la résolution de l'ensemble de Mandelbrot correspond à une étape de calculs d'un point représenté par l'équation V.1. Le nombre total d'étapes de calcul nécessaires à la terminaison de l'algorithme sur une machine mono-processeur divisé par le nombre d'itérations en parallèle représente le gain algorithmique (une itération correspondant à l'exécution parallèle d'une étape de calcul sur l'ensemble des processeurs élémentaires).

Le gain algorithmique sur le nombre d'itérations est égal à :

$$S_I = \frac{I_1}{I_n} \tag{V.2}$$

où I_1 est le nombre d'itérations pour la résolution du problème sur un système mono-processeur et I_n est le nombre d'itérations pour le même problème sur un ensemble de n processeurs.

4.2 Le gain d'exécution

La mesure du gain présentée par la formule V.2 présente un inconvénient majeur dans la mesure de performance. Du fait de sa définition qui s'appuie sur le nombre d'étapes de calcul et d'itérations au lieu de la durée d'exécution , le temps passé dans les phases d'équilibrage est occulté. La phase de redistribution des données est absente dans le décompte du nombre d'itérations. Comme Mahanti et *al* [MD93], nous modifions légèrement la définition du gain afin qu'elle puisse refléter le temps dépensé durant les phases d'équilibrage.

Soit T_{lb} le temps total passé pendant l'équilibrage, le gain peut être écrit de la manière suivante :

$$
\begin{aligned}
S_{\text{lb}} &= \frac{T_1}{T_n + T_{\text{lb}}} \\
&= \frac{T_1}{T_n} \times \frac{1}{1 + \frac{T_{\text{lb}}}{T_n}} \\
&= S_I \times \frac{1}{1 + \frac{T_{\text{lb}}}{T_n}}
\end{aligned}
$$

4.3 La qualité

À la différence des opérateurs précédents qui permettent une comparaison du système parallèle par rapport à une machine séquentielle, la qualité compare l'algorithme incluant les routines de redistribution des données avec la même exécution sur le même système sans autoriser l'équilibrage dynamique. La qualité correspond au ratio du nombre d'itérations parallèles sans équilibrage au nombre d'itérations parallèles avec équilibrage.

Formellement, la qualité s'écrit sous la forme :

$$Q = \frac{I_{\text{sans équilibrage}}}{I_n}$$

Comme pour le cas du gain algorithmique, cette définition ne prend pas en compte la durée de l'équilibrage. Par conséquent, nous introduisons la qualité d'équilibrage incluant les pertes occasionnées :

$$Q_{\text{lb}} = \frac{T_{\text{sans équilibrage}}}{T_n}$$

4.4 Fréquence de l'équilibrage

Les critères de gain algorithmique et gain d'exécution ne permettent pas d'apprécier la fréquence de l'équilibrage, paramètre important, en particulier pour les problèmes dont le nombre d'étapes de calculs est important. Dans de tels cas, il est primordial de réaliser l'équilibrage le moins souvent possible même à un coût plus élevé. Le calcul de la fréquence d'équilibrage permet d'estimer au bout de combien d'itérations une redistribution des données est réalisée.

Dans notre cas, La fréquence de l'équilibrage est calculée par rapport à une base 10000. Le nombre total d'itérations est ramené à 10000. Formellement :

$$f_{\text{eq}} = \frac{m}{I_n} \times 10000$$

où m est le nombre d'équilibrages effectués

4.5 Une fonction indiquant le déséquilibre

Nous avons introduit la fonction de déséquilibre f_{des} qui permet d'estimer l'irrégularité de la répartition des charges sur le système. Cette fonction est calculée pour un exemple où l'équilibrage dynamique est absent. Elle évalue la différence pour chaque processeur entre le nombre de cycles de calculs effectués et le nombre moyen de cycles. Elle correspond à la variance du nombre d'itérations.

Si x_i est le nombre d'itérations effectuées pour le PE numéro i, $\bar{x}$ le nombre moyen d'itérations, la variance est définie par :

$$V = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}$$

On a :

$$f_{\text{des}} = \frac{\sqrt{V}}{\bar{x} \times \sqrt{n}}$$

Le déséquilibre est le plus important lorsqu'un processeur possède l'ensemble de la charge du système et que les autres PEs sont inoccupés. Soit i_0 ce processeur, et $X = x_{i_0}$ la charge totale du système, on a :

$$V = \frac{(X - \frac{X}{n})^2}{n} \sim \frac{X^2}{n}$$

Ce qui correspond à une fonction f_{des} :

$$f_{\text{des}} \sim \frac{X}{\sqrt{n}} \times \frac{n}{X \times \sqrt{n}} = 1$$

Une valeur de 1 pour la fonction f_{des} indiquant un déséquilibre complet où un processeur possède l'intégralité de la charge de la grille. Une valeur de 0 indique au contraire un équilibre parfait (la variance est nulle), chaque PE possédant la charge moyenne de la grille.

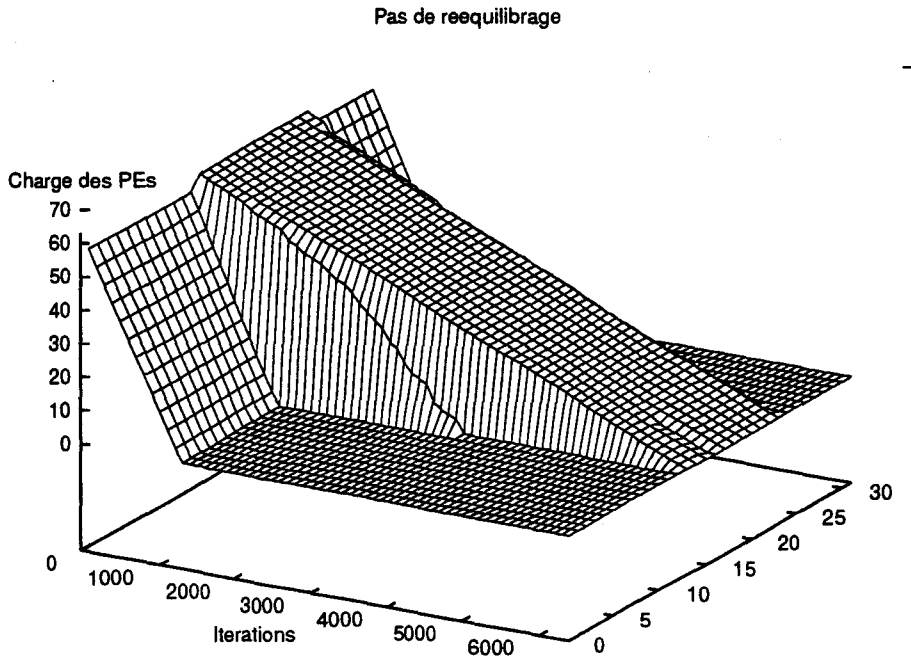


FIG. V.5 - *Résolution de l'ensemble de Mandelbrot sans équilibrage*

$$I_n = 6400$$
$$S_I = 300$$

5 Résultats expérimentaux

Les résultats présentés dans cette section correspondent à de nombreuses résolutions de l'ensemble de Mandelbrot sur une MasPar MP-1 de 1k à 16k PEs. Nous avons conservé une déséquilibre constant pour toutes les expérimentations. La fonction f_{des} étant constante, les algorithmes peuvent être comparés sur une base commune. Nous analyserons suivant les différents critères introduits précédemment l'équilibrage dynamique sur une machine SIMD. Nous étudierons l'influence de la charge pour chaque algorithme pour un nombre donné de processeurs. Le nombre de calculs I_1 correspond au nombre d'itérations nécessaires à la terminaison de l'ensemble de Mandelbrot sur un système mono-processeur.

La figure V.5 présente sur une machine de 1k l'évolution de la charge pour la résolution de l'ensemble de Mandelbrot pour un ensemble de processeurs élémentaires représentatifs (diagonale de la grille) numéroté de 0 à 31. On constate qu'au bout de quelques itérations, près des 2/3 des processeurs deviennent oisifs. Ils doivent attendre le tiers restant des processeurs pendant près de 6000 itérations.

5.1 Évaluation du coût

Les figures suivantes comparent le gain algorithmique dû à la parallélisation par rapport au gain d'exécution S_{lb}. On a bien entendu :

$$S_{\text{lb}} \leq S_I$$

L'égalité n'étant vérifiée que pour un équilibrage à coût nul ! La « différence » entre les deux courbes donnant une estimation du coût de la redistribution des données.

Algorithme Râteau

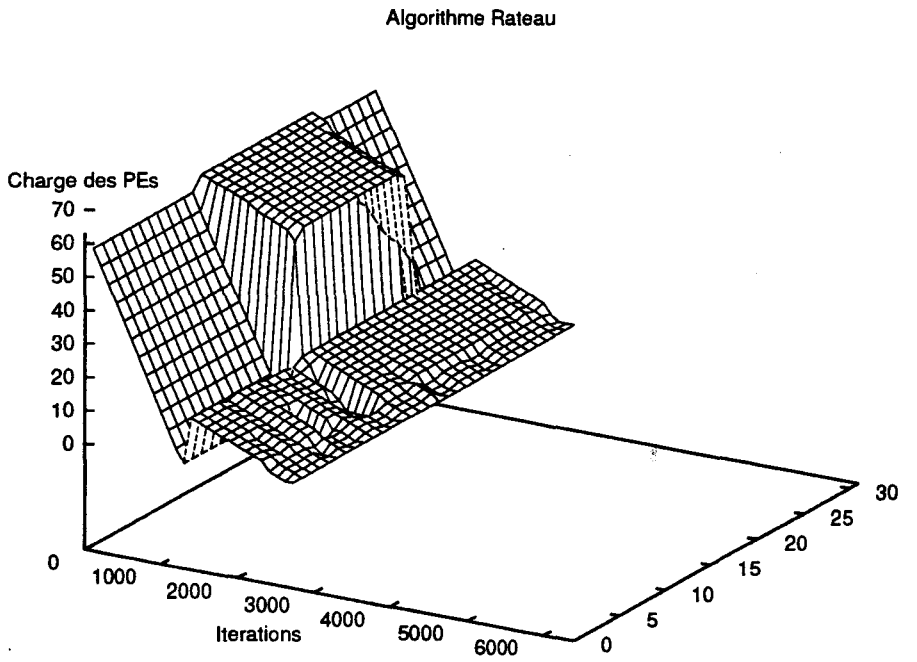


FIG. V.6 - *Rééquilibrage à l'aide de l'algorithme Râteau*

$$I_n = 2780 \quad m = 3$$
$$S_I = 719 \quad S_{\text{lb}} = 600$$
$$Q = 2.3 \quad Q_{\text{lb}} = 1.7$$

Le rééquilibrage effectué par l'algorithme *Râteau* présenté en figure V.6 montre la puissante redistribution proposée par la technique *Râteau* sur toute la grille. Peu d'itérations sont nécessaires pour la terminaison de l'ensemble de Mandelbrot.

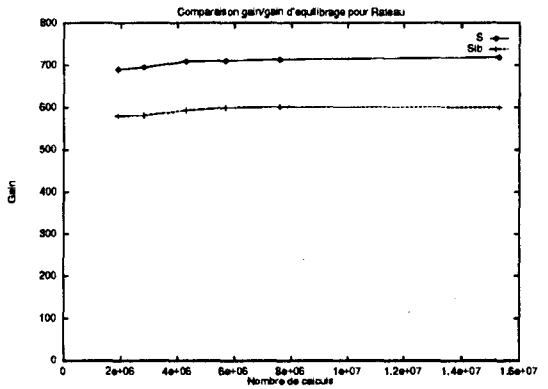


FIG. V.7 - *Comparaison du gain algorithmique et du gain d'exécution pour l'algorithme Râteau*

L'algorithme *Râteau* (*cf.* figure V.7) présente un écart important entre les deux courbes de gain, ce qui prouve le coût important de cette technique d'équilibrage. Par contre, on remarque que les deux courbes tendent à être parallèles à l'axe des abscisses. L'overhead induit par la redistribution des données peut donc être considéré comme relativement constant pour un nombre de processeurs donné lorsque la charge augmente.

Algorithmes Central/Rendez-vous

L'analyse des courbes des algorithmes *Central* et *Rendez-vous* présentées en figure V.8 et V.9 permet de comparer les différentes politiques d'appariement de ces deux techniques. La technique *Central* réalise un appariement en associant PEs oisifs et PEs chargés suivant leur index. L'algorithme *Rendez-vous* associe les PEs les plus chargés avec les processeurs inactifs. Cette différence s'observe sur la figure V.9 par la disparition des processeurs les plus chargés situés « au centre ». La qualité de ces algorithmes est toujours supérieure à 2, ce qui indique que les communications globales dans notre cas sont peu pénalisantes.

La comparaison des courbes produites par les algorithmes *Central* et *Rendez-vous* (*cf.* figures V.10 et V.11) montre le faible coût de ces deux techniques, S_I et S_{lb} étant pratiquement confondues. Par contre, le processus plus puissant d'appariement de la technique *Rendez-vous* n'apparaît pas clairement. Cette non accélération de la technique Rendez-vous peut s'expliquer par le coût de la recherche de partenaire.

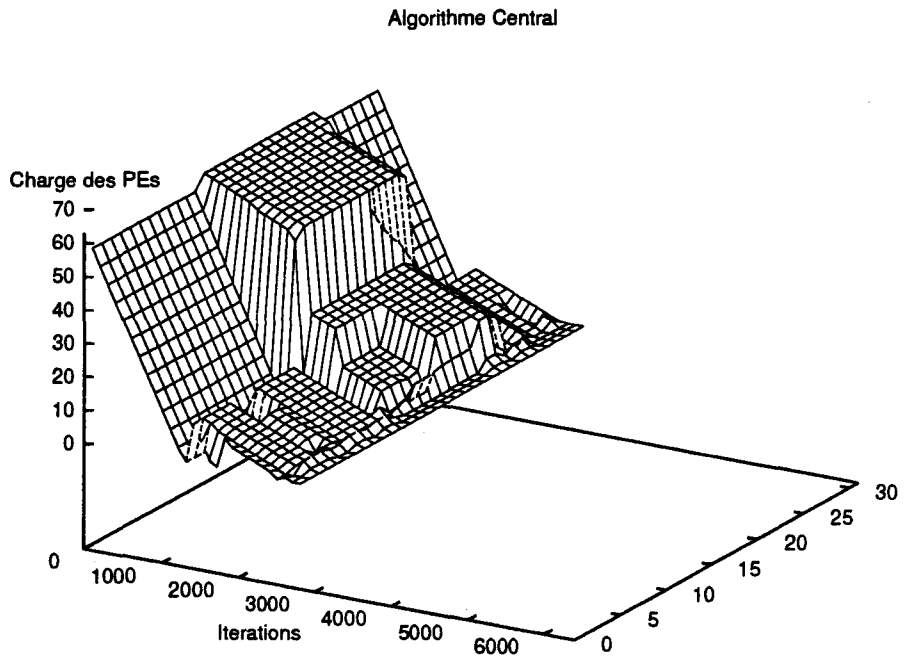


FIG. V.8 - *Rééquilibrage à l'aide de l'algorithme Central*
$I_n = 2880 \quad m = 8$
$S_I = 665 \quad S_{\text{lb}} = 660$
$Q = 2.22 \quad Q_{\text{lb}} = 2.1$

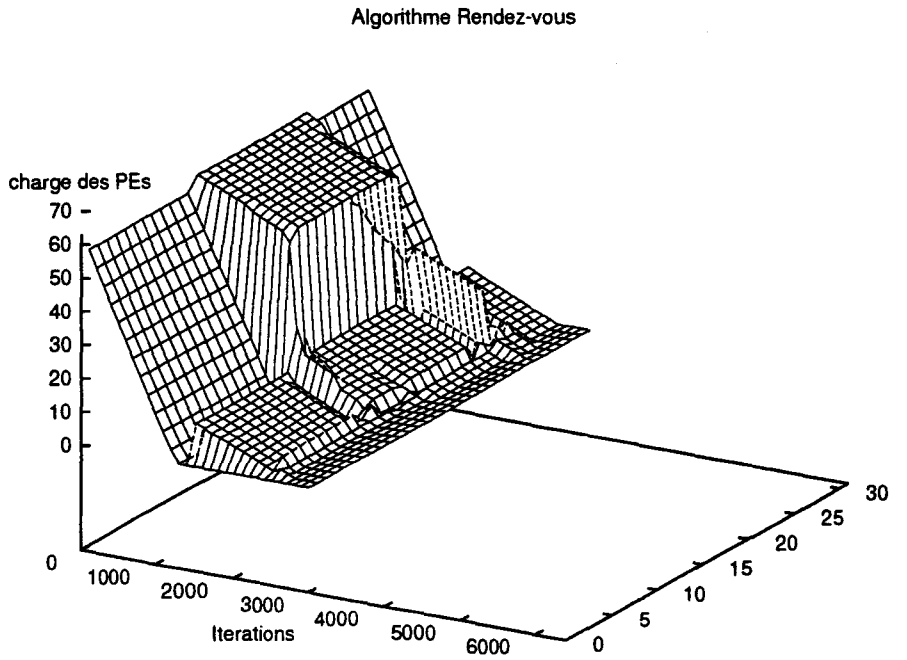


FIG. V.9 - *Rééquilibrage à l'aide de l'algorithme Rendez-vous*

$$I_n = 2940 \quad m = 7$$
$$S_I = 652 \quad S_{\text{lb}} = 647$$
$$Q = 2.17 \quad Q_{\text{lb}} = 2.1$$

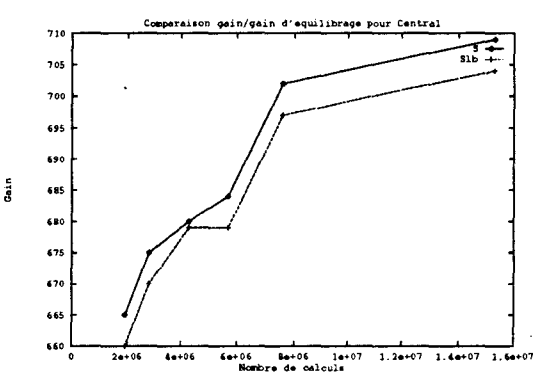


FIG. V.10 - *Comparaison du gain algorithmique et du gain d'exécution pour l'algorithme Central*

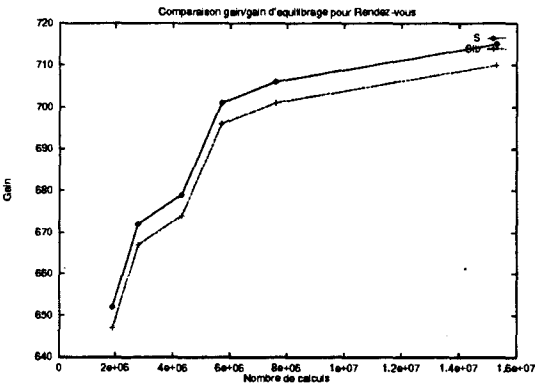


FIG. V.11 - *Comparaison du gain algorithmique et du gain d'exécution pour l'algorithme Rendez-vous*

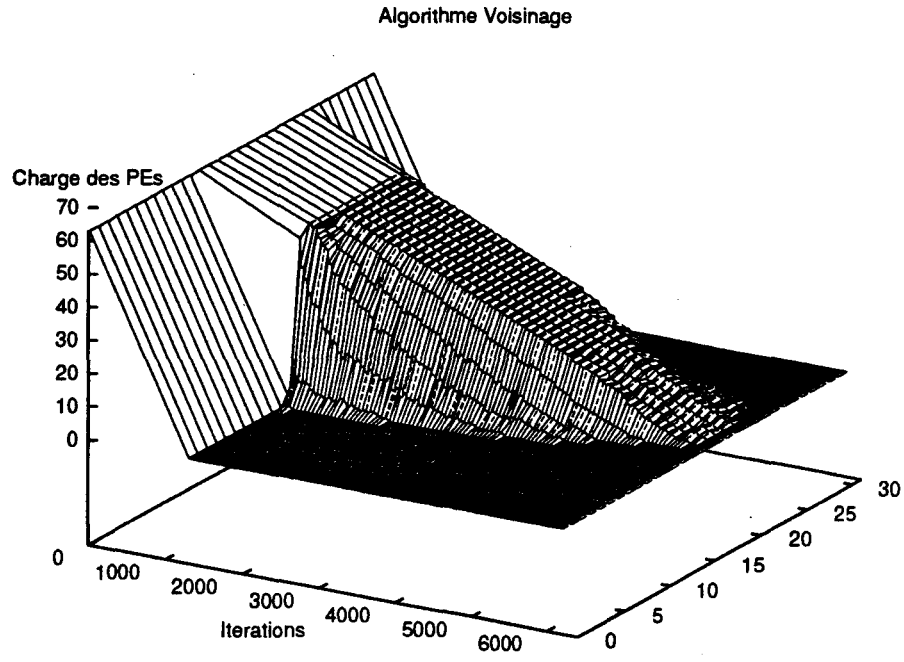


FIG. V.12 - *Rééquilibrage à l'aide de l'algorithme Voisinage*
$I_n = 6400$ $m = 99$
$S_I = 300$ $S_{\text{lb}} = 293$
$Q = 1.00$ $Q_{\text{lb}} = 0.96$

Algorithmes Voisinage/X-Voisinage

Sur un problème irrégulièrement distribué comme celui utilisé dans cette simulation, les différences entre les techniques n'utilisant que des communications de voisinage et celles utilisant des communications uniformes sont flagrantes. Sur la figure V.12, l'échange de travail n'est réalisé que sur les frontières entre les zones surchargées et sous-chargées. La diffusion des données n'est réalisée que de manière fort lente à travers le système. Par contre, l'algorithme *X-Voisinage* permet une accélération de la diffusion des données sur la grille. Dès les premiers équilibrages, les échanges de charge sont réalisés dans tout le système. La figure V.13 montre les envois de données distants et l'équilibrage à nouveau réalisé sur des îlots de taille 1 à cette distance. La technique *Voisinage* possède une qualité inférieure à 1, ce qui indique que son utilisation ralentit le système.

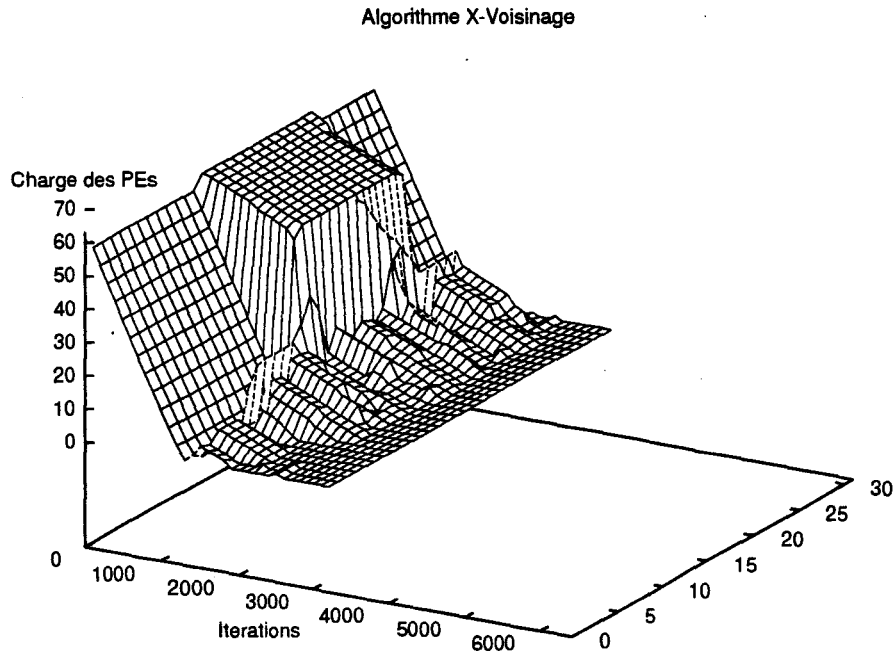


FIG. V.13 - *Rééquilibrage à l'aide de l'algorithme X-Voisinage*

$$I_n = 3200 \quad m = 19$$
$$S_I = 600 \quad S_{\text{lb}} = 590$$
$$Q = 2.00 \quad Q_{\text{lb}} = 1.9$$

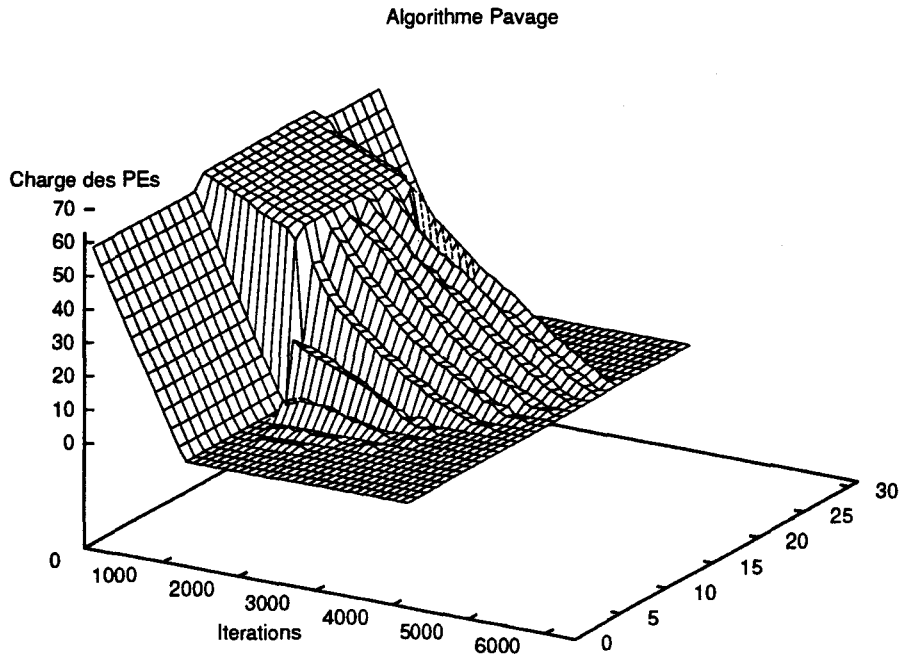


FIG. V.14 - *Rééquilibrage à l'aide de l'algorithme Pavage*
$I_n = 4240$ $m = 55$
$S_I = 452$ $S_{\text{lb}} = 418$
$Q = 1.51$ $Q_{\text{lb}} = 1.36$

Algorithmes Pavage et X-Pavage

La figure V.14 présente l'équilibrage réalisé par la technique *Pavage*. On peut apercevoir les fenêtres de redistribution sur lesquelles le travail est égalisé. Comme la technique *Voisinage*, la diffusion du travail vers les zones à faible charge est lente. La technique *X-Pavage* permet un équilibrage de qualité de manière rapide.

L'utilisation de la méthode *X-Pavage* permet d'améliorer de façon importante le gain algorithmique et le gain d'exécution par rapport à l'approche *purement* locale offerte par l'algorithme *Pavage* (*cf.* figure V.16 et V.17). L'écart entre les courbes de gain d'exécution et de gain algorithmique est plus importante pour la technique *Pavage* que *X-Pavage*. Cet écart important s'explique dans le cas de l'algorithme *Pavage* par le nombre important de communications réalisées.

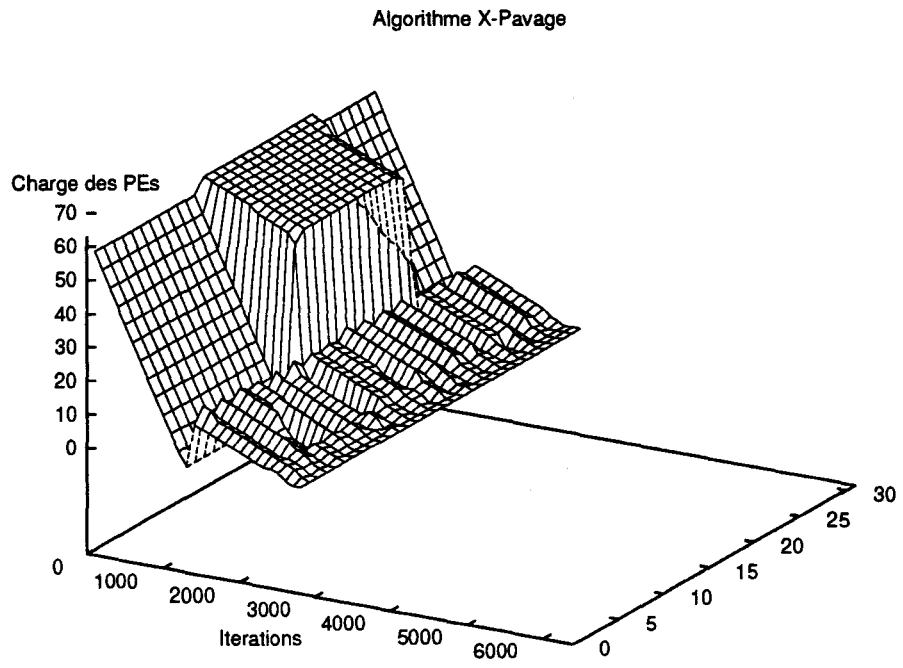


FIG. V.15 - *Rééquilibrage à l'aide de l'algorithme X-Pavage*

$$I_n = 2840 \quad m = 5$$
$$S_I = 675 \quad S_{\text{lb}} = 653$$
$$Q = 2.25 \quad Q_{\text{lb}} = 2.15$$

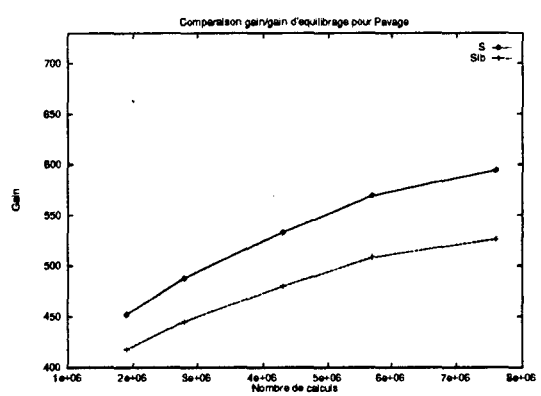


FIG. V.16 - *Comparaison du gain algorithmique et du gain d'exécution pour l'algorithme Pavage*

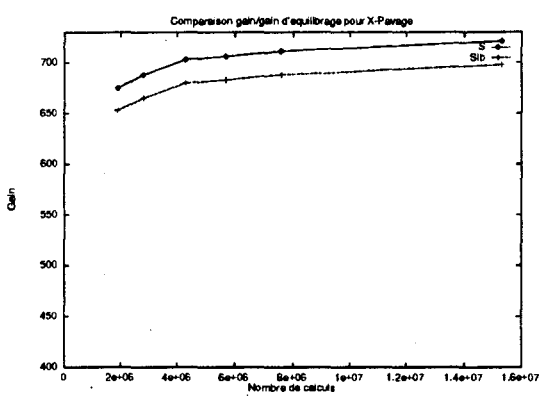


FIG. V.17 - *Comparaison du gain algorithmique et du gain d'exécution pour l'algorithme X-Pavage*

En conclusion :

- l'algorithme *Râteau* présente un des meilleurs gains algorithmiques. Par contre, son coût élevé fait chuter de manière importante le gain d'exécution ;
- les techniques *Central* et *Rendez-vous* présentent des gain algorithmique/gain d'exécution importants. Les communications globales ne semblent pas diminuer les performances de ces algorithmes. Le gain élevé peut s'expliquer par le faible nombre de communications globales réalisées à chaque appel ;
- les algorithmes basés sur l'utilisation des communications uniformes dépassent nettement les performances des algorithmes *purement* locaux.

5.2 Fréquence de l'équilibrage dynamique

La fréquence est un paramètre primordial de notre étude. Un équilibrage à coût important peut être accepté s'il entraîne une fréquence faible. Dans les figures, plus la fréquence tend vers de fortes valeur, plus l'algorithme de redistribution des données est appelée. Une fréquence faible indique par contre une meilleure répartition des points sur la grille de processeurs et un appel aux routines de rééquilibrage plus rare. Ces différentes résultats doivent être comparés au gain d'exécution.

Fréquence de l'algorithme Râteau

La fréquence de l'algorithme *Râteau* est extrêmement faible (*cf.* figure V.18) et diminue même avec l'accroissement de la charge des processeurs. Cette faible fréquence s'explique par l'équilibrage quasi parfait obtenu après chaque phase de réarrangement dynamique. La contrepartie de cette faible fréquence est le coût non négligeable du réarrangement dynamique.

Fréquence des algorithmes Central et Rendez-vous

La comparaison (*cf.* figure V.19) de la fréquence d'équilibrage pour les techniques *Central* et *Rendez-vous* montre un net avantage comme nous l'avions prévu à ce dernier algorithme. La technique d'appariement plus coûteuse permet une diminution de la fréquence. La fréquence de la technique *Central* augmente avec le nombre de calculs tandis qu'elle diminue et reste plus faible pour l'algorithme *Rendez-vous*. Les deux techniques présentent cependant une fréquence légèrement plus forte que l'algorithme *Râteau* qui du fait même de sa conception provoque un équilibrage parfait et nécessite un nombre minimal d'équilibrages.

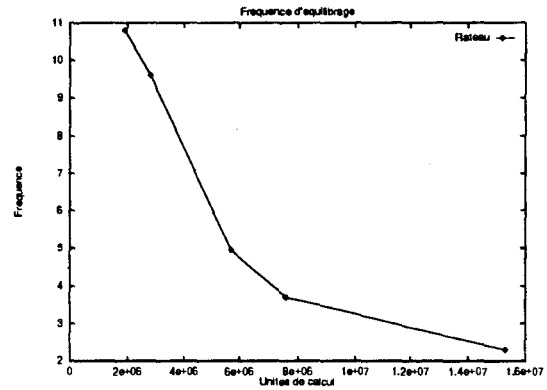


FIG. V.18 - *Fréquence de l'algorithme Râteau*

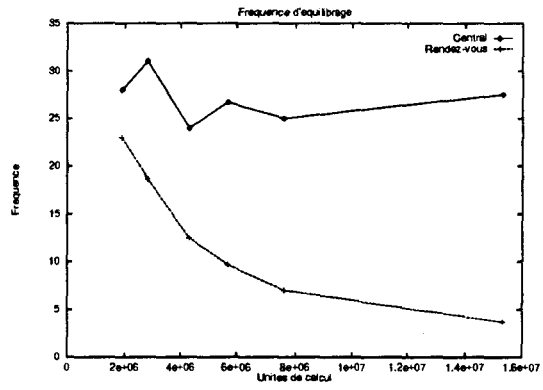


FIG. V.19 - *Comparaison des fréquences d'équilibrage des algorithmes Central et Rendez-vous*

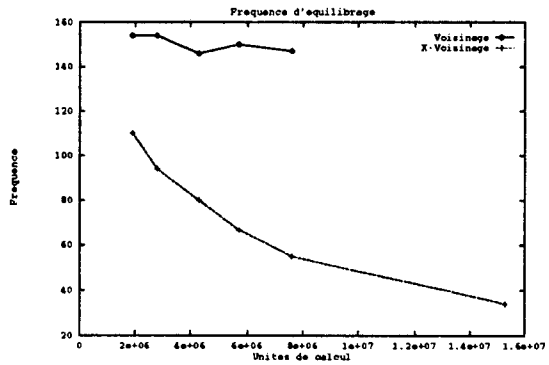


FIG. V.20 - *Comparaison des fréquences d'équilibrage des algorithmes Voisinage et X-Voisinage*

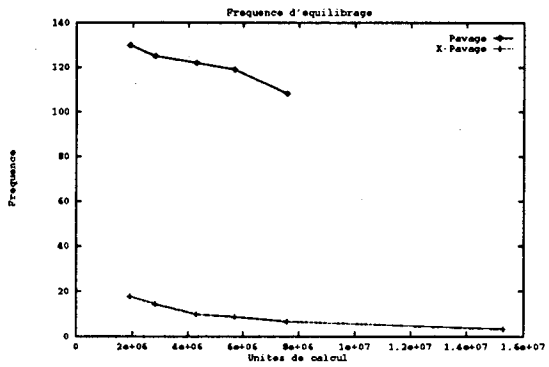


FIG. V.21 - *Comparaison des fréquences d'équilibrage des algorithmes Pavage et X-Pavage*

Fréquence des algorithmes Pavage et X-Pavage

La technique *X-Pavage* montre sa supériorité manifeste par rapport à l'algorithme *Pavage* qui se limite à des communications locales (*cf.* figure V.21). La même constatation peut aussi être effectuée après examen des résultats des techniques *Voisinage* et *X-Voisinage* (*cf.* figure V.20). Par contre, la fréquence extrêmement faible de l'algorithme *X-Pavage* corrobore les résultats théoriques présentés dans le chapitre IV.

Les résultats expérimentaux sur la fréquence d'équilibrage permettent de tirer les conclusions suivantes :

- les meilleurs résultats sont obtenus par l'algorithme *Râteau*, ce qui est logique puisqu'il entraîne une uniformisation de la charge sur l'ensemble de la grille à chaque exécution;
- un problème uniformément irrégulier provoque des appels fréquents aux routines de redistribution pour les techniques *purement* locales ;
- les bons comportements des techniques de réarrangement dynamique *X-Pavage* et *Rendez-vous* qui s'approchent asymptotiquement de la fréquence de l'algorithme *Râteau* ;
- la baisse de fréquence entre les techniques *Central* et *Rendez-vous*. Un appariement de qualité entraîne une diminution du nombre de redistribution ;
- la fréquence de l'algorithme *Rendez-vous* converge vers celle de l'algorithme *Râteau*.

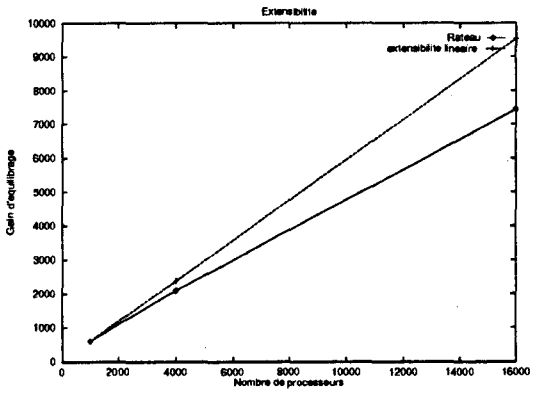


FIG. V.22 - *Extensibilité de l'algorithme Râteau*

5.3 Extensibilité des algorithmes

L'extensibilité permet d'apprécier le comportement des algorithmes lorsque le nombre de processeurs élémentaires augmente. Un algorithme est dit *extensible* si l'efficacité reste constante lorsque le nombre de processeurs augmente en proportion avec le travail.

Dans le cas idéal, une extensibilité linéaire est obtenue : lorsque le nombre de processeurs augmente linéairement, l'efficacité est constante si le travail croît dans la même proportion. Par contre, s'il faut augmenter le nombre de processeurs de façon exponentielle lorsque le travail croît linéairement, l'algorithme est considéré comme faiblement extensible.

Dans de nombreux problèmes comme le parcours d'arbre en parallèle, l'extensibilité est difficile à mesurer car surviennent des problèmes de super-linéarité. On obtient alors des gains supérieurs au nombre de processeurs du système. La résolution de l'ensemble de Mandelbrot permet d'éviter le problème de la super-linéarité. Pour mesurer l'extensibilité de nos différents algorithmes d'équilibrage, nous avons étudié l'augmentation du gain d'exécution S_{lb} en fonction du nombre de processeurs. La MasPar MP-1 permet d'être configurée en grilles de 32×32, 64×64 ou 128×128 processeurs élémentaires, soit 1024, 4096 ou 16384 processeurs.

Extensibilité de l'algorithme Râteau

L'extensibilité de l'algorithme *Râteau* est relativement faible (*cf.* figure V.22). Ce résultat est assez prévisible. Le coût d'exécution de l'algorithme *Râteau* est dépendant de l'architecture. Dans le cas d'une grille 2D comme la MasPar MP-1, il faut deux appels à la routine pour effectuer une redistribution des données. Sur une grille de taille plus importante, la durée d'exécution augmentera en proportion. Pour cette raison, l'extensibilité de la technique *Râteau* ne peut être très élevée.

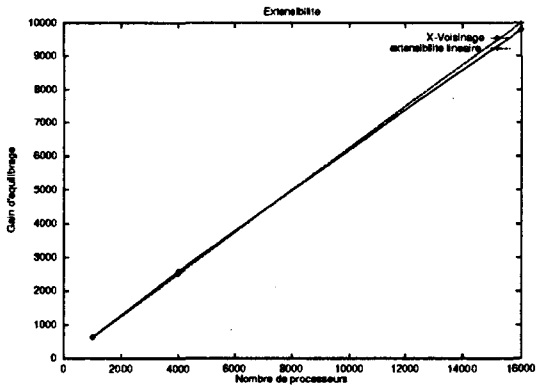


FIG. V.23 - *Extensibilité de l'algorithme X-Voisinage*

Extensibilité des algorithmes Voisinage et X-Voisinage

L'algorithme *Voisinage* sur l'exemple que nous étudions a un comportement globalement mauvais. Le nombre d'itérations nécessaire pour la terminaison de l'ensemble de Mandelbrot est identique avec ou sans le rééquilibrage *Voisinage*. Ce problème est dû au cas particulier que nous avons choisi d'étudier où les zones surchargées et sous-chargées sont disjointes. Il est par conséquent inintéressant d'examiner l'extensibilité de cette technique de redistribution des données.

L'algorithme *X-Voisinage* présente une très bonne extensibilité comme indiqué sur la figure V.23. La fonction chemin que nous avons définie intuitivement répond bien à de tels problèmes déséquilibrés.

Extensibilité des algorithmes Pavage et X-Pavage

L'extensibilité des techniques de rééquilibrage *Pavage* et *X-Pavage* confirment les résultats que nous avions pressentis dans la chapitre IV. Pour les grilles de taille importante (128×128), l'extensibilité de *Pavage* devient beaucoup plus faible que l'extensibilité de *X-Pavage*. Le coût de l'algorithme *X-Pavage* reste toujours faible par l'utilisation de communications uniformes (*cf.* figures V.24 et V.25).

6 Conclusion

Les résultats présentés dans cette section n'ont pas pour but de quantifier toutes les algorithmes d'équilibrage dans tous les cas de figures. De nombreuses autres courbes auraient également pu être présentées sur des problèmes dont le déséquilibre est plus ou moins im-

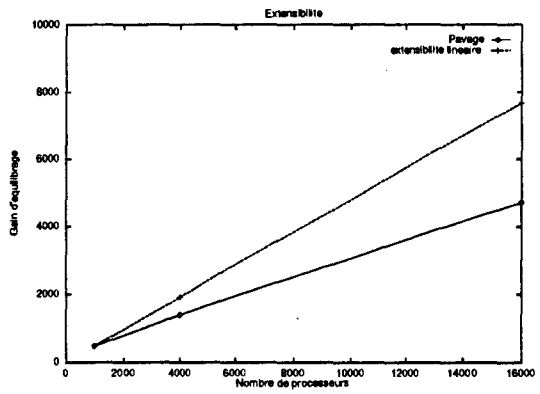


FIG. V.24 - *Extensibilité de l'algorithme Pavage*

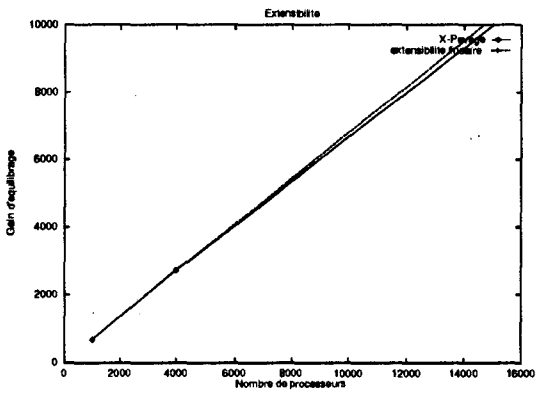


FIG. V.25 - *Extensibilité de l'algorithme X-Pavage*

portant. D'autres paramètres comme l'utilisation ou l'efficacité auraient pu être introduits. Nous souhaitons montrer que l'équilibrage dynamique sur une machine SIMD provoque un accroissement des performances y compris sur un problème dont le coût calculatoire reste modique. Dans le cas présenté dans les exemple précédents, la qualité des algorithmes est presque toujours supérieure à 2.

Conclusion

Les algorithmes à piles sont caractérisés par l'application d'une même séquence d'instructions sur des données différentes représentées par une pile. Ces entités sont indépendantes les unes des autres. La parallélisation de ce type d'algorithmes est possible par la projection de la pile sous forme de piles distribuées sur l'ensemble des processeurs élémentaires. La parallélisation entraîne généralement un déséquilibre, certains PEs « épuisant » plus rapidement leur pile que d'autres.

Nous avons présenté dans un premier temps l'existant des méthodes d'équilibrage MIMD et les éventuels apports sur le modèle SIMD. Nous avons défini cinq critères permettant de caractériser les principales techniques d'équilibrage MIMD : les politiques d'informations, de déclenchement, de sélection, de désignation locale et d'appariement. Cette étude de l'existant nous a permis de proposer un modèle d'équilibrage pour les machines synchrones dont les politiques sont : déclenchement, sélection, appariement, communications. Nous avons proposé différents algorithmes obéissant à ce modèle en utilisant un pseudo-code data-parallèle. Les algorithmes proposés sont classés en deux catégories : les techniques utilisant des communications globales et celles utilisant des communications de voisinage ou uniformes.

À partir de l'étude des méthodes d'analyse mathématique des algorithmes d'équilibrage dynamiques, les termes de convergence et de vitesse de converge des techniques d'équilibrage ont été présentés. La méthode itérative matricielle a permis de calculer la vitesse de convergence de nos algorithmes *Pavage* et *X-Pavage*. Par des considérations sur l'évolution de la charge des processeurs, la convergence de nos algorithmes *Râteau* et *Rendez-vous* a été prouvée.

Nous avons montré sur un exemple simple (calcul de l'ensemble de Mandelbrot) que l'équilibrage sur des architectures synchrones amène une amélioration certaine des performances et que le sur-coût provoqué par le l'équilibrage est compensé par le gain.

Vers une stratégie de choix d'un algorithme de rééquilibrage

Les chapitres précédents ont introduit les diverses politiques de redistribution et ont présenté leurs performances au niveau empirique et théorique. Pour un programmeur qui souhaite utiliser les algorithmes de redistribution, il est nécessaire de choisir parmi nos diverses techniques celles les mieux adaptées, nous proposons donc ici différents critères.

Déséquilibre temporel Cette notion introduite dans le chapitre consacré à l'analyse de performances permet d'estimer le déséquilibre entre les processeurs. Plus le déséquilibre est important, plus la politique de redistribution devra se soucier de la qualité de l'appariement (les processeurs lourdement chargés s'associeront avec des PEs fortement déficitaires). Une technique comme *Rendez-vous* permettra de mieux répondre aux problèmes fortement déséquilibrés temporellement. Les algorithmes *Pavage* et *X-Pavage* permettent une bonne égalisation de la charge à travers le système. Cependant, une technique comme *X-Pavage* a une politique de sélection systématique. Elle aura le même comportement vis-à-vis d'un problème légèrement déséquilibré que d'un problème possédant un déséquilibre important. Un tel appariement est désavantageux lorsque la charge est déjà régulièrement répartie. L'algorithme *Rendez-vous* est le plus apte à répondre à un fort déséquilibre temporel du système.

Déséquilibre spatial Cette notion caractérise la dispersion spatiale de la charge dans le système. Typiquement, deux cas peuvent se présenter: soit l'irrégularité de la charge est répartie uniformément dans le système, soit au contraire des îlots possédant une charge sensiblement égale apparaissent. Si des techniques faisant progresser la charge par diffusion dans le système sont applicables dans le premier cas, elles sont inadaptées dans le second cas. Les techniques *X-* permettent de répondre aux problèmes ayant un fort déséquilibre spatial.

Réseau de communications du système Les machines devant simuler les communications globales sur un réseau de voisinage ne seront pas adaptées aux algorithmes d'équilibrage globaux. Le ratio communications globales/communications locales est un guide dans le choix d'une stratégie d'équilibrage ; les stratégies locales nécessitent généralement un nombre de communications supérieur aux politiques globales. Le ratio permet de déterminer si une technique d'équilibrage nécessitant un nombre important de communications de voisinage à faible coût est plus « adaptée » qu'un technique d'équilibrage nécessitant un plus faible nombre de communications globales à coût élevé.

Un critère important dans le choix d'un algorithme d'équilibrage est la politique de déclenchement. Des méthodes simples comme les techniques à base de seuil peuvent sous certaines conditions avoir un comportement défavorable. Un algorithme proposant une bonne qualité

de rééquilibrage peut être appelé avec un seuil plus faible. Les algorithmes utilisant des communications de voisinage nécessiteront un seuil plus bas afin que le coût du rééquilibrage ne dépasse pas celui du problème.

Vers des algorithmes d'équilibrages SPMD?

Le modèle de programmation data-parallèle est bien adapté aux algorithmes à piles. Les techniques d'équilibrage présentées dans cette thèse s'appliquent aux machines SIMD. Sans s'écarter du modèle de programmation data-parallèle, un modèle d'exécution SPMD sur machines homogènes ou hétérogènes est un axe de recherche envisageable. Une modification de la politique d'équilibrage dynamique apparaît nécessaire pour prendre en compte ce nouveau modèle d'exécution. Quelques idées peuvent être émises : adaptation de l'indicateur de charge à la puissance des machines, modification de la politique de déclenchement, prise en compte de l'asynchronisme dans les transferts...

Les stations de travail proposées actuellement par les constructeurs proposent des puissances de calcul très importantes. Cette puissance est en outre exploitée efficacement par l'adjonction de réseaux permettant des communications à grande vitesse. Le modèle de programmation généralement offert aux programmeurs est celui basé sur le passage de messages. Des outils comme PVM ou MPI permettent une gestion plus ou moins simple et transparente de ces machines. Les inconvénients de ce modèle de programmation sont bien connus : difficulté de conception des algorithmes parallèles, non déterminisme de l'exécution, débogage délicat... L'utilisation du paradigme data-parallèle sur les machines MIMD proposée par exemple par Hatcher et Quinn [HQ91] permet d'utiliser un modèle de programmation parallèle simple et bien connu.

La forme de data-parallélisme implantée sur les machines MIMD est appelée SPMD. Le même programme est exécuté par chacun des processeurs, mais chaque processeur n'exécute pas obligatoirement la même instruction que les autres. Cette forme de data-parallélisme est rentable pour des structures simples comme le `if then else`. Cette instruction provoque l'inactivité d'une partie des processeurs élémentaires dans le cas d'une exécution SIMD, par contre dans le modèle SPMD, et sous certaines conditions, suivant la valeur du prédicat la partie `then` ou `else` sera exécutée. Le modèle d'exécution SPMD permet le passage d'un parallélisme à grain fin à un parallélisme à grain moyen.

Une question de pose : quelle est l'influence du modèle SPMD sur les algorithmes présentés? Ou encore, les algorithmes à piles peuvent-ils bénéficier de l'équilibrage dynamique sur le modèle SPMD? Généralement dans le modèle d'exécution SPMD, une ou plusieurs topologies virtuelles représentatives d'un domaine (une grille par exemple) sont simulées. Par contre, dans le cas des algorithmes à piles, aucune virtualisation n'est nécessaire, un nombre

de piles égal au nombre de processeurs sera créé. Le code généré de calcul des données sur chaque pile sera dupliqué sur chacun des processeurs.

L'introduction de l'asynchronisme permet à chaque station de travailler sur sa pile de données à sa propre vitesse. L'hétérogénéité caractérise la non-uniformité des performances des machines en terme de :

- vitesse propre au processeur ;
- charge du système (nous travaillons maintenant en environnement multi-utilisateurs et multi-tâches).

Un indicateur de la vitesse synthétisant ces deux aspects peut être par exemple, le nombre d'itérations effectuées pendant une certaine période. L'hétérogénéité des machines devra se refléter sur les politiques d'équilibrage à plusieurs niveaux :

- au niveau de la redistribution des données, les échanges entre les différentes machines seront pondérés par leur vitesse respective. Un système fonctionnant deux fois plus rapidement qu'un autre acceptera à l'issue du transfert une charge deux fois plus importante. Comme dans le cadre SIMD, chaque machine possédera un index local estimant la charge instantanée. Cet index, qui dépend seulement pour les machines SIMD du nombre de données ou d'objets présents dans chaque pile, sera en SPMD une fonction à deux variables : la charge et la vitesse du processeur. Une mise à jour périodique des vitesses des différents systèmes est également envisageable pour prendre en compte l'évolution possible des capacités de calcul au cours de la résolution du système ;
- au niveau des politiques de déclenchement, le rééquilibrage doit être déclenché de manière opportune. Il est nécessaire de prendre en compte la puissance respective des différents systèmes. La machine dont la vitesse est la plus grande doit être favorisée et doit provoquer un équilibrage plus rapidement qu'une machine à faible vitesse. Le seuil doit être défini au prorata de la vitesse instantanée des machines.

Les échanges de données doivent être réalisés avec précaution de manière à éviter d'éventuels goulots d'étranglement. Deux approches peuvent être envisagées :

- un site peut être désigné comme « maître », comme dans la stratégie développée par Utard et Cuvillier [UC94]. À partir des informations transmises par les différents sites, il décide ou non de l'équilibrage et dans l'affirmative transmet la nouvelle distribution de la charge sur les machines. À la vue de la nouvelle distribution, chaque nœud effectue les envois de données correspondants. Des méthodes d'équilibrage comme *Central* et *Rendez-vous* peuvent être employées pour ces distributions ;

- en s'inspirant des méthodes de sélection aveugle comme *X-Pavage*, un appariement peut être réalisé en associant les machines par paires dans un ordre déterminé. Un exemple est présenté en figure V.26 pour un ensemble de 8 stations de travail. L'avantage de cette méthode est qu'elle ne nécessite pas de site centralisateur. En outre, les échanges de travail sont réalisés de façon asynchrone empêchant ainsi une saturation du réseau de communications.

L'exécution des algorithmes à piles suivant le modèle SPMD présente des perspectives intéressantes. La conservation du modèle de programmation data-parallèle garantit une exécution déterministe et une programmation simple. L'hétérogénéité des machines interviendra au niveau de l'indicateur de charge et de la vitesse instantanée. Elle se répercutera au niveau du modèle d'équilibrage.

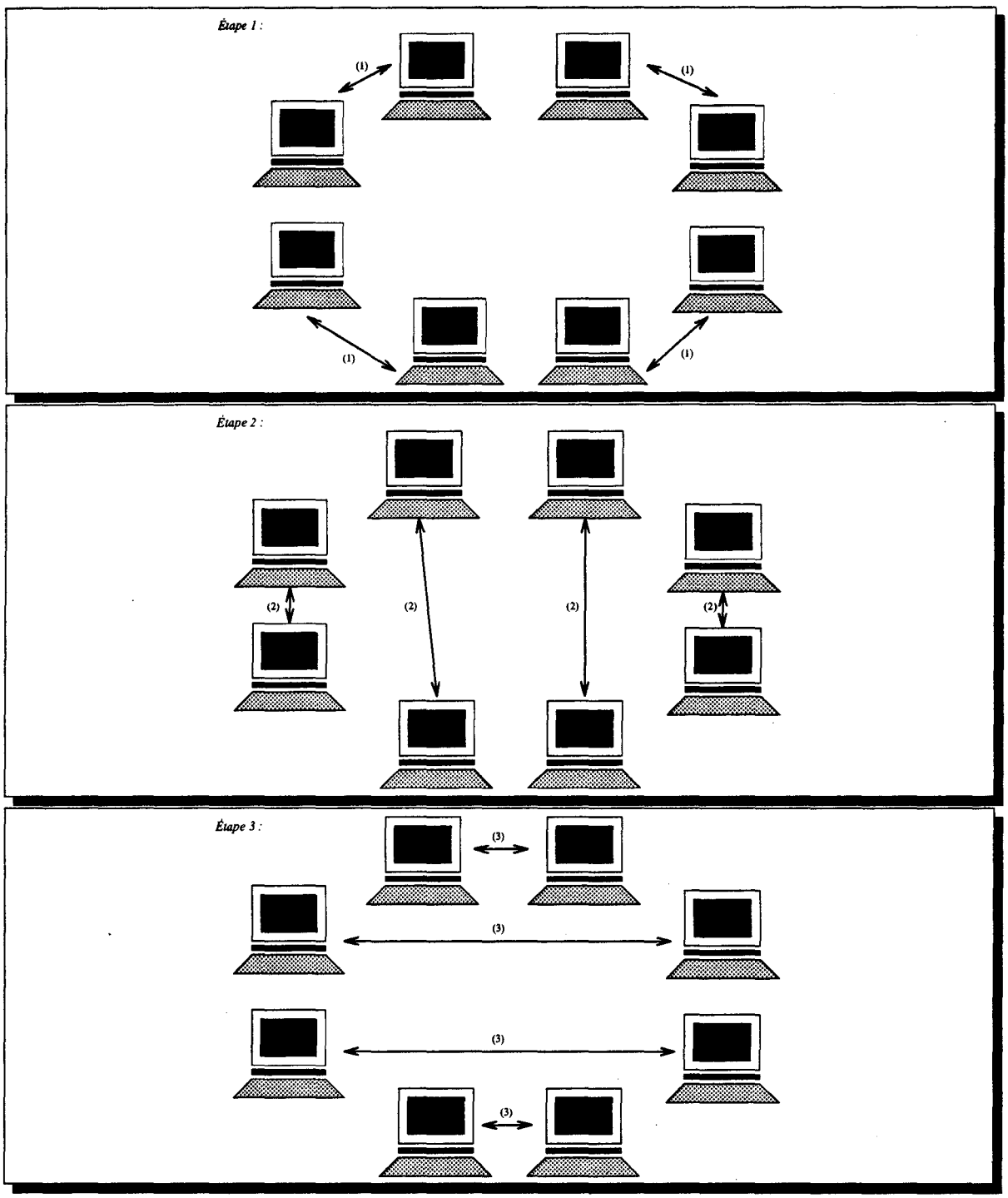


FIG. V.26 - *Exemple de redistribution de charge sur un ensemble de 8 stations de travail*

L'équilibrage avec contraintes

Une autre direction de recherches présentant des perspectives intéressantes est l'équilibrage avec contraintes. L'équilibrage avec contraintes consiste à rééquilibrer la charge des processeurs élémentaires tout en conservant certaines contraintes comme le voisinage ; les valeurs des points sont calculées de façon itérative par une convolution sur les voisins. Par exemple, dans l'amincissement d'images [Ols92], la disparition de pixels est calculée d'après les valeurs fournies par son entourage. Le déséquilibre est provoqué par les différents taux de convergence des points. Au bout de quelques itérations, un certain nombre de pixels sont devenus inactifs et n'ont plus besoin d'être calculés. Les données restantes qui nécessitent la poursuite des calculs ne sont pas en général distribuées de manière égale sur l'ensemble des PEs. L'équilibrage dynamique de ces points permet d'obtenir une meilleure répartition de la charge. Cependant, le traitement à venir sur un de ces points nécessite la lecture des éléments voisins. Le rééquilibrage ne peut donc se faire sans considérer la totalité du voisinage des points. On parle dans ce cas d'équilibrage dynamique avec contraintes.

L'équilibrage avec contraintes redistribue les données sur un ordinateur massivement parallèle en conservant le principe de la localité. Il est en général appliqué sur des topologies en forme de *grilles*. Le mot *grille* étant pris dans le sens large du terme, une grille à une dimension indiquant une chaîne de processeurs, une grille à deux dimensions un tableau de PEs... Chaque processeur de la grille est connecté à un ensemble de processeurs voisins. Les données utilisées dans les calculs sont structurées sous forme de grilles. Les opérations effectuées sur les points de la grille de données nécessitent des valeurs possédées par les points voisins. Lorsque la projection de la grille de données est effectuée sur la grille de processeurs, il est nécessaire de conserver les contraintes de voisinage. Comme le traitement de certains points sera terminé avant certains autres, il faudra redistribuer les points sur les processeurs en conservant le principe de la localité. Examinons par exemple, la figure V.27, il s'agit de projeter un plan de 16 données groupées suivant un schéma 4×4 sur une grille de 2×2 processeurs élémentaires, tout en conservant le principe de la localité et en équilibrant le mieux possible la charge des 4 processeurs (on suppose dans ce cas un poids identique pour chacun des points). La solution dans ce cas est triviale.

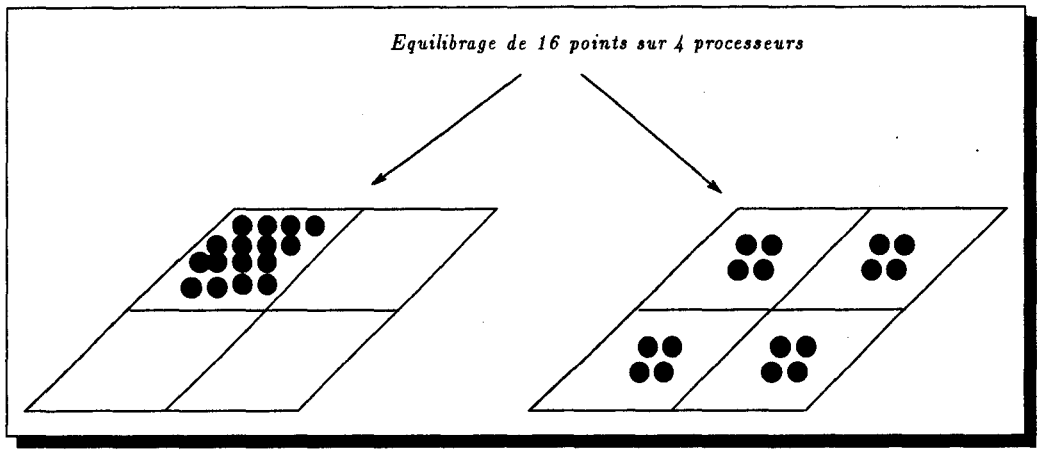


FIG. V.27 - *Projection de 16 points en respectant les contraintes de localité*

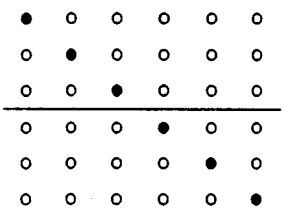
Sur une topologie mono-dimensionnelle, il existe des techniques relativement simples pour trouver une distribution optimale des données sur un ensemble de N processeurs. L'algorithme élastique [MR92] développe ce principe pour l'amincissement d'images. L'image est stockée sous forme de colonnes sur un anneau de 32 transputers. Les colonnes sont déplacées régulièrement de transputers en transputers afin de conserver une charge égale sur chaque machine. L'algorithme *Râteau* en une dimension permet également la conservation de la localité si la gestion de la charge locale est de type FIFO.

Cependant, dès que l'on aborde des problèmes de dimension supérieure, il devient difficile de trouver un algorithme performant pour trouver une redistribution optimale. Traiter, les dimensions les unes à la suite des autres ne mène pas à un partitionnement idéal. Soit l'exemple suivant :

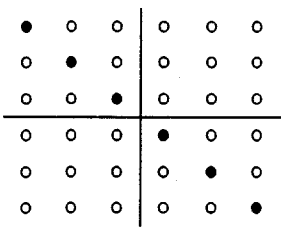
$$\begin{array}{cccccc}
\bullet & \circ & \circ & \circ & \circ & \circ \\
\circ & \bullet & \circ & \circ & \circ & \circ \\
\circ & \circ & \bullet & \circ & \circ & \circ \\
\circ & \circ & \circ & \bullet & \circ & \circ \\
\circ & \circ & \circ & \circ & \bullet & \circ \\
\circ & \circ & \circ & \circ & \circ & \bullet
\end{array}$$

pour lequel un $\bullet$ représente un charge de 1 et $\circ$ une charge nulle. Soit cette structure à projeter sur une grille 2×2 de processeurs. Considérons tout d'abord un découpage horizontal ; on

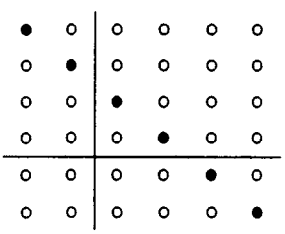
obtient



Si à la suite de ce découpage horizontal, on réalise un découpage vertical, on obtient



C'est-à-dire que deux processeurs sont chargés de 3 valeurs et deux autres processeurs n'ont aucune charge. Ce découpage n'est pas optimal ; le découpage suivant



donne une charge de 2 valeurs pour trois processeurs et un seul processeur non chargé.

Cette technique de partitionnement multi-dimensionel a été étudiée par Biagioni [Bia91] dans sa thèse sous le nom d'OMS qui désigne un découpage Orthogonal, Monotone et Surjectif. Cette approche permet d'obtenir un partitionnement en travaillant sur une dimension puis l'autre, mais qui ne garantit pas l'optimalité de la redistribution. Une variante de OMS appelé *partitionnement rectilinéaire* a été étudié de manière théorique par Nicols [Nic91]. Une solution *localement* optimale est proposée en 2D. Aucune solution globale n'a pour le moment été proposée. Le cas 3D est connu comme étant NP-complet. Un problème similaire est le découpage optimal d'un graphe pondéré projeté sur une grille à deux dimensions, en minimisant le déséquilibrage. Une méthode pour le partitionnement d'une telle grille est appelée *bisection spectrale récursive* [PSL90, DR94]. Cependant, l'utilisation d'une telle méthode qui nécessite le calcul de valeurs propres n'est pas utilisable sur une machine pour laquelle le sur-coût induit par l'équilibrage doit être le plus faible possible.

L'équilibrage avec contraintes présente des perspectives de recherches intéressantes, en particulier le problème du partitionnement équilibré d'une grille à plusieurs dimensions (traitement d'images sur deux dimensions, problèmes de mécanique des fluides et d'éléments finis en trois dimensions...).

Bibliographie

[AG90] Ishfaq Ahmad and Arif Ghafoor. A demi distributed task allocation strategy for large hypercube supercomputers. In *Supercomputing 90*, pages 898–905, 1990.

[AGM93] Ishfaq Ahmad, Arif Ghafoor, and Kishan Mehrotra. Performance prediction of distributed load balancing on multicomputer systems. Technical report, Syracuse University, Syracuse, USA, 1993.

[All80] Arnold O. Allen. Queuing models of computer systems. *IEEE Computer*, 13(4):13–24, April 1980.

[BF92] Raouf Boutaba and Bertil Folliot. Presentation of a multicriteria load balancing. Technical Report MASI 92.42, Institut Blaise Pascal, Paris, France, June 1992.

[BGT94] P. Bouvry, J-M Geib, and D. Trystram. *Conception et analyse de programmes parallèles*, chapter Placement statique et dynamique. Hermès, April 1994.

[Bia91] Edoardo Stanley Biagioni. *Scan Directed Load Balancing.* PhD thesis, University of North Carolina, Chapel Hill, NC, USA, 1991.

[BKW88] Katherine M. Baumgartnet, Ralph Kling, and Benjamin Wah. A global load balancing strategy for a distributed system. In *IEEE Conference on Distributed Computing Systems*, pages 93–102, Hong-Kong, 1988.

[Bla90] Tom Blank. The MasPar MP-1 architecture. In *Proceedings of the IEEE Compcon Spring 1990*, pages 20–24, San Francisco, CA, February 1990.

[BP79] A. Berman and R.J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences.* Academic Press, New York, 1979.

[BS85] Amnon Barak and Amnon Shiloh. A distributed load-balancing policy for a multicomputer. *Software — Practice and Experience*, 15(9):901, September 1985.

[BSS91] Guy Bernard, Dominique Stève, and Michel Simatic. Placement et migration de processus dans les systèmes répartis faiblement couplés. *Technique et Science Informatiques*, 10(5):375–392, 1991.

[BT89] D.P. Bertsekas and John N. Tsitsiklis. *Parallel & Distributed Computation*, chapter 7.4, pages 519–526. Prentice Hall International Editions, 1989.

[Cab86] L.F. Cabrera. The influence of workload on load balancing strategies. In *USENIX Summer'86*, Atlanta, USA, June 1986.

[Cho90] Shyamal Chowdhury. The greedy load sharing algorithm. *Journal of Parallel and Distributed Computing*, 9:93–99, 1990.

[CK82] Yaun-Chien Chow and Walter H. Kohler. Models for dynamic load balancing in homogenous multiple processor systems. *IEEE Transactions on Computers*, C-36(6):667–679, May 1982.

[CK87] Thomas L. Casavant and Jon G. Kuhl. Analysis of three dynamic distributed load-balancing strategies with varying global information requirements. In *7th International Conference on Distributed Computing Systems*, pages 185–192, Berlin, Germany, September 1987.

[CK88] Thomas L. Casavant and Jon G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engeneering*, 14(2):141–154, February 1988.

[CLZ92] A. Corradi, L. Leonardi, and F. Zambonelli. Load balancing strategies for massively parallel architectures. *Parallel Processing Letters*, 2(2 & 3):139–148, 1992.

[Cyb87] George Cybenko. Dynamic load balancing for distributed memory multriprocessors. Technical Report TR 87-1, University of Medford, MA, USA, 1987.

[Cyb89] George Cybenko. Dynamic load balancing for distributed memory multiprocessors. *Journal of Parallel and Distributed Computing*, 7:279–301, 1989.

[DD92] Olivier Dukers and Salah Dowaji. Comparaison de trois stratégies d'équilibrage de charge sur un réseau simulé de processeurs homogènes. Technical Report MASI 92.79, Institut Blaise Pascal, Paris, France, November 1992.

[Dei84] H.M. Deitel. *An Introduction to Operating Systems*, chapter Queuing Theory, pages 380–412. Addison Wesley, Massachusetts, 1984.

[Dek86] Jean-Luc Dekeyser. *Architectures et algorithmes parallèles pour les méthodes Montè Carlo en physique des particules.* PhD thesis, Université de Lille, Lille, France, 1986.

[Dew85] Alan Dewdney. Computer recreations: A computer microscope zooms in for a look at the most complex object in mathematics. *Scientific American*, page 16, August 1985. Traduit en Français dans Pour La Science.

[DFM93] Jean-Luc Dekeyser, Cyril Fonlupt, and Philippe Marquet. Distributions dynamiques de données pour machines massivement parallèles. In *Renpar5, 5es Rencontres sur le Parallélisme*, pages 183–186, Brest, May 1993.

[DFM94a] Jean-Luc Dekeyser, Cyril Fonlupt, and Philippe Marquet. A data-parallel view of the load balancing, experimental results on MasPar MP-1. In *High Performance Computing and Networking Conference*, volume 797 of *Lectures Notes in Computer Science*, pages 338–343, Munich, Germany, April 1994. Springer-Verlag.

[DFM94b] Jean-Luc Dekeyser, Cyril Fonlupt, and Philippe Marquet. Dynamic load-balancing on SIMD data-parallel supercomputers. In *EUROSIM'94*, Amsterdam, NL, 1994.

[DFM94c] Jean-Luc Dekeyser, Cyril Fonlupt, and Philippe Marquet. Rééquilibrage dynamique à parallélisme de données. In Luc Bougé, editor, *6èmes Rencontres Francophones du parallélisme*, pages 153–156, Lyon, France, June 1994.

[Dig92a] Digital Equipment Corporation, Massachusetts, USA. *DECmpp Parallel FORTRAN Reference Manual*, January 1992.

[Dig92b] Digital Equipment Corporation, Massachusetts, USA. *DECmpp Programming Language Reference Manual*, January 1992.

[DO91] Fred Douglis and John Ousterhout. Transparent process migration: Design alternatives and the sprite implementation. *Software — Practice and Experience*, 21(8):757–785, August 1991.

[DR94] Rafael Van Driessche and Dirk Roose. Dynamic load balancing with an improved spectral bisection algorithm. In *Scalable High-Performance Computing Conference*, Knoxville, Tenessee, USA, May 1994.

[DTJ89] Piyush Dikshit, Satish K. Tripathi, and Pankaj Jalote. SAHAYOG: a test bed for evaluating dynamic load-sharing policies. *Software — Practice and Experience*, 19(5):411–435, May 1989.

[ELZ86a] Derek L. Eager, Edward D. Lazowska, and John Zahorjan. Adaptive load sharing in homogeneous distributed systems. *IEEE Transactions on Software Engeneering*, SE-12(5):662–675, May 1986.

[ELZ86b] Derek L. Eager, Edward D. Lazowska, and John Zahorjan. A comparison of receiver-initiated and sender-initiated adaptive load sharing. *Performance Evaluation*, 6(1):53–68, 1986.

[FDM93] Cyril Fonlupt, Jean-Luc Dekeyser, and Philippe Marquet. Redistribution dynamique des données sur machine massivement parallèle. Technical report, Laboratoire d'Informatique Fondamentale de Lille, Université de Lille I, France, June 1993.

[Fel79] S.I. Feldman. Make – a program for maintening computer programs. *Software — Practice and Experience*, 9(4):255–265, 1979.

[FM87] Raphael Finkel and Udi Manber. DIB-a distributed implementation of backtracking. *ACM Transactions on Programming Languages and Systems*, 9(2), April 1987.

[FM90] R. Frye and J. Myczkowski. Exhaustive tree search of unstructured trees on the connection machine. Technical Report TMC-196, Thinking Machines Corporation, Cambride, MA, USA, 1990.

[FZ86] Domenico Ferrari and Songnian Zhou. A load index for dynamic load balancing. In *Fall Joint Computer Conference*, pages 684–690, Dallas, Texas, USA, November 1986. IEEE.

[FZ87] Domenico Ferrari and Songnian Zhou. An empirical investigation of load indices for load balancing applications. In P.-J. Courtois and G. Latouche, editors, *Performance'87*, pages 515–529, Bruxelles, Belgium, 1987. IFIP WG7.3, Elsevier Science Publishers.

[Gas93] Michel Gastaldo. Dictionary machine on SIMD architectures. Technical report, LIP École Normale Supérieure de Lyon, Lyon, France, 1993.

[Gav92] Cyril Gavoille. Evaluation of the MasPar performances. Technical report, Ecole Normale Supérieure de Lyon, Lyon, France, February 1992.

[GHPS94] H. Guyennet, B. Herrmann, L. Philippe, and F. Spies. A performance study of dynamic load balancing algorithms for multicomputers. In *MPCS*, pages 252–254, Ischia, Italie, 1994.

[GMS94] Jack J. Gongarra, Hans W. Meuer, and Erich Strohmaier. *TOP500 Supercomputers sites.* University of Mannheim. Disponible sur `ftp.uni-mannheim.de`, June 1994.

[GS93] Hervé Guyennet and François Spies. Étude comparative de différents algorithmes de répartition de charge dans les systèmes distribués. *La lettre du transputer*, pages 31–55, June 1993.

[Har86] R.S. Harbus. Dynamic process migration: To migrate or not to migrate. Technical Report CSRI-42, University of Toronto, Toronto, Canada, July 1986.

[HB84] Kai Hwang and Fayé A. Briggs. *Computer Architecture and Parallel Processing.* 1984.

[HCG+82] Kai Hwang, William J. Croft, George H. Goble, Benjamin W. Wah, Faye A. Briggs, William R. Simmons, and Clarence L. Coates. A unix-bases local computer network with load balancing. *IEEE Computer*, 15(4):55–65, April 1982.

[Hen94] Dominik Henrich. Local load balancing for data-parallel branch-and-bound. In Prof. L. Dekker, editor, *2nd International Conference on Massively Parallel Processing Applications and Development*, Delft, The Netherlands, June 1994. Eurosim.

[Hil88] Daniel Hillis. *La machine à connexions.* Masson, Paris, France, 1988. Traduction de "The Connexion Machine", MIT, 1985.

[HLM+90] Seyed H. Hosseini, Bruce Litow, Mohammad I. Malkawi, Joseph McPherson, and K. Vairavan. Analysis of a graph coloring based distributed load balancing algorithm. *Journal of Parallel and Distributed Computing*, 10:160–166, 1990.

[HLMV87] Seyed H. Hosseini, Bruce Litow, Mohammad I. Malkawi, and K. Vairavan. Distributed algorithms for load balancing in very large homogeneous systems. In *1987 Joint ACM-IEEE Comp. Conference*, pages 397–404, Dallas, USA, 1987. ACM-IEEE.

[HQ91] Philip J. Hatcher and Michael Quinn. *Data-Parallel Programming on MIMD Computers.* The MIT Press, Cambridge, MA, November 1991.

[HS86] W Hillis and Guy Steele. Data parallel algorithms. *Communications of the ACM*, 29(2):1170–1183, December 1986.

[Hwa93] Kai Hwang. *Advanced Computer Architecture. Parallelism, Scalability, Programmability.* Mc Graw Hill, 1993.

[JBSV92] Wouter Joosen, Yolande Berbers, Marc Snyers, and Pierre Verbaeten. Transparent object migration in adaptive parallel applications. In *Parallel Computing: From Theory to Sound Practice*, pages 300–311. IOS Press, 1992, 1992.

[JBV93] Wouter Joosen, Stijn Bijnens, and Pierre Verbaeten. A reusable load balancer for parallel search problems. In *Euromicro 93*, pages 205–212, Barcelona, September 1993.

[KG91] Vipin Kumar and Anshul Gupta. Analyzing the scalability of parallel algorithm and architectures: A survey. In *International Conference on Supercomputing*, June 1991.

[KGR91] Vipin Kumar, Ananth Y. Grama, and Vempaty Nageshwara Rao. Scalable load balancing techniques for parallel computers. Technical Report 91-55, University of Minnesota, Minneapolis, Minnesota, USA, 1991.

[KGV94] Vipin Kumar, Ananth Y. Grama, and Nageshwara Rao Vempaty. Scalable load balancing techniques for parallel computers. *Journal of Parallel and Distributed Computing*, 22:60–79, 1994.

[KK92] George Karypis and Vipin Kumar. Unstructured tree search on SIMD parallel computers: A summary of results. In *Supercomputing 92*, pages 453–462, 1992.

[Kle76] L. Kleinrock. *Queuing Systems*, volume 1. Wiley, New York, 1976.

[KR87] Vipin Kumar and V. Nageshwara Rao. Parallel depth first search. part II. *Int'l Journal of Parallel Programming*, 16(6):501–519, 1987.

[KS92] Phillip Kreuger and Mukesh Singhal. Load distributing in locally distributed systems. *IEEE Computer*, pages 33–44, December 1992.

[Kun91] Thomas Kunz. The influence of different workload descriptions on a heuristic load balancing scheme. *IEEE Transactions on Software Engeneering*, 17(7):725–730, July 1991.

[KVM68] P.J. Kiviat, R. Villanueva, and H.M. Markowitz. *The SIMSCRIPT II Programming Language*. Prentice Hall, 1968.

[KW91] H. Kuchen and A. Wagener. Comparison of dynamic load balancing strategies. *Journal of Parallel and Distributed Processing*, pages 303–314, 1991.

[LEZ88] E.D. Lazowska, D. Eager, and J. Zahorjan. The limited performance benefits of migrating active processes for load sharing. *ACM, Performance Evaluation Review*, 16:63–72, May 1988.

[LK87] F. Lin and R. Keller. The gradient model load balancing method. *IEEE Transactions on Software Engeneering*, 13(1):32, January 1987.

[LLM88] M.J. Litzkow, M. Livny, and M.W. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the Eighth International Conference on Distributed Computing Systems*, page 104, Los Alamitos, California, 1988. IEEE CS Press.

[LM82] Miron Livny and Myron Melman. Load balancing in homogenous broadcast distributed systems. *ACM Performance Evaluation*, 11(1):47–55, 1982.

[LM93] R. Lüling and B. Monien. Load balancing for distributed branch-&-bound. In *Third Mons Workshop on Parallel Computing*, September 1993.

[LMR91] R. Lüling, B. Monien, and F. Ramme. Load balancing in large networks: A comparative study. In *3rd IEEE Symposium on Parallel and Distributed Processing*, Dallas, 1991.

[LO86] Will E. Leland and Teunis J. Ott. Load-balancing heuristics and process behavior. In *ACM SIGMETRICS 1986*, pages 54–69, Raleigh, USA, May 1986. ACM.

[LT87] K.J. Lee and D. Towsley. Distributed optimization algorithms for quasi-static threshold load balancing. Technical Report 87-113, University of Massachusetts, October 1987.

[Man92] Bernard Mans. Un algorithme d'équilibrage de charge dynamique et auto-adaptatif pour le branch and bound. Technical Report 92.67, Laboratoire de méthodologie et architectures des systèmes informatiques, Institut Blaise Pascal, October 1992.

[Mar69] Harry M. Markowitz. Simulating with SIMSCRIPT. *Management Science - Serie B*, 12(10):396–405, June 1969.

[MD93] Ambuj Mahanti and Charles J. Daniels. A SIMD approach to parallel heuristic search. *Artificial Intelligence*, 60:243, 1993.

[MR92] Serge Miguet and Yves Robert. Elastic load-balancing for image processing algorithms. Technical report, LIP, Lyon, France, 1992.

[MT91] Traian Muntean and El-Ghazali Talbi. Méthodes de placement statique de processus sur des architectures parallèles. *Technique et Science Informatiques*, 10(5):355–373, 1991.

[Nic91] David M. Nicol. Rectilinear partitionning of irregular data parallel computations. Technical Report 91-55, ICASE, National Aeronautics and Space Administration, Hampton, Virginia, July 1991.

[NXG85] Lionel M. Ni, Chong-Wei Xu, and Thomas B. Gendreau. A distributed drafting algorithm for load balancing. *IEEE Transactions on Software Engeneering*, 11(10):1153–1161, October 1985.

[Ols92] Jan Olszewski. A flexible thinning algorithm allowing parallel, sequential, and distributed application. *ACM Transactions on Mathematical Software*, 18(1):35–45, March 1992.

[PB90] Srinivas Patil and Prithviraj Banerjee. A parallel branch and bound algorithm for test generation. *IEEE Transactions on Computer Aided Design*, 1990.

[PFK91] Curt Powley, Chris Ferguson, and Richard E. Korf. Parallel tree search on a SIMD machine. In *Third IEEE Symposium on Parallel and Distributed Processing*, Dallas, Texas, USA, December 1991. IEEE.

[PFK93] Curt Powley, Chris Ferguson, and Richard Korf. Depth-first heuristic search on a SIMD machine. *Artificial Intelligence*, 60, 1993.

[Phi93] Laurent Philippe. *Contribution à l'étude et la réalisation s'un système d'exploitation à image unique pour multicalculateur.* PhD thesis, Université de Franche-Comté, Besançon, France, January 1993.

[Pre92] Lutz Prechelt. Measurements of MasPar MP-1216A communication operations. Technical report, Universität Karlsruhe, 1992.

[PSL90] A. Pothen, H.D. Simon, and K.-P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal of Matrix Analysis*, 11(3):430–452, 1990.

[PTS88] Spiridon Pulidas, Don Towsley, and John A. Stankovic. Imbedding gradient estimators in load balancing algorithms. In *8th International Conference on Distributed Computing Systems*, pages 482–490, San Jose, California, USA, 1988.

[QY94] Xiaoshu Qian and Qing Yang. An analytical model for load balancing on symmetric multiprocessor systems. *Journal of Parallel and Distributed Computing*, 20:198–211, 1994.

[RAA+88] Marc Rozier, Vadim Abrossimov, François Armand, Ivan Boule, and Michel Gien. CHORUS distributed operating systems. *Computing Systems Journal*, 1(4):305–370, 1988.

[Rab76] M.O. Rabin. *Algorithms and Complexity*, chapter Probalistic Algorithms. Academic Press, New-York, 1976.

[RK87] V. Nageshwara Rao and Vipin Kumar. Parallel depth first search part I. *Int'l Journal of Parallel Programming*, 16(6):479–499, 1987.

[RM90] A. Ross and B. McMillin. Experimental comparison of bidding and drafting load sharing protocols. In *Proceedings of the 5th Distributed Memory Computing Conference*, pages 968–974, April 1990.

[RO93] Gowri Ramanathan and Joel Oren. Survey of commercial parallel machines. Technical report, Computer Science Department, Oregon State University, March 1993.

[RSZ89] Krithi Ramamritham, John A. Stankovic, and Wei Zhao. Distributed scheduling of tasks with deadlines and resource requirements. *IEEE Transactions on Computers*, 38(8):1110–1123, August 1989.

[Sal90] V.A. Saletore. A distributive and adaptive dynamic load balancing scheme for parallel processing of medium-grain tasks. In *Proceedings of the 5th Distributed Memory Conference*, pages 995–999, April 1990.

[SC88] Kang G. Shin and Y. Chang. Load sharing in distributed real-time systems with state-change broadcasts. *IEEE Transactions on Computers*, 38(8):1124–1142, August 1988.

[Sed88] Robert Sedgewick. *Algorithms*. Addison-Wesley Publishing Company, 2nd edition edition, 1988.

[SK90a] Niranjan G. Shivaratri and Phillip Krueger. Adaptive location policies for global scheduling. In *Proceedings of the 10th International Conference on Distributed Computing Systems*, pages 502–509, Los Alamitos, California, USA, 1990. IEEE CS Press.

[SK90b] Niranjan G. Shivaratri and Phillip Krueger. Two adaptive location policies for global scheduling algorithms. In *10th International Conference on Distributed Computing Systems*, pages 502–509, Paris, France, 1990.

[SKS92] Niranjan G. Shivaratri, Phillip Krueger, and Mukesh Singhal. Load distributing for locally distributed systems. *IEEE Computer*, pages 33–44, December 1992.

[Smi80] Reid G. Smith. The contract net protocol: High level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, December 1980.

[Son94] Jianjian Song. A partially asynchronous and iterative algorithm for distributed load balancing. *Parallel Computing*, 20:835–868, 1994.

[SS84] J.A. Stankovic and I.S. Sidhu. An adaptive bidding algorithm for processes, clusters and distributed groups. In *IEEE 4th International Conference on Distributed Computing Systems*, pages 49–59, 1984.

[ST85] C.C. Shen and W.H. Tsai. A graph matching approach to optimal task assignment in distributed computing systems using a minmax criterion. *IEEE Transactions on Computers*, C-34(3):197–203, March 1985.

[Sta84] John Stankovic. Simulations of three adaptive decentralized controlled, job scheduling algorithms. *Computer Networks*, 8(3):199–217, June 1984.

[Sta85] John Stankovic. An application of bayesian decision theory to decentralized control of job scheduling. *IEEE Transactions on Computers*, C-34(2):1141–1151, February 1985.

[Stu88] M. Stumm. The design and implementation of a decentralized scheduling facility for a workstation cluster. In *Proceedings of the Second Conference on Computer Workstations*, page 12, Los Alamitos, California, 1988. IEEE CS Press.

[SU87] Eli Shamir and Eli Upfal. A probabilistic approach to the load-sharing problem in distributed systems. *Journal of Parallel and Distributed Computing*, 4:521–530, 1987.

[SW91] Mark Smith and Greg Wilson. Dynamic load-balancing on a one-dimensional mesh. Technical report, Edinburgh Parallel Computing Centre, April 1991.

[Tan92] Andrew Tanenbaum. *Modern Operating Systems.* Prentice Hall Inc, 1992. Publié en français sous le titre "Systèmes d'exploitation" Systèmes centralisés, Systèmes distribués.

[TZB94] Stefan Tritscher, Roman Zajcew, and Michael Barnett. Load leveling on the paragon multicomputer. In Wolfgang Gentzsch and Uwe Harms, editors, *High-Performance Computing and Networking*, volume 797 of *Lecture Notes in Computer Science*, pages 330–337, Munich, Germany, April 1994. HPCN Europe, Springer-Verlag.

[UC94] Gil Utard and Guy Cuvillier. Compilation de programmes data-parallèles pour un réseau de stations de travail. Communication personelle, 1994.

[Var62] R.S. Varga. *Matrix Iterative Analysis.* Prentice-Hall, 1962.

[Vor86] Oliver Vornberger. Implementing branch-&-bound in a ring of processors. In *Conference on Algorithms and Hardware for Parallel Processing*, volume 237, pages 157–164, Aachen, Germany, September 1986. CONPAR, Springer.

[Vor90] Oliver Vornberger. Load balancing in a networks of Transputers. Technical report, Dept. of Mathematics/Computer Science, University of Paderborn, Germany, 1990.

[WLR89] Marc H. Willebeek-LeMair and Anthony P. Reeves. Dynamic load balancing for highly parallel multicomputer systems. Technical report, Cornell University, Ithaca, USA, December 1989.

[WLR93] Mark H. Willebeek-LeMair and Anthony P. Reeves. Strategies for dynamic load balancing on highly parallel computers. *IEEE Transactions on Parallel and Distributed Systems*, 4(9):979–993, September 1993.

[XH93] Jian Xu and Kai Hwang. Heuristic methods for dynamic load balancing in a message-passing supercomputer. *Journal of Parallel and Distributed Computing*, 18(1):888–897, 1993.

[XL92] C.Z. Xu and F.C.M. Lau. Analysis of the generalized dimension exchange method for dynamic load balancing. *Journal of Parallel and Distributed Computing*, 16:385–393, 1992.

[XL94a] Cheng-Zhong Xu and Francis C. M. Lau. Decentralized remapping of data parallel computations with the generalized dimension exchange method. In *Scalable High-Performance Computing Conference*, pages 414–421, Knoxville, Tenessee, USA, May 1994.

[XL94b] Cheng-Zhong Xu and Francis C. M. Lau. The generalized dimension exchange method for load balancing in k-ary n-cubes and variants. 1994. To appear in Journal of Parallel and Distributed Processing.

[Zho88] Songnian Zhou. A trace-driven simulation study of dynamic load balancing. *IEEE Transactions on Software Engeneering*, 14(9):1327–1340, December 1988.

[ZWZD93] Songnian Zhou, Jingwen Wang, Xiaohu Zheng, and Pierre Delisle. UTOPIA: a load sharing facility for large, heterogeneous distributed computer systems. *Software — Practice and Experience*, 23(12):1305, December 1993.

Table des figures