Université de Montréal

Approches de parallélisation basées sur l'organisation de la mémoire pour des méthodes de séparations et évaluations progressives

par

M. Benoît Bourbeau

Département d'informatique et recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en informatique et recherche opérationnelle

Septembre, 1997

Université de Montréal
Faculté des études supérieures

Ce mémoire intitulé:

Approches de parallélisation basées sur l'organisation de la mémoire pour des méthodes de séparations et évaluations progressives

présenté par:

M. Benoît Bourbeau

a été évalué par un jury composé des personnes suivantes:

Jean-Yves Potvin ....., président-rapporteur
Théodor Gabriel Crainic ....., directeur de recherche
Bernard Gendron ....., codirecteur
Marc Feeley ....., membre du jury
....., ~~examinateur externe~~

Mémoire accepté le: 27-01 1998

SOMMAIRE

L'objectif de ce mémoire est de réaliser une étude comparative de trois approches de parallélisation de méthodes de séparations et évaluations progressives basées sur l'organisation de la mémoire. Ces approches sont développées dans l'optique de résoudre efficacement des problèmes dérivés de modèles de conception de réseaux. Nous sélectionnons un problème représentatif de cette classe, le problème de localisation multi-produits avec équilibrage, pour tester les performances des trois approches.

Nos approches de parallélisation sont adaptées à un environnement parallèle asynchrone à échange de messages, où les processeurs sont puissants et capables de stocker une grande quantité de données. Elles sont basées sur l'exploration d'une arborescence par un examen concurrent des sous-problèmes. Les approches proposées, qui diffèrent selon la répartition des pools, sont les suivantes:

Une approche centralisée : Tous les sous-problèmes générés sont conservés dans un seul pool. Toutes les sélections et les insertions de sous-problèmes se font par ce pool global.

Une approche décentralisée : Tous les sous-problèmes générés sont répartis sur plusieurs pools (un par processeur). Les sélections et les insertions de sous-problèmes sont réalisées dans chacun de ces pools.

Une approche hybride : Certains sous-problèmes générés sont répartis dans les pools locaux à chacun des processeurs, les autres étant stockés dans un pool global accessible par tous les processeurs.

Chaque approche est composée d'un ensemble de processus communicants: un coordonnateur et un nombre fixe de travailleurs. Le coordonnateur assure

un contrôle sur l'exploration de l'arborescence, facilite l'équilibrage de travail et coordonne les échanges d'information entre les travailleurs. Les travailleurs sont responsables de l'évaluation et de la séparation des sous-problèmes.

Nous réalisons des expérimentations sur un réseau de 16 stations de travail en utilisant plusieurs jeux de données du problème de localisation multi-produits avec équilibrage. Ces expérimentations visent d'abord à sélectionner, parmi plusieurs versions, une stratégie d'affectation plus prometteuse pour chacune des trois approches. Puis, nous comparons les performances relatives des trois approches. De ces expérimentations, nous tirons les conclusions suivantes:

- Pour chacune des trois approches, les tâches issues de la décomposition correspondent à:
  - l'examen d'un sous-problème (approche centralisée);
  - l'exploration d'une sous-arborescence partielle (approche décentralisée);
  - l'exploration d'une branche jusqu'à l'obtention d'une feuille (approche hybride).
- Peu importe l'approche utilisée, on n'observe aucune pénalité de recherche significative.
- Les performances relatives des trois approches dépendent d'un compromis entre le nombre de tâches générées et le rapport du nombre de processeurs sur le temps par tâche.
- Lorsque la granularité de calcul est forte et les arborescences relativement petites, l'approche centralisée est favorisée, car les deux autres ne génèrent pas suffisamment de tâches, et ainsi n'utilisent pas pleinement les processeurs.
- Lorsque la granularité de calcul est faible et les arborescences relativement grandes, l'approche décentralisée est favorisée car elle possède le plus petit rapport du nombre de processeurs sur le temps par tâche. En particulier,

dans l'approche centralisée, nous observons un goulot d'étranglement en raison de l'accès simultané des travailleurs au pool global.

Mots-clés: Séparations et évaluations progressives, parallélisme, mémoire distribuée, organisations de la mémoire, stratégies d'affectation, localisation multi-produits avec équilibrage.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

À mes parents

REMERCIEMENTS

Je tiens à remercier mon directeur de maîtrise, le Professeur Teodor Gabriel Crainic, pour m'avoir suggéré ce sujet de recherche et pour m'avoir soutenu et encouragé tout au long de ce projet. Je le remercie pour sa patience, sa générosité et ses conseils judicieux. Je le remercie tout particulièrement de m'avoir donné l'opportunité d'effectuer un stage en France.

Je tiens également à remercier mon codirecteur et grand ami, le Professeur Bernard Gendron, pour son support, sa grande disponibilité et ses encouragements durant la réalisation de ce travail. Je le remercie pour les nombreux et fructueux échanges.

Je tiens à remercier le Professeur Catherine Roucairol du laboratoire PRiSM de l'Université de Versailles pour m'avoir accueilli au sein de son équipe durant mon stage de maîtrise. Je profite de l'occasion pour remercier les membres du laboratoire PRiSM, en particulier ceux de l'équipe PNN, pour leur support et l'amitié qu'ils m'ont témoigné.

Je remercie le Fonds pour la formation des chercheurs et l'aide à la recherche (FCAR) pour leur soutien financier sans lequel mon stage de maîtrise n'aurait été possible.

Je souhaiterais remercier le Professeur Michael Florian pour avoir suscité en moi un goût pour l'optimisation mathématique et pour m'avoir fait connaître le Centre de recherche sur les transports.

Je remercie le personnel du C.R.T. pour leur grande disponibilité et leur gentillesse à mon égard. Je remercie aussi le Centre pour l'utilisation de ses ressources informatiques.

Je remercie mon grand ami Luc Deneault pour ses commentaires pertinents à la lecture de ce mémoire. Je remercie également Nathalie Talbot et Alexandre Le Bouthillier pour leur aide précieuse dans la préparation des expérimentations numériques. Je tiens aussi à remercier Steven Chamberland, étudiant au doctorat, pour m'avoir si souvent rappelé que dans la vie, il faut aussi manger.

Je tiens à remercier très chaleureusement Mme Geneviève Le Fustec et M. Édouard Turpin pour leur accueil, leur amitié et leur grande générosité lors de mon passage en France.

Enfin, je remercie mes parents et mes soeurs pour leur amour et leur support durant mes longues années études. Je tiens à remercier aussi "ma gang" pour leur écoute, leurs encouragements et pour tous les bons moments passés ensemble.

INTRODUCTION

Ce travail s'inscrit dans la mouvance de travaux récents visant la résolution efficace de problèmes d'optimisation combinatoire difficiles dérivés des *modèles de conception de réseaux* (Gendron 1994; Gendron, Crainic et Frangioni 1997). Ces modèles proviennent de nombreuses applications dans les domaines des transports et des télécommunications (Magnanti et Wong 1984; Minoux 1989; Gavish 1991; Balakrishnan, Magnanti, Shulman et Wong 1991), et interviennent souvent comme outils d'aide à la planification des réseaux (installation d'infrastructures, établissement d'itinéraires, localisation d'entrepôts). Sommairement, le problème de la conception optimale d'un réseau consiste à choisir un sous-ensembles d'arcs (ou de noeuds) dans le graphe sous-jacent au réseau, de façon à satisfaire certaines contraintes et à optimiser un objectif.

Les *méthodes de séparations et évaluations progressives* (SÉP) sont les plus couramment utilisées pour déterminer une solution optimale à ce type de problèmes. Ces méthodes, mieux connues sous le nom de "branch-and-bound", décomposent récursivement l'espace des solutions possibles jusqu'à ce que la meilleure soit isolée, un processus qu'on peut représenter sous la forme d'une arborescence. L'intérêt de ces méthodes réside dans le fait que l'espace de solutions n'a pas à être décomposé entièrement pour arriver à l'optimum. À l'aide de bornes évaluées pour chaque solution, on évite l'exploration de certaines portions de l'arborescence. Malgré cette décomposition "intelligente", plusieurs problèmes difficiles, tels les problèmes de conception de réseaux, requièrent une puissance de calcul élevée afin de résoudre en un temps raisonnable des exemplaires de grande taille. En effet, les arborescences générées sont souvent de grande taille et ce, même pour des problèmes de dimensions relativement modestes. De plus, les méthodes d'évaluation de bornes à chaque noeud de l'arborescence sont com-

plexes et nécessitent une grande quantité de calculs (on dit aussi qu'elles sont à granularité forte). Dans ce contexte, le *parallélisme* apparaît comme un outil privilégié pour accélérer la résolution de ces problèmes.

Le parallélisme se situe dans le prolongement direct des efforts accomplis par les chercheurs en informatique depuis les débuts de cette discipline dans leur quête vers la conception d'ordinateurs toujours plus puissants. Ce n'est que récemment que des innovations technologiques ont permis de réaliser des machines parallèles. Le développement de telles machines ainsi que de l'algorithmique distribuée qui l'accompagne, permettent aujourd'hui d'accélérer grandement la résolution de problèmes complexes. Des défis de taille se présentent toutefois au concepteur d'algorithmes parallèles: bien coordonner les activités entre les processeurs, ainsi que les communications inter-processeurs, de manière à effectuer un travail coopératif et à répartir le travail le plus uniformément possible. Afin d'être efficace, une telle conception doit de plus être adaptée à l'architecture de la machine parallèle.

Le parallélisme peut s'adapter de plusieurs façons aux méthodes de SÉP pour donner naissance à des *méthodes parallèles de SÉP*. Les méthodes développées dans ce travail sont basées sur l'exploration d'une arborescence réalisée par un examen concurrent des solutions. La conception de telles méthodes parallèles pose des problèmes intéressants au niveau de la répartition des solutions en attente d'exploration et du contrôle de l'exploration, et nécessite le développement de plusieurs mécanismes de contrôle des activités. Ces activités comprennent la gestion des solutions, l'affectation des solutions aux processeurs, la répartition des opérations sur l'ensemble des processeurs, ainsi que l'échange et la diffusion des informations nécessaires à la coopération inter-processeurs. Le développement de ces mécanismes est contraint par plusieurs facteurs inhérents aux caractéristiques des méthodes de SÉP, de même qu'à celles de l'environnement parallèle. Ces mécanismes de contrôle, une fois clairement définis, donnent naissance à ce que nous appelons une *approche de parallélisation.*

Dans le cadre de ce mémoire, nous nous intéressons au développement d'approches de parallélisation adaptées à une architecture parallèle à mémoire distribuée: un groupe de processeurs interconnectés, capables de communiquer à l'aide de messages, et disposant chacun d'une mémoire locale. Sur l'ensemble de ces mémoires, plusieurs façons de conserver les solutions, pendant la construction de l'arborescence, peuvent être envisagées. Cet aspect des approches de parallélisation, qui constitue le coeur de notre étude, fait référence à ce que nous appelons l'*organisation de la mémoire.* Ainsi, nous proposons les trois approches suivantes:

- *Une approche centralisée*: Tous les sous-problèmes générés au cours de l'exploration de l'arborescence sont conservés dans la mémoire d'un seul processeur.
- *Une approche décentralisée*: Les sous-problèmes générés au cours de l'exploration de l'arborescence sont répartis dans la mémoire de chacun des processeurs.
- *Une approche hybride*: Certains sous-problèmes générés au cours de l'exploration sont répartis dans la mémoire de chacun des processeurs, les autres étant stockés dans la mémoire d'un seul processeur.

L'objectif principal de notre travail est de contribuer à la compréhension du fonctionnement de ces trois approches de parallélisation. Nous analysons l'impact de ces organisations sur la construction de l'arborescence et nous décrivons, pour chacune d'elles, les mécanismes qui permettent de diriger les activités. Enfin, nous testons ces approches en utilisant un cas particulier de problème de conception de réseaux: le problème de localisation multi-produits avec équilibrage.

Ce problème est résolu à l'aide d'un algorithme séquentiel de SÉP efficace (Gendron et Crainic 1995). Pour des problèmes de taille réaliste, les arborescences sont suffisamment grandes pour justifier l'utilisation du parallélisme. De plus, la procédure d'évaluation des bornes à chacun des sous-problèmes est à granularité forte. Ainsi, cet algorithme de SÉP possède des caractéristiques similaires à ceux

utilisés pour résoudre des problèmes plus difficiles de conception de réseaux, tels ceux décrits par Gendron, Crainic et Frangioni (1997). Il constitue par conséquent un candidat idéal pour l'étude de parallélisations d'algorithmes de SÉP adaptés à la résolution de problèmes de conception de réseaux.

Ce mémoire est divisé en cinq chapitres. Le chapitre 1 donne un aperçu des notions qui constituent la base de ce travail. Nous décrivons les méthodes de SÉP en situant leur utilisation et en décrivant leur fonctionnement. Nous exposons ensuite quelques notions élémentaires de parallélisme: les principales architectures parallèles, les méthodes de parallélisation et les mesures de performance. Nous voyons ensuite de quelle façon le parallélisme peut être inséré dans le contexte des méthodes de SÉP.

Dans le chapitre 2, nous présentons le problème de localisation multi-produits avec équilibrage, ainsi que son modèle, puis nous décrivons l'algorithme séquentiel de SÉP conçu par Gendron et Crainic (1995). Nous spécifions ensuite le cadre méthodologique qui a servi de base au développement de nos approches de parallélisation. Enfin, nous faisons une courte revue des efforts déjà investis dans le développement d'approches de parallélisation semblables à celles que nous étudions.

Le chapitre 3 expose les trois approches de parallélisation basées sur l'organisation de la mémoire. Nous décrivons tour à tour les approches centralisée et décentralisée, de même que l'approche hybride. Pour chacune d'elles, nous exposons leurs mécanismes de contrôle sous-jacents.

Le chapitre 4 décrit la mise en oeuvre des trois approches de parallélisation. Nous présentons l'environnement qui a servi de support à cette mise en oeuvre, puis nous enchaînons avec une description détaillée des implantations des trois approches de parallélisation.

Le chapitre 5 analyse les résultats d'expérimentations numériques réalisées avec les implantations des trois approches de parallélisation. Le chapitre est divisé

en huit sections. La première traite des jeux de données utilisés pour tester les implantations. La seconde décrit l'environnement dans lequel nous avons réalisé les expérimentations. La troisième section traite des mesures de performance que nous utilisons afin d'analyser les parallélisations. Les trois sections suivantes exposent les résultats obtenus avec chacune des approches en faisant varier les paramètres propres à chacune. La septième section présente une étude comparative des performances des trois implantations. Enfin, la dernière section résume les conclusions qui se dégagent des expérimentations.

CHAPITRE 1

Notions préliminaires

Dans ce chapitre, nous donnons un aperçu des notions qui constituent la base de ce mémoire. Nous commençons par une description générale des méthodes de SÉP. Nous exposons ensuite quelques notions élémentaires de parallélisme: nous faisons un survol des *architectures parallèles* et des *méthodes de parallélisation*, puis nous introduisons quelques *mesures de performance* des implantations parallèles. Nous voyons ensuite comment s'insère le parallélisme dans le contexte des méthodes de SÉP.

1.1 Description des méthodes de SÉP

Les méthodes de SÉP (également connues sous le nom de méthodes de "branch-and-bound") ont pour but de résoudre de manière exacte des problèmes d'optimisation difficiles qui peuvent être modélisés à l'aide de variables à valeurs entières.

Ces méthodes consistent à énumérer potentiellement toutes les solutions réalisables d'un problème en vue de déterminer une solution optimale. Lorsque la taille du problème est élevée, une énumération exhaustive de toutes les solutions réalisables devient une tâche ardue, en particulier lorsque le nombre de solutions réalisables croît de manière exponentielle avec la taille du problème. Pour lutter contre ce phénomène, les méthodes de SÉP partitionnent récursivement l'espace des solutions jusqu'à ce qu'une solution optimale soit isolée, en éliminant certaines régions grâce à l'information générée au fur et à mesure du processus de

partitionnement (on dit aussi "séparation"). Cette information consiste essentiellement en des bornes (ou "évaluations") sur la valeur optimale du problème restreint à un sous-espace des solutions réalisables.

Nous présentons une méthode générique de SÉP en utilisant la description introduite par Gendron et Crainic (1994). Cette description présente la méthode comme un ensemble d'opérations détachées de tout enchaînement particulier, ce qui constitue un avantage afin de concevoir des algorithmes parallèles basés sur cette méthode.

L'algorithme de SÉP peut être vu comme une méthode d'énumération pour résoudre un problème d'optimisation P:

$$Z(P) = \min_{x \in S} f(x)$$

où f est une fonction réelle, et S un sous-ensemble d'un espace vectoriel réel V qui forme l'ensemble des solutions réalisables, ou *domaine réalisable*, de P. On suppose que P peut être résolu en énumérant un nombre fini de points (pas nécessairement connus à l'avance) dans S. On suppose également que le problème est soit non-réalisable ($S = \emptyset$), soit possède une valeur optimale finie ($Z(P) > -\infty$).

Lorsque le problème ne peut être résolu directement, une approche *diviser-pour-régner* (Brassard et Bratley 1987) peut être utilisée afin de le résoudre. Elle consiste à partitionner l'ensemble des solutions réalisables en n sous-ensembles $S_1, ..., S_n$ ($\bigcup_{i=1}^{n} S_i = S$ et $S_i \bigcap S_j = \emptyset,\ i \neq j$). Dénotons par P_i le problème d'optimisation associé au sous-ensemble S_i, et $Z(P_i)$ sa valeur optimale ($i = 1, ..., n$). Nous pouvons alors reformuler le problème de la façon suivante:

$$Z(P) = \min_{1 \leq i \leq n} Z(P_i)$$

(avec la convention qu'un problème non-réalisable possède une valeur optimale infinie). Si les sous-problèmes $P_1, ..., P_n$ ne sont pas résolus directement, une

décomposition similaire appelée *opération de séparation* peut être appliquée sur chacun d'eux. Les décompositions successives peuvent être appliquées jusqu'à ce que tous les sous-problèmes obtenus deviennent faciles à résoudre (nous verrons cependant que, dans la plupart des cas, il n'est pas nécessaire de résoudre tous les sous-problèmes). On suppose aussi que le processus génère un nombre fini de sous-problèmes, puisque P peut être résolu en énumérant un nombre fini de points de S.

Il n'est pas nécessaire de décomposer un sous-problème Q, s'il obéit à une des deux règles suivantes:

- *règle d'élimination 1*: Le sous-problème a été résolu, c'est-à-dire qu'il est soit non-réalisable, soit on a déterminé une solution optimale.
- *règle d'élimination 2*: Il existe un autre sous-problème R, ayant une valeur optimale inférieure ou au plus égale à la valeur optimale du sous-problème donné ($Z(R) \leq Z(Q)$).

En vue d'appliquer la règle d'élimination 2, il semble qu'on doive connaître la valeur optimale des sous-problèmes. Cependant, il est suffisant de connaître une borne inférieure sur la valeur optimale de Q ainsi qu'une borne supérieure sur la valeur optimale de R. Ceci constitue le rôle de l'*opération d'évaluation.* Celle-ci associe à chacun des sous-problèmes Q une borne inférieure $Z^l(Q)$ et une borne supérieure $Z^u(Q)$. Normalement, une borne supérieure finie pour un sous-problème Q correspond aussi à une solution réalisable de ce sous-problème, et ainsi, du problème original P. Alors, si un sous-problème donné Q possède une borne inférieure qui est supérieure ou égale à une borne supérieure connue sur la valeur optimale du problème, alors il n'a pas besoin d'être décomposé. Cette règle est appelée le *test de la borne inférieure.*

L'algorithme de SÉP consiste à appliquer les opérations de séparation et d'évaluation, ainsi que les règles d'élimination, et peut être décrit comme un processus de construction d'une arborescence, appelée *arborescence de SÉP.* La racine de cette arborescence correspond au problème original, alors que les fils d'un sous-

problème donné Q correspondent aux sous-problèmes obtenus par décomposition de Q. Les feuilles de l'arborescence sont les sous-problèmes qui n'ont pu être décomposés.

Lors de la construction de l'arborescence, les sous-problèmes peuvent se trouver dans l'un des trois états suivants: généré, évalué ou examiné. Un sous-problème est *généré* lorsqu'il est obtenu d'un autre sous-problème par décomposition (initialement, le problème original est le seul problème généré). Un sous-problème généré est *évalué* quand une opération d'évaluation est appliquée à celui-ci, alors qu'il est *examiné* dans le cas où une opération de séparation a été appliquée sur ce dernier (on dit alors qu'il fût *décomposé*), ou lorsque les règles d'élimination ont montré qu'il était inutile de le décomposer (on dit alors qu'il fût *éliminé* ou *élagué*).

Pour résoudre un problème donné, il est possible de définir plusieurs opérations d'évaluation, l'objectif principal étant d'obtenir des bornes *serrées* (c'est-à-dire près de la valeur optimale), le tout en réduisant le temps de calcul. En particulier, l'obtention d'une borne supérieure de qualité dès les premiers niveaux de l'arborescence tend à réduire le nombre de sous-problèmes générés. Afin de définir des opérations d'évaluation efficaces, on fait appel à des méthodes de programmation mathématique. Deux classes de méthodes se distinguent pour l'évaluation des bornes inférieures et supérieures respectivement: les méthodes de relaxation et les méthodes heuristiques.

Le principe des *méthodes de relaxation* consiste à déterminer une borne inférieure sur la valeur optimale du problème en retirant un sous-ensemble des contraintes du modèle et en pénalisant la violation de ces contraintes. Un choix judicieux de ce sous-ensemble de contraintes et un ajustement efficace des facteurs de pénalité permettent de déterminer une borne inférieure de qualité et de faciliter son évaluation. Les *méthodes heuristiques* permettent de déterminer une solution réalisable, fournissant ainsi une borne supérieure sur la valeur optimale. Ces méthodes sont généralement basées sur une exploration partielle des solutions

réalisables.

L'exploration d'une arborescence de SÉP peut être dirigée grâce à une *règle de sélection* ou *règle de parcours.* Deux règles de sélection principales sont utilisées dans les algorithmes de SÉP implantés en environnement séquentiel: la règle *meilleur d'abord* et la *règle profondeur d'abord.*

Selon la règle meilleur d'abord, la valeur de la meilleure borne supérieure est gardée à jour à l'aide d'une variable ($\overline{Z^u}$) qui permet une exécution rapide du test de la borne inférieure. Une liste L des sous-problèmes évalués mais non-examinés est conservée en mémoire. Au départ, $\overline{Z^u}$ est initialisée à l'infini et le problème original est évalué. Dans le cas où il n'est pas résolu, il est inséré dans la liste. À chaque étape, si L est non-vide, un sous-problème Q ayant la plus petite borne inférieure parmi tous les sous-problèmes déjà dans L, est sélectionné. Le test de la borne inférieure est appliqué à Q, et dans le cas où Q n'est pas éliminé, il est décomposé suivant l'opération de séparation. Chacun des sous-problèmes nouvellement générés est alors évalué, et si sa borne supérieure est meilleure que la valeur courante de $\overline{Z^u}$, elle la remplace. Les deux règles d'élimination sont aussi appliquées sur les sous-problèmes nouvellement générés, qui sont insérés dans la liste s'ils ne sont pas éliminés. Le processus se poursuit ainsi jusqu'à ce que la liste L soit vide. Cette règle de sélection peut être implantée efficacement en gérant la liste à l'aide d'une structure de monceau (Standish 1980). Le principal avantage de l'approche meilleur d'abord par rapport aux autres règles de sélection est qu'elle minimise le nombre de sous-problèmes décomposés, lorsque deux conditions sont satisfaites: la liste L ne contient pas de sous-problèmes ayant la même borne inférieure; les opérations de séparation et d'évaluation ne dépendent pas de l'historique du processus (Fox, Lenstra, Rinnooy Kan et Schrage 1978). Un désavantage de cette méthode est la quantité importante d'espace mémoire requise par la liste L.

La règle de sélection *profondeur d'abord* définit un autre ordre d'exécution des opérations. Dans ce cas, la liste L représente l'ensemble des sous-problèmes

non-évalués et non-examinés. Au départ, $\overline{Z^u}$ est initialisé à l'infini, le problème original est évalué et, s'il n'est pas résolu, est décomposé selon l'opération de séparation. À chaque étape, si le dernier sous-problème examiné est décomposé, les fils de ce sous-problème sont ajoutés à la liste, sauf un, qui est évalué et examiné. Si le dernier sous-problème examiné fut éliminé, l'opération de sélection consiste à choisir un sous-problème dans L parmi ceux qui ont été générés en dernier. Ce sous-problème est alors évalué et examiné. Chaque fois qu'un sous-problème est évalué, la variable $\overline{Z^u}$ est mise à jour si nécessaire. Le processus se poursuit ainsi jusqu'à ce que la liste L soit vide. Une structure de pile (Standish 1980) peut être utilisée afin d'implanter efficacement la gestion de la liste L. Il y a trois avantages en faveur de l'utilisation de cette approche. Premièrement, parmi toutes les opérations de sélection, elle minimise l'espace mémoire requis pour une exécution complète de l'algorithme (Ibaraki 1987). Deuxièmement, quand un sous-problème n'est pas éliminé, une partie significative de l'information générée par la dernière opération d'évaluation peut être exploitée afin de faciliter l'évaluation du prochain sous-problème, qui est un fils du dernier noeud évalué (Nemhauser et Wolsey 1988). En comparaison, une telle opportunité de réoptimisation est plus difficile à implanter lorsque l'approche meilleur d'abord est utilisée. Troisièmement, les solutions réalisables sont généralement trouvées plus rapidement qu'avec les autres règles de sélection (Ibaraki 1987, Nemhauser et Wolsey 1988). Un désavantage de cette méthode est qu'elle peut générer un nombre élevé de sous-problèmes. Ce désavantage peut être réduit en identifiant une borne supérieure de qualité rapidement, ce qui a pour effet d'améliorer le test de la borne inférieure, réduisant ainsi le nombre de sous-problèmes générés.

1.2 Notions de parallélisme

Les machines parallèles sont constituées d'un ensemble de processeurs capables de communiquer entre eux tout en effectuant des calculs simultanément. Tout en travaillant, ces processeurs peuvent ainsi coopérer, via la communica-

tion, à la résolution concurrente d'un même problème, réduisant ainsi le temps de calcul requis à sa résolution par un seul processeur. Ces deux principes, la *coopération* et la *concurrence*, sont à la base du développement des architectures et des algorithmes parallèles.

Dans cette section, nous faisons d'abord un survol des architectures parallèles. Nous traitons ensuite des méthodes de parallélisation des algorithmes séquentiels, en particulier dans le contexte des algorithmes de SÉP. Nous concluons cette section en décrivant quelques mesures de performances qui permettent notamment d'évaluer les gains reliés à l'utilisation d'un environnement parallèle par rapport à celle d'un environnement séquentiel.

1.2.1 Architectures parallèles

Michael J. Flynn (1966) a proposé une classification des architectures d'ordinateurs basée sur la notion de flot d'informations. Plus précisément, sa classification repose sur deux types de flot d'informations: le *flot de données* (D) et le *flot d'instructions* (I). Chacun de ces flots peut être *simple* (S) ou *multiple* (M). La classification contient alors quatre classes distinctes: SISD, SIMD, MISD et MIMD.

La classe SISD ("Single Instruction Single Data") correspond à l'ensemble des *machines séquentielles* basées sur le modèle de Von Neumann. Ces machines sont caractérisées par un flot unique d'instructions et de données. De façon synchronisée, chaque donnée est traitée par une instruction. Les classes suivantes concernent les *machines parallèles.* Nous voyons que la multiplicité des flots d'informations caractérise ce type de machines. La classe de machines SIMD ("Single Instruction Multiple Data") est caractérisée par un flot unique d'instructions et un flot multiple de données. Dans ce contexte, une seule instruction est exécutée sur plusieurs données à la fois. La classe de machines MISD ("Multiple Instruction Single Data") est caractérisée, quant à elle, par un flot multiple d'instructions et un flot unique de données. Dans ce type de machines, une donnée est traitée

simultanément par plusieurs instructions. On ne connaît pas à ce jour de machines respectant ce type d'organisation. Enfin, la classe de machines MIMD ("Multiple Instruction Multiple Data") englobe les machines caractérisées par un flot multiple de données et d'instructions. Avec ce type d'architecture, plusieurs données distinctes sont manipulées simultanément par plusieurs instructions à la fois. Conceptuellement, on peut représenter plusieurs machines de cette classe par un ensemble de machines de la classe SISD.

Bien que la classification de Flynn donne un aperçu de l'architecture des machines parallèles, il est nécessaire d'ajouter à cette classification un certain nombre de paramètres permettant de bien décrire les machines parallèles. Cette liste de paramètres, tirée de Bertsekas et Tsitsiklis (1989), nous permet aussi d'introduire un certain nombre de termes reliés au parallélisme.

Le *contrôle* fait référence à la présence ou à l'absence d'une unité de contrôle globale. Dans notre cas, nous considérons seulement les architectures parallèles bâties selon le modèle de *flot de contrôle*: chaque processeur exécute les instructions dans un ordre déterminé par l'unité de contrôle. Les autres modèles proposés à ce jour sont les modèles à *flot de données*, dans lesquels les processeurs exécutent les instructions selon la disponibilité des données, ainsi que les modèles à *flot de demande*, dans lesquels les processeurs exécutent les instructions selon un ordre déterminé par la demande pour des données (voir Treleaven, Brownbridge et Hopkins 1982, pour une description plus complète de ces modèles). Les architectures parallèles à flot de contrôle ne possédant qu'une seule unité de contrôle appartiennent à la classe SIMD, alors que les systèmes en possédant plusieurs appartiennent à la classe MIMD.

La *synchronisation* fait référence à la présence ou à l'absence d'une horloge globale utilisée pour la synchronisation des opérations entre les processeurs. Lorsqu'il n'y a qu'une horloge, on parle de système *synchrone*, alors que la présence de plusieurs horloges, typiquement une par processeur, nous amène à parler de système *asynchrone*. Les machines de la classe SIMD sont synchrones par

définition, alors que celles de la classe MIMD sont généralement asynchrones.

La *granularité* fait référence à la quantité de données que chacun des processeurs d'un système peut gérer. Dans les systèmes à *granularité fine*, chaque processeur ne peut gérer qu'une petite quantité de données, correspondant à des opérations scalaires ou vectorielles. À l'autre extrême, les systèmes à *granularité forte* sont caractérisés par la possibilité d'un traitement simultané sur de grandes quantités de données.

La *communication* fait référence à la façon dont les processeurs échangent de l'information. On distingue généralement deux approches: les processeurs peuvent écrire et lire dans une mémoire commune accessible par tous (systèmes à *mémoire partagée*), ou alors ils peuvent échanger des messages (systèmes à *échange de messages*). Les systèmes à mémoire partagée sont caractérisés soit par la présence d'une mémoire commune physiquement réalisée (systèmes *fortement couplés*), soit par un mécanisme permettant l'accès de chaque processeur à n'importe quelle région de la mémoire (systèmes *faiblement couplés*). Les systèmes à échange de messages sont caractérisés par leur *topologie du réseau d'interconnexion*, qui décrit comment les processeurs sont reliés entre eux. Les topologies les plus courantes sont l'anneau, l'arborescence, la grille et l'hypercube (pour des détails additionnels, voir Bertsekas et Tsitsiklis 1989).

Le *nombre de processeurs* constitue le dernier paramètre. Les systèmes *massivement parallèles* sont constitués d'un grand nombre de processeurs, de l'ordre des milliers. Les systèmes à granularité fine sont habituellement massivement parallèles, alors que les systèmes à granularité forte possèdent généralement moins de processeurs, disons dans l'ordre des dizaines (mais la situation évolue rapidement; avec les progrès technologiques, il existe maintenant des systèmes à granularité forte comportant des centaines et même des milliers de processeurs). Il est à noter que les systèmes à mémoire partagée fortement couplés sont généralement contraints à posséder un nombre faible de processeurs, disons une vingtaine, en raison de la difficulté d'implanter des mécanismes d'accès simultanés à la mémoire

sans provoquer des goulots d'étranglement.

Parmi les systèmes parallèles, on distingue également les *systèmes distribués* qui sont caractérisés principalement par une distance inter-processeurs relativement grande et un réseau de communication impliquant des délais imprévisibles. De plus, la topologie inter-processeurs peut changer au cours d'une exécution en raison d'une panne de communication. Les systèmes distribués sont généralement *faiblement couplés*, chaque processeur pouvant effectuer des calculs indépendamment des autres alors qu'ils coopèrent afin de résoudre un problème.

Les *réseaux de stations de travail* constituent un exemple de système distribué. Chaque station de travail possède sa propre mémoire locale, une puissance ainsi qu'une granularité relativement élevées. Les stations sont reliées par un réseau de communication. Les communications sont souvent assurées par un logiciel permettant l'échange de messages (la bibliothèque de fonctions PVM constitue un exemple de ce type de logiciel; voir Geist, Beguelin, Dongarra, Jiang, Manchek, Sunderam 1993). On pourrait inclure ces réseaux de stations de travail parmi les machines MIMD asynchrones à mémoire distribuée.

1.2.2 Méthodes de parallélisation

Comme nous l'avons vu à la section précédente, les architectures parallèles sont toujours basées sur un flot multiple de données ou d'instructions. S'inspirant de cette idée, il est naturel de voir la parallélisation d'un algorithme séquentiel comme un processus de décomposition de données et d'instructions. Deux approches ont été proposées (Foster 1994). La première approche (*décomposition des données*) consiste à diviser un algorithme par l'analyse des données qu'il manipule, tandis que la deuxième approche (*décomposition fonctionnelle*) est dirigée par l'analyse des instructions à exécuter. Chacune de ces approches donne naissance à un ensemble de tâches, chacune étant composée de données et d'instructions, pouvant s'exécuter simultanément tout en coopérant par échange d'informations.

Le principe de la décomposition des données consiste à diviser en parties égales (ou adaptées à la granularité des processeurs) les structures de données qui présentent une certaine régularité et qui sont manipulées par un bloc d'instructions. Les processeurs sont programmés pour effectuer le bloc d'instructions et chacune des divisions des structures concernées est affectée à un processeur. Une tâche correspond donc à l'exécution du bloc d'instructions sur une division.

La décomposition fonctionnelle s'intéresse, quant à elle, aux opérations effectuées par un algorithme. Dans certains cas, l'analyse d'un algorithme permet d'identifier des ensembles d'opérations (ou instructions) pouvant s'exécuter concurremment. Une fois que les processeurs sont programmés de façon à exécuter ces blocs d'instructions, on leur affecte les données requises à leur traitement respectif. De la même façon, on donne naissance à un ensemble de tâches. Avec ce genre de décomposition, il est souvent nécessaire d'établir des communications entre les tâches afin qu'elles puissent partager l'information dont elles ont besoin pour leurs traitements.

Étant donné que la parallélisation d'un algorithme séquentiel dépend en grande partie de sa structure particulière, il apparaît exclu de donner des règles générales de décomposition. Par conséquent, nous illustrons les deux types de décomposition à partir de l'algorithme de SÉP vu à la section 1.1.

Nous avons vu que cette méthode correspond à un processus de construction d'une arborescence de sous-problèmes évalués et sélectionnés dans un ordre déterminé. Gendron et Crainic (1994) ont identifié trois types de parallélisme pouvant s'intégrer à l'algorithme:

- Type 1: parallélisation d'une ou plusieurs opérations reliées au traitement d'un sous-problème;
- Type 2: parallélisation de l'exploration de l'arborescence de SÉP (plusieurs sous-problèmes sont examinés concurremment);
- Type 3: exploration simultanée de plusieurs arborescences de SÉP.

Le parallélisme de type 1 s'insère au niveau des opérations exécutées sur les sous-problèmes. Il consiste par exemple à exécuter l'opération d'évaluation en parallèle sur plusieurs processeurs de façon à accélérer la résolution. La plupart des algorithmes appartenant à ce type de parallélisme sont basés sur une décomposition des données. Lorsque l'opération parallélisée débute, les données associées à un sous-problème sélectionné sont divisées en parts relativement égales et chaque part est affectée à un processeur. Les résultats obtenus par le traitement de chaque part sont mis en commun. Dans de tels algorithmes, l'ordre dans lequel les sous-problèmes sont explorés demeure le même que dans l'algorithme séquentiel, ce qui implique que ce type de parallélisme n'a pas d'influence sur la structure générale de l'algorithme de SÉP.

Le parallélisme de type 2 consiste en l'examen simultané de plusieurs sous-problèmes, l'exploration d'un sous-problème donné étant réalisée par un seul processeur. Dans ce contexte, les structures de données associées aux sous-problèmes ne sont pas décomposées. Plutôt, on exploite le fait que l'examen d'un sous-problème (évaluation et séparation) puisse se faire en concurrence avec celui d'un autre. On peut donc explorer autant de sous-problèmes concurremment qu'il y a de processeurs disponibles. Dans ce type de parallélisme, l'ordre dans lequel les sous-problèmes sont explorés peut différer de celui de l'algorithme séquentiel. Cette parallélisation est, par conséquent, basée sur une décomposition fonctionnelle où les instructions reliées à l'examen d'un sous-problème sont exécutées concurremment. Dans ce contexte, une tâche correspond justement à l'examen d'un sous-problème.

Le parallélisme de type 3 consiste à construire concurremment plusieurs arborescences de SÉP à partir du problème à résoudre, chacune d'elle étant générée par un algorithme de SÉP possédant son ensemble distinct de règles et d'opérations. En partageant l'information générée par chacune des recherches, on espère globalement accélérer la résolution. Dans ce contexte, une tâche correspond à l'exploration complète d'une arborescence.

Une fois la décomposition appliquée à un algorithme séquentiel, il est nécessaire de rendre accessibles les tâches créées par cette décomposition aux processeurs disponibles. De façon générale, rendre accessible une tâche à un processeur signifie que ce dernier est programmé pour exécuter une série d'instructions déterminées et doit avoir accès à des données qu'il peut traiter. Ainsi, donner accès à une tâche est souvent équivalent à donner accès aux données qui définissent la tâche. Avec les machines à mémoire partagée, l'accès aux tâches (ou aux données) se fait via la mémoire commune. On utilise généralement des structures de données adaptées à cet environnement de façon à organiser ces données en mémoire et à en faciliter l'accès et la gestion. Avec les machines à mémoire distribuée, l'accès aux tâches se fait via les échanges de messages. Chacun des processeurs disposant d'une certaine quantité de mémoire (appelée aussi mémoire locale), les données définissant les tâches peuvent être *centralisées* (dans une seule mémoire locale) ou *réparties* (dans plusieurs mémoires locales) sur l'ensemble des processeurs de la machine parallèle. Dans les deux cas, on doit développer un mécanisme permettant aux processeurs d'avoir accès aux tâches. Un tel mécanisme s'appelle une *stratégie d'affectation.*

L'objectif de la stratégie d'affectation consiste à minimiser le temps d'exécution d'une parallélisation. Pour atteindre ce but, il est soit nécessaire d'ordonnancer l'exécution des tâches entre les processeurs, soit d'équilibrer le travail sur l'ensemble des processeurs, soit une combinaison des deux. Élaborer une telle stratégie d'affectation optimale constitue un problème très difficile en raison de plusieurs facteurs reliés aux caractéristiques de l'application, de même qu'à celles de l'environnement parallèle (El-Rewini, Lewis et Ali 1994):

- *Le nombre de tâches*: Le nombre de tâches peut ne pas être connu après décomposition de l'algorithme séquentiel. Dans ce cas, les tâches sont générées en cours d'exécution. C'est le cas d'une parallélisation de type 2 de l'algorithme de SÉP. L'affectation des tâches ne peut être planifiée à l'avance, rendant ainsi non triviale l'atteinte de l'objectif de minimiser le temps d'exécution total.

- *La quantité de travail associée aux tâches*: Chaque tâche nécessite une certaine quantité de travail pour être complétée et cette quantité peut être variable d'une tâche à l'autre.
- *Les relations de précédence entre les tâches*: Une parallélisation peut être décomposée de façon à créer des tâches possédant des relations de précédence entre elles: deux tâches sont en relation de précédence si les résultats de la première sont nécessaires à l'exécution de la deuxième.
- *Les délais de communication*: La stratégie d'affectation doit tenir compte des délais de communication entre les processeurs impliqués dans une communication de tâches.
- *La puissance des processeurs*: Dépendant de la puissance des processeurs, le travail associé à une tâche peut être réalisé plus ou moins rapidement.

Les stratégies d'affectation peuvent être divisées en deux classes: les stratégies *statiques* et les stratégies *dynamiques*. Une stratégie d'affectation est dite statique si l'ensemble des tâches définies par la décomposition sont affectées avant le début du traitement parallèle, sans possibilité de mouvements de tâches entre les processeurs. Dans le cas où les affectations s'effectuent au cours du traitement parallèle et que des mouvements de tâches inter-processeurs sont permis, la stratégie d'affectation est qualifiée de dynamique.

Une stratégie d'affectation dynamique est composée de quatre politiques (Kumar A., Kumar C.B., Ullah et Szygenda 1995):

- La *politique de déclenchement* exploite un mécanisme de seuil permettant de distinguer les processeurs hautement chargés des processeurs faiblement chargés. Sachant cela, deux types de politiques de déclenchement peuvent être utilisées: à l'initiative de l'envoyeur et à l'initiative du receveur. Selon la politique à l'initiative de l'envoyeur, un processeur hautement chargé tente de localiser un processeur faiblement chargé afin de partager une partie de ses tâches. Inversement, selon la politique à l'initiative du receveur, un processeur faiblement chargé tente d'obtenir des tâches d'un processeur

hautement chargé.

- La *politique de sélection de tâches* consiste à choisir une tâche ou un groupe de tâches destinées à être exécutées. Ce choix peut être arbitraire ou dirigé selon un ordre établi. Par exemple, pour une parallélisation de type 1 obtenue par décomposition des données, le choix de la tâche pourrait être arbitraire étant donné que le travail est divisé en morceaux et que l'ensemble du travail doit être effectué. Par contre, pour une parallélisation de type 2, bien que l'examen simultané de plusieurs sous-problèmes soit possible, il est avantageux de choisir les sous-problèmes dans un ordre qui permet une exploration efficace de l'arborescence.
- La *politique de localisation* est responsable de localiser le receveur (ou le donneur) de tâches lorsqu'une politique de déclenchement à l'initiative de l'envoyeur (ou du receveur) est utilisée.
- La *politique d'information* s'intéresse à la gestion de l'information nécessaire aux politiques de localisation et de sélection.

1.2.3 Mesures de performance

Afin de juger de la qualité d'une parallélisation, il est souhaitable d'estimer les gains atteints par rapport à la version séquentielle, d'évaluer les niveaux d'activité des processeurs, de même que les surcoûts impliqués lors de l'exécution. À cette fin, nous utilisons des mesures de performance, qui permettent également de comparer entre elles plusieurs parallélisations d'un même algorithme.

La section présente dresse une liste des principales mesures de performance utilisées. Ces mesures de performances sont définies à partir des deux paramètres suivants:

- *Taille n*: La taille n correspond à la taille du problème à résoudre.
- *Nombre de processeurs p*: Ce paramètre représente le nombre de processeurs de la machine parallèle.

a) *Temps*

Le temps, noté par $T(n,p)$, représente la durée d'exécution d'un algorithme pour la résolution d'un problème de taille n sur p processeurs. Afin de distinguer le temps requis par un algorithme séquentiel du temps requis par un algorithme parallèle utilisant un seul processeur, nous utilisons respectivement les notations $T_s(n)$ et $T(n,1)$.

b) *Accélération* ("*speedup*")

L'accélération donne le facteur de gain en temps obtenu par l'utilisation d'une parallélisation par rapport à l'utilisation de la version séquentielle. L'accélération, notée par $S(n,p)$, correspond au ratio du temps séquentiel sur le temps parallèle selon le rapport $T_s(n)/T(n,p)$. Dans un article de Barr et Hickman (1989), on raffine la notion d'accélération en faisant des hypothèses sur les valeurs de $T_s(n)$ et $T(n,p)$:

- *Accélération (définition de base)*:
  $T_s(n)$: code séquentiel le plus rapide exécuté sur un processeur d'une machine parallèle particulière;
  $T(n,p)$: code parallèle utilisant p processeurs de la même machine.
- *Accélération relative*:
  $T_s(n)$: code parallèle avec $p = 1$ processeur exécuté sur une machine particulière ($T_s(n) = T(n,1)$);
  $T(n,p)$: code parallèle utilisant p processeurs de la même machine.
- *Accélération absolue*:
  $T_s(n)$: temps séquentiel le plus rapide obtenu (meilleur code séquentiel sur la machine la plus rapide);
  $T(n,p)$: code parallèle utilisant p processeurs.

En supposant qu'une parallélisation est composée de deux phases, dont l'une est parallélisée et l'autre non, $T(n,p)$ peut s'exprimer comme la somme de $T_P(n,p)$ et de $T_N(n)$, qui correspondent respectivement au temps requis par

chacune de ces deux phases. Si, de plus, la phase parallélisée présente une accélération "idéale" égale à p, $T_s(n)$ s'exprime alors comme la somme de $T_N(n)$ et de $p \cdot T_P(n,p)$. Sous ces hypothèses, l'accélération est limitée par l'expression $p/(1+(p-1)f_s(n))$, où $f_s(n) = T_N(n)/T_s(n)$ représente le rapport du temps requis par la phase non parallélisée sur le temps séquentiel. Cette expression est appelée *loi d'Amdahl* (Amdahl 1967).

c) *Efficacité*

L'efficacité tente de mesurer la fraction moyenne du temps où les processeurs demeurent actifs pendant une exécution parallèle. L'efficacité, notée par $E(n,p)$, est définie par le ratio de l'accélération sur le nombre de processeurs $S(n,p)/p$

d) *Surcoût de parallélisation absolu* ("absolute burden")

Le surcoût de parallélisation absolu, mesure introduite par Chabini (1994), donne l'écart de l'amélioration "idéale" d'une exécution séquentielle à une exécution à p processeurs. On considère qu'une parallélisation "idéale" est celle qui, utilisant p processeurs, s'exécute p fois plus rapidement que la version séquentielle. Le surcoût de parallélisation, noté par $A(n,p)$, est défini selon l'expression $T(n,p) - T_s(n)/p$

e) *Surcoût de parallélisation relatif* ("relative burden")

Le surcoût de parallélisation relatif, mesure qui découle de la précédente, est une mesure normalisée du surcoût de parallélisation absolu par rapport au temps séquentiel. Le surcoût de parallélisation relatif, noté par $B(n,p)$, est défini selon l'expression $A(n,p)/T_s(n)$. Il est à noter que le temps requis par une parallélisation peut être exprimée en fonction du surcoût de parallélisation relatif de la façon suivante: $T(n,p) = T_s(n)(1/p + B(n,p))$. De cette expression, on peut déduire la relation suivante entre le surcoût de parallélisation relatif et l'accélération: $B(n,p) = 1/S(n,p) - 1/p$.

f) *Travail*

Le travail mesure l'effort de calcul total requis par une parallélisation. Le travail, noté par $W(n,p)$, est défini par le nombre total d'instructions exécutées par tous les processeurs. En particulier, si $W_j(n,p)$ représente le nombre d'instructions exécutées par le processeur j, $W(n,p) = \sum_j W_j(n,p)$. Concrètement, il est plus aisé d'évaluer le travail en fonction du temps. C'est pourquoi on définit également $W_j^t(n,p) = T_j^A(n,p)$ où $T_j^A(n,p)$ correspond au temps d'activité du processeur j pendant le temps déterminé par $T(n,p)$. Conséquemment, on définit $W^t(n,p) = \sum_j W_j^t(n,p)$. Sachant que $T_j^A(n,p) = T(n,p) - T_j^I(n,p)$, où $T_j^I(n,p)$ correspond au temps d'inactivité du processeur j pendant le temps déterminé par $T(n,p)$, il est également possible de poser $W^t(n,p) = p \cdot T(n,p) - \sum_j T_j^I(n,p)$. C'est pourquoi on trouve dans la littérature, l'approximation $W^t(n,p) = p \cdot T(n,p)$ comme mesure du travail (Gengler, Ubéda et Desprez 1996). On observe que cette approximation est valable lorsque les temps d'inactivité des processeurs $(T_j^I(n,p))$ sont faibles.

g) *Utilisation*

L'utilisation fournit la proportion de travail effectué par un processeur q pendant une parallélisation. L'utilisation, notée $U_q(n,p)$, est définie selon le rapport $W_q(n,p)/T(n,p)$. Lorsque le travail est évalué par rapport au temps, la notation et la définition de l'utilisation deviennent: $U_q^t(n,p) = W_q^t(n,p)/T(n,p)$.

h) *Facteur d'équilibrage de travail*

Le facteur d'équilibrage de travail mesure la répartition de ce dernier sur l'ensemble des processeurs. Le facteur d'équilibrage, noté par $F(n,p)$, est défini selon le rapport $\frac{Min_{1\leq j\leq p} U_j(n,p)}{Max_{1\leq j\leq p} U_j(n,p)}$. Il est également possible de définir le facteur d'équilibrage en fonction du temps: $F^t(n,p) = \frac{Min_{1\leq j\leq p} U_j^t(n,p)}{Max_{1\leq j\leq p} U_j^t(n,p)}$.

1.3 Méthodes de SÉP parallèles

Cette section constitue la base même du sujet du présent mémoire et fait la synthèse des notions que nous avons abordées dans les deux sections précédentes. Nous présentons une classification des algorithmes de SÉP, basés sur le parallélisme de type 2, introduite par Gendron et Crainic (1994). Ces algorithmes sont conçus pour les architectures MIMD asynchrones. Toujours selon Gendron et Crainic (1994), nous exposons les aspects essentiels au développement d'implantations parallèles de SÉP. Enfin, nous présentons des mesures de performance spécifiques aux algorithmes parallèles de SÉP.

1.3.1 Classification des méthodes de SÉP parallèles

Afin de classifier les implantations sur des machines MIMD asynchrones, nous faisons la distinction entre algorithmes synchrones et asynchrones. Un deuxième paramètre de classification est basé sur la notion de *pool*, qui correspond à un espace-mémoire où sont conservées les unités de travail. Nous décrivons ci-dessous ces deux paramètres.

Dans les implantations *synchrones*, les processus responsables de l'exploration des sous-arborescences sont divisés en phases, pendant lesquelles les processus exécutent leur travail indépendamment des autres, alors que les communications peuvent survenir entre les phases. Les implantations *asynchrones* sont celles où les processus font leur travail indépendamment des autres et où les communications peuvent survenir à tout moment au cours de l'exécution.

Un pool correspond à un espace-mémoire permettant de conserver les unités de travail, qui sont, dans le cas de méthodes de SÉP, les sous-problèmes générés qui ne sont pas encore examinés. Typiquement, un processus sélectionne un sous-problème dans un pool, l'évalue et l'examine. Quand il a terminé, le processus insère dans un ou plusieurs pools les sous-problèmes non encore examinés. Un

processus peut travailler sans faire usage d'aucun pool. Par exemple, un sous-problème nouvellement généré par le processus peut être évalué et même examiné sans être inséré ou retiré d'un pool. La classification repose sur les approches à un seul pool ("single pool") et à plusieurs pools ("multiple pool"). Plusieurs organisations des pools sont possibles dans les approches à pools multiples. Les plus communes sont les approches *collégiales*, *groupées* et *mixtes*. Dans une organisation collégiale, chaque pool est associé à un seul processus. Dans une organisation groupée, les processus sont partitionnés et un pool est associé à chacune de ces partitions. Dans une organisation mixte, chaque processus possède son propre pool, mais a également accès à un pool global, partagé par tous les processus.

Enfin, en combinant ces deux paramètres, on peut former quatre classes d'algorithmes dédiés à une exécution en environnement MIMD asynchrone: SSP ("Synchronous Single Pool"), SMP ("Synchronous Multiple Pool"), ASP ("Asynchronous Single Pool") et AMP ("Asynchronous Multiple Pool").

Dans les algorithmes à pool simple, toute l'information sur l'exploration de l'arborescence de SÉP est concentrée en un seul endroit. On espère ainsi favoriser une sélection judicieuse des sous-problèmes en attente d'exploration. La classe SSP contient des algorithmes parallèles synchrones ne comportant qu'un seul pool. Essentiellement, une parallélisation de cette classe est très proche de l'algorithme séquentiel. Il y a un seul pool qui contient les sous-problèmes en attente d'exploration. Par contre, plutôt que de disposer d'une seule unité de traitement pour examiner ces sous-problèmes, il y en a plusieurs, disons p. On examine alors les sous-problèmes par groupes de p à la fois. La classe ASP contient des parallélisations qui sont presque identiques à celles de la classe SSP, sauf qu'ici, une unité de traitement a la possibilité d'accéder le pool global dès que son traitement est terminé. Notons cependant qu'il peut y avoir des délais si plusieurs unités de traitement tentent d'accéder au pool simultanément. Cette approche est avantageuse lorsque le temps de traitement de chacun des sous-problèmes est susceptible de varier beaucoup. Les implantations de ces classes d'algorithmes sur les architectures à échange de messages et à mémoire partagée

se distinguent principalement par la possibilité de développer pour ces dernières des structures de données à accès concurrents (Le Cun 1996).

L'idée principale des algorithmes à plusieurs pools est de diviser l'exploration de l'arborescence de solutions en l'exploration de plusieurs sous-arborescences. Sachant que les sous-arborescences n'ont pas toutes la même "probabilité" de mener à la solution optimale, certaines seront élaguées beaucoup plus rapidement que d'autres, créant ainsi des déséquilibres entre les pools locaux, d'où la nécessité du partage de sous-problèmes entre les pools. Comme dans les algorithmes à simple pool, on retrouve les classes d'algorithmes synchrones, SMP, et les classes d'algorithmes asynchrones, AMP. Le synchronisme et l'asynchronisme de ces classes d'algorithmes se situent essentiellement au niveau des communications inter-processeurs. De façon générale, dans les algorithmes SMP, on retrouve des phases de travail (examen de sous-problèmes) entrecoupées de phases de communications de données (bornes, sous-problèmes). Dans les algorithmes AMP, les échanges de données se font au cours de l'exploration. Avec les architectures à échange de messages, par exemple, les processus vérifient régulièrement la réception de messages et envoient des messages au cours de leur traitement respectif.

1.3.2 Conception des méthodes de SÉP parallèles

Dans cette section nous passons en revue quatre aspects essentiels à la conception de méthodes de SÉP parallèles.

1.3.2.1 Stratégie d'affectation initiale

Un de ces aspects, l'affectation et la génération initiale des sous-problèmes, survient au début de l'exécution de l'algorithme, lorsque seulement un sous-problème, la racine de l'arbre, est disponible à l'ensemble des processeurs. Puisqu'il arrive souvent que l'opération de séparation ne génère que peu de sous-

problèmes, une phase de démarrage où le parallélisme n'est pas totalement exploité, semble difficile à éviter. D'autre part, utiliser le parallélisme aussitôt que les sous-problèmes sont disponibles ne constitue pas nécessairement une bonne politique.

Considérons, par exemple, un algorithme séquentiel selon un parcours "profondeur d'abord" utilisant un schéma de séparation dichotomique. Il n'est pas rare de définir une opération de séparation de telle façon qu'un seul des deux sous-problèmes générés ait une bonne probabilité de mener à une solution optimale. La règle de sélection "profondeur d'abord" choisirait précisément ce sous-problème, pendant que l'évaluation et l'examen de l'autre sous-problème serait retardé jusqu'à ce que la sous-arborescence associée au sous-problème sélectionné soit complètement examinée. À ce stade, si une solution optimale a été trouvée, il y a une bonne probabilité que l'autre sous-problème soit immédiatement éliminé. Si la même opération de séparation est utilisée en environnement parallèle, et que les sous-problèmes sont examinés aussitôt qu'ils sont disponibles, les deux sous-problèmes peuvent être examinés simultanément. Alors, la sous-arborescence qui prend racine au sous-problème qui n'est pas sélectionné par la règle de sélection "profondeur d'abord" séquentielle peut maintenant, en environnement parallèle, contenir plus de sous-problèmes générés, représentant un travail supplémentaire par rapport à l'algorithme séquentiel. Ceci explique ce qu'on appelle l'*anomalie détrimentale* (Li et Wah 1984), où l'algorithme parallèle devient plus lent sur un exemplaire du problème que l'algorithme séquentiel. Le problème de l'affectation initiale peut alors être formulé ainsi: utiliser tous les processus aussitôt que possible, tout en évitant de leur attribuer des sous-problèmes non-prometteurs (qui ont une mince chance de mener à une solution optimale).

Plusieurs stratégies pour attaquer ce problème ont été proposées: (1) on affecte le problème original à un processus, et on diffuse graduellement aux processus les unités de travail aussitôt qu'elles sont créées; (2) on génère plusieurs sous-problèmes en appliquant une opération spéciale de séparation (le nombre de sous-problèmes ainsi créés pouvant être plus grand que le nombre de proces-

sus); (3) un processus exécute un algorithme séquentiel de SÉP jusqu'à ce qu'une quantité "suffisante" de sous-problèmes non-examinés soit disponible; (4) les processus exécutent une phase séquentielle dans laquelle la même arborescence est construite par chacun d'eux et, lorsque le nombre de problèmes non-examinés devient au moins égal au nombre de processus, chaque processus sélectionne les sous-problèmes qu'il examinera.

Les trois premières stratégies peuvent être utilisées dans tous les types d'algorithmes, alors que la dernière est particulièrement adaptée pour les algorithmes à pools multiples. Le choix d'une stratégie appropriée dépend de considérations pratiques, telles que le type d'architecture parallèle utilisé et les caractéristiques du problème à résoudre.

1.3.2.2 Stratégie d'affectation

Les objectifs des stratégies d'affectation sont d'*équilibrer le travail* sur l'ensemble des processus (tous les processus devraient avoir une part égale de travail de façon à exploiter le plus possible le parallélisme), et de leur attribuer des sous-problèmes prometteurs, afin d'éviter des situations où l'algorithme parallèle génère plus de sous-problèmes que l'algorithme séquentiel. Atteindre de tels objectifs est relativement facile dans des algorithmes à pool simple. Dans les algorithmes à pools multiples par contre, la situation est plus complexe. Une alternative consiste à créer dynamiquement de nouveaux processus à partir des pools très chargés (Schwan, Gawkowski et Blake 1988; Schwan, Blake, Bo et Gawkowski 1989a,b; Jansen et Sijstermans 1989). Une autre alternative plus commune, consiste à créer un nombre fixe de processus et de permettre l'échange de sous-problèmes entre les différents pools. Si cette dernière stratégie est utilisée, nous faisons la distinction entre des stratégies d'affectation *statique* et *dynamique* (voir section 1.2.2).

Dans une stratégie statique, un certain nombre de sous-problèmes sont

initialement créés, et les processus se partagent subséquemment ces derniers. Conséquemment, dans une telle approche, aucun mouvement de sous-problèmes ne s'effectue entre les pools locaux.

Dans les stratégies dynamiques, par contre, ces mouvements sont permis. Les façons de gérer ces échanges sont généralement sous la responsabilité des processus associés aux pools. On distingue, selon la politique de déclenchement utilisée, trois classes de stratégies dynamiques: les *stratégies avec requêtes*, les *stratégies sans requête* et les *stratégies combinées*.

Dans les stratégies avec requêtes, un processus se retrouvant avec un pool vide, réagit en faisant une requête de travail à un autre processus. La requête peut être acceptée, entraînant ainsi le transfert de sous-problèmes d'un pool à l'autre, ou peut être refusée, auquel cas le processus peut prendre la décision d'envoyer une autre requête. Si le processus receveur accepte la requête, il doit décider du choix et de la quantité de sous-problèmes qui seront transférés.

Dans les stratégies sans requête, les processus prennent la décision de partager leurs sous-problèmes sans être sollicités par d'autres processus. Avant d'envoyer leurs sous-problèmes, chaque processus doit savoir à quel moment il doit les transférer, à quel(s) pool(s) les envoyer, comment les sélectionner et la quantité à envoyer.

Dans les stratégies combinées, les processus échangent des sous-problèmes sans être sollicités, et effectuent aussi des requêtes de travail lorsque le nombre de sous-problèmes contenus dans leur pool est trop faible.

1.3.2.3 Gestion de la borne supérieure

En vue d'appliquer le test de la borne inférieure, un processus a besoin de connaître une borne supérieure sur la valeur optimale du problème original. Les stratégies de communication de la borne supérieure entre les processus dépendent

principalement du type d'algorithme et de l'architecture parallèle utilisée pour l'implanter. Par exemple, avec un algorithme asynchrone implanté en environnement à mémoire partagée, une variable accessible par tous les processus conserve la valeur de la meilleure borne supérieure trouvée au cours de l'exécution. Quand un processus trouve une meilleure borne supérieure, il la communique aux autres en mettant à jour la valeur de cette variable. Avec un algorithme asynchrone implanté en environnement à mémoire distribuée, chaque processus réserve dans sa mémoire locale une variable responsable de contenir la valeur de la meilleure borne supérieure. Quand un processus détermine une meilleure borne supérieure, il la communique à d'autres processus à l'aide d'un message contenant sa valeur. Habituellement, cette communication est une diffusion du message à tous les processus. L'efficacité d'une telle diffusion est clairement dépendante de la topologie du réseau d'interconnexion.

1.3.2.4 Détection de la fin

Le problème de la détection de la fin est trivialement résolu pour des algorithmes à pool simple et des algorithmes de la classe SMP. Le problème survient seulement pour des algorithmes de la classe AMP sur des architectures à mémoire distribuée, puisqu'il n'est pas suffisant que tous les pools soient vides pour déclarer la fin de l'exécution. En effet, des messages peuvent être encore véhiculés à travers le réseau d'interconnexion. Bien sûr, ce problème n'est pas spécifique aux algorithmes de SÉP parallèles, et les stratégies de détection de la fin d'algorithmes sur des systèmes distribués (sans contrôle central) ont été largement étudiés (consultez à titre d'exemple, l'article de Dijkstra et Sholten (1980), et les références données au chapitre 8 du livre de Bertsekas et Tsitsiklis 1989).

1.3.3 Mesures de performance des méthodes de SÉP parallèles

Dans le contexte des méthodes de SÉP, les parallélisations du type 2 sont basées sur l'exploration simultanée de sous-problèmes. En conséquence, le travail des processeurs peut être exprimé en terme du nombre de sous-problèmes examinés. De plus, nous avons vu que lors d'une exploration parallèle d'une arborescence de SÉP, il était possible d'examiner un nombre différent de sous-problèmes comparativement à la version séquentielle, en raison, notamment, de la stratégie d'affectation utilisée. Pour toutes ces raisons, il est possible de définir un certain nombre de mesures de performance supplémentaires permettant de décrire plus précisément les algorithmes de SÉP parallèles. Les mesures suivantes ont été décrites par Gendron et Crainic (1997).

a) *Facteur de pénalité de recherche*

Soit $N(n,p)$ le nombre de sous-problèmes explorés par une parallélisation, et soit $N_s(n)$ le nombre de sous-problèmes explorés par un algorithme séquentiel résolvant le même problème. Le facteur de pénalité de recherche, noté $P(n,p)$, est défini selon le rapport $N(n,p)/N_s(n)$.

b) *Écart de pénalité de recherche*

L'écart de pénalité de recherche tente de mesurer l'amplitude du facteur de pénalité de recherche. L'écart de pénalité de recherche, noté par $\Delta P(n,p)$, est défini de la façon suivante: $|P(n,p) - 1|$.

c) *Accélération ajustée*

L'accélération ajustée tente de mesurer, tout comme l'accélération, l'amélioration en temps obtenu du passage d'une exécution uni-processeur à une exécution p-processeur, mais cette fois, en ne tenant pas compte du surcoût de recherche. L'accélération ajustée, notée par $S_a(n,p)$, est définie selon le produit $S(n,p) \cdot P(n,p)$.

d) *Surcoût de parallélisation relatif ajusté*

Nous avons vu précédemment que le surcoût de parallélisation pouvait s'exprimer à partir de l'accélération. Un surcoût de parallélisation ajusté, noté par $B_a(n,p)$, peut donc être évalué à partir de l'accération ajustée de la façon suivante: $1/S_a(n,p) - 1/p$. Selon le même raisonnement que pour l'accélération ajustée, ce surcoût de parallélisation ajusté correspond au surcoût de parallélisation, mais en ne tenant pas compte du surcoût de recherche implicite à la version parallèle considérée.

CHAPITRE 2

Problématique et cadre méthodologique

Rappelons que l'objectif de notre travail est d'étudier la performance d'approches de parallélisation de méthodes de SÉP en environnement distribué, visant la résolution de problèmes difficiles de conception de réseaux. Un problème représentatif de cette classe, le problème de localisation multi-produits avec équilibrage, est choisi pour effectuer notre étude. Dans ce chapitre, nous présentons ce problème ainsi que son modèle, puis nous décrivons l'algorithme séquentiel de SÉP conçu par Gendron et Crainic (1995). Nous spécifions ensuite le cadre méthodologique qui a servi de base au développement de nos approches de parallélisation, en justifiant certains choix par notre connaissance *a priori* de l'algorithme séquentiel et de l'environnement parallèle utilisé. En terminant, nous faisons une revue de la littérature portant sur des stratégies de parallélisation adaptées à des problèmes difficiles possédant des caractéristiques similaires à ceux qui nous intéressent.

2.1 Le problème de localisation multi-produits avec équilibrage

Le problème de localisation multi-produits avec équilibrage, introduit pour la première fois par Crainic, Dejax et Delorme (1989), apparaît dans le contexte suivant. Une flotte de conteneurs hétérogènes, c'est-à-dire de différents types et de différentes tailles, sont utilisés pour desservir un ensemble d'entreprises en produits de toutes sortes. Après avoir déchargé les conteneurs, chacune des entreprises se retrouve avec un certain nombre de conteneurs vides. De plus, ces mêmes entreprises ont besoin de conteneurs pour acheminer à leur tour des items

qu'ils produisent. Sur le territoire où sont localisées ces entreprises, on retrouve un certain nombre de sites permettant d'entreposer les conteneurs vides déchargés ou de s'approvisionner en conteneurs vides selon leurs besoins. Ainsi, l'entreposage et l'approvisionnement de conteneurs vides aux différents sites provoquent des variations de disponibilité de ceux-ci sur le territoire. Toutefois, afin de permettre de rééquilibrer la situation et afin de satisfaire la demande des entreprises, on prévoit le déplacement de conteneurs vides entre les sites.

Sachant qu'un coût fixe est relié à l'ouverture d'un site (aussi appelé *dépôt*) et qu'un coût est aussi encouru par le déplacement d'un conteneur vide, d'un type et d'une taille particulière, entre une entreprise et un dépôt ou entre deux dépôts, on cherche alors à choisir un sous-ensemble de dépôts permettant de satisfaire les demandes des entreprises (aussi appelées *clients*) tout en minimisant les coûts.

Ce problème peut être modélisé de la façon suivante (Gendron et Crainic 1995): On considère un réseau orienté $G = (N, A)$, où N est l'ensemble des noeuds (clients et dépôts) et A est l'ensemble des arcs. Plusieurs produits (les types de conteneurs vides) se déplacent dans le réseau, représentés par l'ensemble P. L'ensemble des noeuds peut être partitionné en trois sous-ensembles: O, l'ensemble des noeuds origines (*clients offreurs*); D, l'ensemble des noeuds destinations (*clients demandeurs*); T, l'ensemble des noeuds de transfert (*dépôts*). Pour chaque $j \in T$, on définit $O(j) = \{i \in O : (i, j) \in A\}$ et $D(j) = \{i \in D : (j, i) \in A\}$, les ensembles de clients adjacents à ce dépôt, et on suppose qu'il existe au moins une origine ou une destination adjacente à chaque dépôt j ($O(j) \cup D(j) \neq \emptyset$). Pour chaque noeud $i \in N$, on définit les ensembles de dépôts adjacents à ce noeud dans les deux directions: $T^+(i) = \{j \in T : (i, j) \in A\}$, et $T^-(i) = \{j \in T : (j, i) \in A\}$. Puisqu'on suppose qu'il n'existe pas d'arcs entre les clients, l'ensemble des arcs peut être partitionné en trois sous-ensembles: les arcs client–dépôt, $A_{OT} = \{(i, j) \in A : i \in O, j \in T\}$; les arcs dépôt–client, $A_{TD} = \{(i, j) \in A : i \in T, j \in D\}$; et les arcs dépôt–dépôt, $A_{TT} = \{(i, j) \in A : i \in T, j \in T\}$.

Le problème consiste à minimiser les coûts encourus par le mouvement des flots à travers le réseau en vue de satisfaire les offres aux origines et les demandes aux destinations. Pour chacun des clients offreurs $i \in O$, l'offre du produit p est notée par o_i^p, alors que pour chacun des clients demandeurs $i \in D$, la demande pour le produit p est notée par d_i^p. Toutes les offres et les demandes sont supposées non négatives et connues à l'avance. Un coût non négatif c_{ij}^p est encouru pour chaque unité de flot du produit p traversant l'arc (i, j). De plus, pour chaque dépôt $j \in T$, un coût fixe non négatif f_j est encouru si le dépôt est ouvert.

Une variable y_j est définie comme prenant la valeur 1 si le dépôt j est ouvert, ou la valeur 0 autrement. Une variable x_{ij}^p représente la quantité de flot du produit p circulant sur l'arc (i, j). Le modèle est alors le suivant:

$$Z = \min \sum_{j \in T} f_j y_j + \sum_{p \in P} \left\{ \sum_{(i,j) \in A_{OT}} c_{ij}^p x_{ij}^p + \sum_{(j,i) \in A_{TD}} c_{ji}^p x_{ji}^p + \sum_{(j,k) \in A_{TT}} c_{jk}^p x_{jk}^p \right\} \quad (2.1)$$

$$\sum_{j \in T^+(i)} x_{ij}^p = o_i^p \qquad \forall\, i \in O,\, p \in P \quad (2.2)$$

$$\sum_{j \in T^-(i)} x_{ji}^p = d_i^p \qquad \forall\, i \in D,\, p \in P \quad (2.3)$$

$$\sum_{i \in D(j)} x_{ji}^p + \sum_{k \in T^+(j)} x_{jk}^p - \sum_{i \in O(j)} x_{ij}^p - \sum_{k \in T^-(j)} x_{kj}^p = 0 \qquad \forall\, j \in T,\, p \in P \quad (2.4)$$

$$x_{ij}^p \leq o_i^p y_j \qquad \forall\, j \in T,\, i \in O(j),\, p \in P \quad (2.5)$$

$$x_{ji}^p \leq d_i^p y_j \qquad \forall\, j \in T,\, i \in D(j),\, p \in P \quad (2.6)$$

$$x_{ij}^p \geq 0 \qquad \forall\, (i, j) \in A,\, p \in P \quad (2.7)$$

$$y_j \in \{0, 1\} \qquad \forall\, j \in T \quad (2.8)$$

Les contraintes (2.2) et (2.3) assurent que les offres et les demandes des clients du réseau sont satisfaites. Les contraintes (2.4), appelées contraintes de conservation

de flot, assurent que le flot entrant et sortant de chacun des dépôts est conservé. Les contraintes (2.5) et (2.6) assurent qu'aucun flot ne puisse passer par un dépôt fermé.

Ce modèle, bien que basé sur la structure conventionnelle d'un problème de localisation sans capacité (Krarup et Pruzan 1983; Cornuéjols, Nemhauser et Wolsey 1990) en diffère par les points suivants:

1. présence de flot inter-dépôts;
2. présence de noeuds clients offreurs et de noeuds clients demandeurs;
3. plusieurs produits peuvent circuler dans le réseau (ce dernier point n'a pas d'impact sur la complexité du modèle mais en accroît grandement les dimensions).

Un algorithme de SÉP séquentiel efficace pour résoudre ce problème a été mis au point. Cet algorithme tire son efficacité des méthodes de calcul de bornes, qui exploitent la structure de réseau sous-jacente au modèle (Crainic et Delorme 1993) en dérivant deux relaxations de celui-ci. La première relaxation est obtenue en retirant du modèle les contraintes qui interdisent la circulation de flot par les dépôts fermés (2.5 et 2.6). Le sous-problème obtenu, possède la structure d'un problème de *flot à coût minimum multiproduits sans capacité.* Une borne supérieure sur la valeur optimale du problème original est alors obtenue à partir d'une solution optimale de ce sous-problème en posant y_j égal à 1 s'il y a du flot passant par le dépôt j, et à 0 sinon. La deuxième relaxation est obtenue en retirant du modèle les contraintes de flot (2.4). Le sous-problème obtenu possède la structure d'un problème de *localisation sans capacité* pour lequel il existe un algorithme de résolution efficace appelé DUALOC (Erlenkotter 1978). Dans ce cas, une borne supérieure sur la valeur optimale du problème original est obtenue en résolvant un problème de flots à coût minimum.

Pour chaque sous-problème généré, on applique une procédure basée sur ces relaxations pour obtenir des bornes inférieures et supérieures. Ces bornes

sont utilisées pour le développement de règles d'élagage et de prétraitement efficaces. Afin de représenter les variables de localisation qui sont fixées au cours des opérations de séparation et des règles de prétraitement, on définit les ensembles $T_{01} = \{j \in T : Dom(y_j) = \{0,1\}\}$, $T_0 = \{j \in T : Dom(y_j) = \{0\}\}$, et $T_1 = \{j \in T : Dom(y_j) = \{1\}\}$ (où $Dom(y_j)$ représente l'ensemble des valeurs que peut prendre la variable y_j), dénotant respectivement les ensembles de dépôts *libres*, *fermés* et *ouverts*.

Pour générer les sous-problèmes à partir d'un sous-problème S, nous utilisons une opération de séparation *dichotomique*: un dépôt $j^* \in T_{01}$ est choisi selon un certain critère (Gendron et Crainic 1995), et $S_0^{j^*}$ est obtenu en transférant j^* à T_0, alors que $S_1^{j^*}$ est obtenu d'un transfert de j^* à T_1. En accord avec la terminologie relative aux arborescences, $S_0^{j^*}$ et $S_1^{j^*}$ sont les *noeud-fils-0* et *noeud-fils-1* respectivement du *noeud père* S, et le problème original, où tous les dépôts sont libres, correspond au *noeud racine*. Afin de décider quel sous-problème sera examiné en priorité, nous utilisons une stratégie de sélection "profondeur d'abord": choisir un des sous-problèmes générés le plus récemment.

Formellement, l'algorithme de SÉP conserve une pile Λ de sous-problèmes générés, ainsi que la valeur $\overline{Z^u}$ de la meilleure solution identifiée à jour, et procède comme suit:

1. (*initialisation*) S est le problème original: $T_{01} \leftarrow T$, $T_0 \leftarrow \emptyset$, $T_1 \leftarrow \emptyset$. $\Lambda \leftarrow \emptyset$. $\overline{Z^u} \leftarrow +\infty$.
2. (*prétraitement*) Tenter de fixer certaines variables (T_{01}, T_0 and T_1 peuvent êtres modifiés).
3. (*opération d'évaluation*) Effectuer la procédure d'évaluation sur S ($\overline{Z^u}$ peut être mis à jour); si S peut être élagué, aller à l'étape 5.
4. (*opération de séparation*) Choisir $j^* \in T_{01}$ et générer $S_0^{j^*}$ and $S_1^{j^*}$; sélectionner l'un d'eux pour l'examen du prochain sous-problème, qui devient le sous-problème S, et ajouter l'autre sous-problème à Λ. Aller à l'étape 2.

5. (*test d'arrêt*) Si $\Lambda = \emptyset$, STOPPER; $\overline{Z^u}$ est la valeur optimale du problème original.

6. (*retour arrière*) Sélectionner le sous-problème S au-dessus de la pile Λ. S'il peut être élagué, aller à l'étape 5; sinon, aller à l'étape 2.

La performance de cet algorithme est principalement influencée par trois facteurs: la qualité des bornes, la capacité d'éviter des calculs non nécessaires par l'utilisation des règles d'élagage et de prétraitement, et enfin, la façon dont les noeuds sont générés et sélectionnés. Pour plus de détails, consulter Gendron et Crainic (1995).

2.2 Cadre méthodologique

Dans cette section, nous cernons le cadre méthodologique qui a soutenu le développement de nos approches de parallélisation. Notre objectif est double. D'une part, nous précisons ce qui a servi de base commune au développement des trois approches de parallélisation, en justifiant certains choix par notre connaissance *a priori* de l'algorithme séquentiel et de l'environnement parallèle utilisé (les distinctions détaillées entre les trois approches seront explicitées au chapitre 3). D'autre part, nous identifions les problèmes de recherche auxquels nous entendons contribuer.

Nos approches de parallélisation sont adaptées à une architecture MIMD asynchrone à échange de messages: plus précisément, nous utilisons un **réseau distribué de stations de travail**. Une croissance technologique rapide fait en sorte que l'utilisation de ce type d'environnement est de plus en plus répandu. De plus, une grande variété d'outils logiciels existent déjà et permettent notamment de gérer efficacement les communications entre les stations (nous reviendrons sur ce point aux sections 4.1 et 4.2).

Nous développons des méthodes parallèles **asynchrones**. En effet, plusieurs arguments militent en faveur des parallélisations asynchrones. D'une part,

en vue d'exploiter pleinement le parallélisme, il est essentiel de minimiser les temps d'inactivité des processeurs. Une parallélisation asynchrone, dans laquelle on ne retrouve pas de phases de synchronisation prédéterminées, permet d'affecter des sous-problèmes à des processeurs inactifs dès qu'ils sont générés, sans attendre qu'une éventuelle phase de communication survienne. Cet aspect est particulièrement important lorsque, pour une application donnée, le travail requis pour l'examen d'un sous-problème varie beaucoup d'un sous-problème à l'autre. D'autre part, il est impératif de diffuser le plus rapidement possible l'information permettant d'éviter le travail inutile. Dans le contexte d'un algorithme de SÉP, la diffusion des meilleures bornes supérieures évaluées est essentielle à l'atteinte de cet objectif. Encore une fois, cet aspect est aussi favorisé par l'absence de phases de communication prédéterminées.

Nous avons vu à la section 1.2.2 du chapitre précédent que plusieurs méthodes de décomposition des algorithmes de SÉP sont possibles. Dans notre cas, nous faisons appel à la **décomposition de type 2** où l'exploration d'une seule arborescence est réalisée par un examen concurrent des sous-problèmes. Contrairement à une décomposition de type 1, qui ne fait généralement qu'accélérer les calculs à chaque noeud de l'arborescence, la décomposition de type 2 modifie l'ordre séquentiel d'exploration des sous-problèmes. Ainsi, elle pose des problèmes intéressants: avec une telle approche, peut-on générer beaucoup moins de noeuds que l'algorithme séquentiel, ou au contraire beaucoup plus, et dans l'un ou l'autre cas, sous quelles conditions? Ces problèmes ont fait l'objet d'une littérature abondante qui s'est penchée sur l'étude de telles anomalies (voir section 1.3.2). L'objectif de toute bonne implantation d'une méthode de type 2 est ainsi de diminuer les risques de générer des arborescences plus grandes que celles obtenues par l'algorithme séquentiel.

Notons que pour le problème de localisation multi-produits avec équilibrage, une parallélisation de type 1 peut être envisagée, qui décomposerait l'évaluation des bornes par produit (Gendron et Crainic 1993). Cependant, l'efficacité d'une telle parallélisation serait limitée car, en pratique, le nombre de produits est

généralement petit (la plus grosse application traitée à ce jour ne possède que douze produits).

Notons également qu'une parallélisation de type 3 n'est intéressante que pour des problèmes pour lesquels les solutions optimales sont identifiées après de longues explorations (Gendron et Crainic 1994). Dans ce contexte, il devient intéressant de faire l'exploration de plusieurs arborescences simultanément dans l'espoir d'accélérer l'obtention d'une solution optimale. Pour la plupart des exemplaires du problème de localisation multi-produits avec équilibrage traités à ce jour, les solutions optimales sont cependant identifiées après une courte exploration (même si l'exploration complète de l'arborescence, nécessaire pour démontrer que cette solution est optimale, est parfois très longue). Par conséquent, il nous est apparu inutile d'étudier les parallélisations de type 3.

Ainsi, basées sur une décomposition de type 2, nous proposons trois approches de parallélisation qui se distinguent par la répartition des pools (ce que nous appelons aussi l'organisation de la mémoire) et le contrôle de l'exploration:

Une approche centralisée : À un extrême, tous les sous-problèmes générés au cours de la résolution sont conservés dans un seul pool, associé à la mémoire d'un des processeurs qui participent à la résolution. Toutes les sélections et les insertions de sous-problèmes se font par ce pool global. Implicitement, le contrôle de l'exploration est donc centralisé.

Une approche décentralisée : À un autre extrême, les sous-problèmes générés au cours de l'exploration de l'arborescence sont répartis sur plusieurs pools, chacun des processeurs ayant accès à son propre pool. Des sélections et des insertions de sous-problèmes sont réalisées dans chacun de ces pools. Implicitement, le contrôle de la recherche est complètement décentralisé.

Une approche hybride : En combinant les deux approches précédentes, on obtient une approche hybride dans laquelle les sous-problèmes générés au cours de la résolution sont répartis dans les pools locaux à chacun des processeurs, mais également, un certain nombre de ces sous-problèmes sont

choisis et stockés dans un pool global accessible par tous les processeurs. On dira alors que le contrôle de la recherche est également hybride.

En accord avec la classification des méthodes parallèles de SÉP vue au chapitre précédent, nous développons donc trois méthodes dont une appartient à la classe **ASP** (approche centralisée), alors que les deux autres appartiennent à la classe **AMP** (approches décentralisée et hybride). Nous avons vu que pour nos trois approches, l'organisation de la mémoire détermine le type de contrôle de l'exploration.

Nos parallélisations sont composées d'un **nombre fixe de processus** séquentiels communicants, créés de manière statique au début de l'exécution. Ces processus peuvent être actifs ou inactifs pendant la résolution. Typiquement, un processus est responsable d'une ou plusieurs fonctions telles que la gestion d'un pool, l'examen de sous-problèmes, l'envoi et la réception de messages, etc. Une autre possibilité aurait été d'avoir un nombre arbitraire de processus créés dynamiquement. Ainsi, chaque processus pourrait correspondre à l'examen d'un sous-problème généré suite aux opérations de séparation. Cependant, dans notre cas, étant donné la forte granularité de calcul des bornes à chacun des sous-problèmes, la première approche s'avère efficace au niveau de l'utilisation des processeurs (nous y reviendrons au chapitre 5), tout en étant plus simple à implanter.

Ainsi, dans chacune des trois approches, il y a un processus **coordonnateur** et un nombre fixe de processus **travailleurs**. Le processus coordonnateur assure un contrôle sur l'exploration de l'arborescence (dans les approches centralisée et hybride), facilite l'équilibrage de travail et coordonne les échanges d'informations entre les travailleurs. Les tâches d'évaluation et de séparation des sous-problèmes sont sous la responsabilité des travailleurs. Typiquement, on retrouve un travailleur par processeur, alors que le coordonnateur s'exécute sur le même processeur qu'un des travailleurs (sinon, il y aurait sous-utilisation des ressources).

Le développement des trois approches de parallélisation se fait en accord

avec les aspects vus à la section 1.3.2.:

Stratégie d'affectation initiale: Une telle stratégie vise à diminuer la possibilité de créer des pénalités de recherche, de même qu'à exploiter pleinement le parallélisme au début de la résolution (voir Gendron et Crainic 1997). Cet objectif peut être atteint en déterminant une solution réalisable de qualité avant de démarrer le traitement parallèle. Dans nos trois approches, nous utilisons une stratégie d'affectation initiale simple qui permet de déterminer une solution réalisable de qualité: la racine est affectée à un seul processus travailleur qui explore l'arborescence jusqu'à ce qu'une feuille soit atteinte.

Stratégie d'affectation: Rappelons que les objectifs de la stratégie d'affectation sont de bien répartir le travail sur les processeurs et, dans le cas d'une méthode de SÉP, d'affecter des sous-problèmes prometteurs. Dans nos trois implantations, la stratégie d'affectation est **dynamique**, et ce pour deux raisons. Premièrement, il est souhaitable de permettre des mouvements de sous-problèmes entre les pools pendant le processus de résolution. En effet, il est généralement difficile de diviser le travail avant de commencer la résolution d'un problème, car on ne sait pas, *a priori*, quelle sera l'allure de l'arborescence générée. Deuxièmement, il est préférable de préserver la possibilité, en cours de résolution, de faire examiner en priorité des sous-problèmes qui apparaissent plus "prometteurs". Pour chacune des trois approches, nous proposons au chapitre 3 plusieurs stratégies d'affectation, qui diffèrent selon les politiques (déclenchement, sélection, localisation et information: voir section 1.2.2) adoptées.

Gestion de la borne supérieure: En raison de la forte granularité de calcul de notre classe de problèmes, il est crucial de diffuser le plus rapidement possible les mises à jour de la meilleure borne supérieure à l'ensemble des processus participant à la résolution. D'une part, un délai important dans la mise à jour de la borne supérieure pourrait avoir comme conséquence d'entraîner l'examen de sous-problèmes qui seraient normalement élagués si la nouvelle borne supérieure était connue. Ceci résulterait en une pénalité de recherche. D'autre part, cette

borne supérieure peut également contribuer à éviter la communication de sous-problèmes qui seraient élagués. Les processus travailleurs, responsables de l'examen des sous-problèmes, se verront assigner la tâche de diffuser aux autres processus les meilleures bornes supérieures qu'ils découvrent.

Détection de la fin: La stratégie de détection de la fin varie selon la stratégie de parallélisation. Chaque méthode possède un critère qui permet de déclarer la fin de la résolution (voir chapitre 3). À cette fin, dans les trois parallélisations, le coordonnateur conserve une information suffisante sur l'état d'activité des travailleurs.

Notons qu'il est important de tester les trois approches, puisqu'il est impossible de savoir *a priori* quelle organisation de la mémoire permet d'explorer une arborescence de SÉP de la manière la plus efficace. Le passage d'une organisation de la mémoire à une autre a un impact sur l'ordre d'exploration des sous-problèmes. À un extrême, centraliser les sous-problèmes dans un seul pool permet de les sélectionner à partir d'une connaissance globale de l'exploration de l'arborescence. À l'autre extrême, décentraliser complètement l'information, par l'utilisation de plusieurs pools, implique des sélections de sous-problèmes à partir d'une connaissance partielle de l'état de l'exploration et, particulièrement dans ce cas, l'ordre d'exploration des sous-problèmes est grandement perturbé par rapport à une exploration séquentielle.

Suite à nos expérimentations sur plusieurs exemplaires du problème de localisation multi-produits avec équilibrage, nous entendons apporter des éléments de réponse aux questions suivantes:

- Pour chacune des approches de parallélisation, quelles stratégies d'affectation sont les plus prometteuses?
- Quelles sont les performances relatives des trois approches de parallélisation?

2.3 Revue des travaux antérieurs

Plusieurs chercheurs ont réalisé des implantations parallèles d'algorithmes de SÉP pour la résolution de problèmes d'optimisation combinatoire particuliers tels que le problème d'affectation quadratique, le problème de recouvrement de sommets, le problème du voyageur de commerce (pour une revue détaillée de ces efforts, consulter Gendron et Crainic 1994). Pour plusieurs de ces problèmes, les méthodes d'évaluation de bornes nécessitent beaucoup moins de calculs que celles utilisées pour les problèmes de conception de réseaux. Dans cette section, nous passons en revue les travaux portant sur des problèmes ayant des caractéristiques semblables: le problème de localisation multi-produits avec équilibrage, mais aussi le problème général de programmation linéaire mixte, pour lequel plusieurs cas particuliers génèrent des méthodes de calcul de bornes à granularité forte.

Gendron et Crainic (1993) (voir aussi Gendron 1991) ont choisi une approche appartenant à la classe SSP pour implanter une version parallèle d'un algorithme séquentiel approximatif permettant de résoudre le problème de localisation multi-produits avec équilibrage. L'algorithme s'arrête lorsqu'un nombre fixe de sous-problèmes ont été examinés. Dans ce contexte, il est crucial d'éviter des situations où l'algorithme parallèle détermine une solution moins bonne que celle obtenue avec l'algorithme séquentiel. Afin d'atteindre cet objectif, les auteurs divisent l'algorithme parallèle en deux phases: une phase séquentielle, où le coordonnateur exécute l'algorithme séquentiel jusqu'à ce que N itérations soient complétées, ainsi qu'une phase parallèle. Dans cette dernière phase, un processus maître sélectionne les sous-problèmes selon leur profondeur, les envoient à des processus travailleurs, qui effectuent les opérations d'évaluation et de séparation sur un sous-probléme. Le maître reçoit les évaluations et génère de nouveaux noeuds, s'il y a lieu. Cette parallélisation est justifiée par le fait que, pour la plupart des jeux de données, le temps nécessaire pour l'opération d'évaluation d'un sous-problème est largement supérieur à celui requis par d'autres opérations, ainsi que par les communications. Des essais sur un réseau de 16 transputers ont été réalisés.

Abdelrahman et Mudge (1988) (voir aussi Abdelrahman 1988) ont développé une méthode parallèle de SÉP appartenant à la classe ASP pour résoudre des problèmes linéaires en variables 0-1. Ces méthodes ont été implantées sur un type particulier d'architecture parallèle où les processeurs sont reliés entre eux selon une topologie en hypercube (NCUBE/6 à 64 processeurs). Le maître gère un pool et sélectionne, selon la règle "meilleur d'abord", un sous-problème qu'il envoie au premier esclave inactif. Après réception d'un sous-problème, chaque esclave applique la règle de séparation et évalue les sous-problèmes générés, après quoi il les envoie au maître. Les auteurs observent que l'accélération est limitée par deux facteurs principaux: la surcoût de communication, qui augmente avec la taille de l'hypercube, ainsi qu'un mauvais équilibrage de travail qui s'explique par le fait que les processeurs étant situés plus près du maître reçoivent plus de travail que les autres.

Eckstein (1994) propose une méthode parallèle de la classe ASP permettant de résoudre des problèmes généraux de programmation linéaire mixte. Sa méthode a été implantée sur une Connection Machine 5 (CM-5), un environnement parallèle asynchrone à échange de messages. Le pool global, géré par le maître, contient une partie de l'information associée à chacun des sous-problèmes générés par les esclaves (en l'occurrence la priorité de chacun des sous-problèmes, ainsi que l'adresse mémoire et l'adresse du processeur où se trouve l'information complète). Dans cette méthode, des mouvements fréquents de sous-problèmes ont lieu entre les esclaves. L'originalité de cette parallélisation, réside dans le fait que bien que l'information soit répartie, le contrôle de la recherche est toutefois centralisé. Des tests ont été réalisés sur plusieurs jeux de données provenant d'applications réelles en utilisant jusqu'à 128 processeurs.

Gendron et Crainic (1997) ont proposé une méthode parallèle appartenant à la classe AMP pour résoudre le problème de localisation multi-produits avec équilibrage. La méthode possède une organisation collégiale et utilise un processus coordonnateur permettant de faciliter les communications inter-processeurs, de même que l'équilibrage de travail. De plus, la résolution est divisée en deux

phases: une phase responsable de la stratégie d'affectation initiale, et une phase principale qui effectue l'exploration parallèle. Plusieurs versions de la première phase sont comparées. La phase principale est ensuite démarrée. Chaque processus travailleur effectue une exploration sur une sous-arborescence selon une approche "profondeur d'abord". Durant cette phase, quand un travailleur se retrouve sans travail, il envoie une requête au coordonnateur, qui à son tour, tente de déterminer un travailleur donneur qui pourrait partager une partie de sa charge de travail. Pour ce faire, le coordonnateur reçoit périodiquement des travailleurs la charge de travail contenue dans leurs pools. Des essais numériques ont été réalisés sur un réseau distribué de 17 stations de travail.

Eckstein (1997) propose des implantations parallèles distribuées d'un algorithme de SÉP pour la résolution du problème général de programmation linéaire mixte. Des essais sont réalisées sur la CM-5. La problématique consiste à déterminer si les fonctions d'ordonnancement de tâches et de stockage des données doivent être prises en charge par un seul processeur ou encore par plusieurs. Les essais réalisés permettent de comparer des méthodes d'ordonnancement de tâches centralisée, groupée, et totalement décentralisée utilisant une stratégie d'équilibrage de travail basée sur une combinaison d'affectation aléatoire et par rendez-vous. De plus, plusieurs schémas de stockage des "pseudo-coûts", données permettant de choisir les variables de branchement, sont analysés.

À notre connaissance, ce mémoire constitue une première étude comparative de trois organisations de la mémoire pour des algorithmes parallèles de SÉP en environnement distribué, appliqués à la résolution de problèmes à granularité forte. Nous proposons de plus un cadre unificateur pour l'implantation des trois approches en utilisant un coordonnateur et des travailleurs. Ce genre d'organisation va de soi pour les implantations d'approches centralisées, mais ce n'est pas le cas pour les autres types d'approches. Dans notre approche décentralisée, nous proposons une stratégie d'affectation dynamique flexible: l'utilisation de plusieurs paramètres permet d'adapter la stratégie à l'environnement parallèle et aux caractéristiques de l'application considérée (voir les détails à la section

3.3.2). Cette stratégie est semblable à celle introduite par Gendron et Crainic (1997), mais nous y avons ajouté de nouveaux paramètres, ainsi que de nouvelles stratégies de sélection de processus donneurs de travail. Mentionnons également que notre approche hybride constitue une première implantation avec une organisation mixte des pools pour résoudre le problème de localisation multi-produits avec équilibrage.

CHAPITRE 3

Approches de parallélisation

Dans ce chapitre, nous exposons trois approches de parallélisation des méthodes de SÉP conçues pour des environnements MIMD asynchrones à échanges de messages. Rappelons que ces parallélisations sont basées sur l'organisation de la mémoire: une approche centralisée, où tous les sous-problèmes conservés sont stockés dans un seul pool accessible par tous les processeurs; une approche décentralisée, où chaque processeur possède son propre pool local; une approche hybride, où un pool global est accessible par tous les processeurs et où chacun dispose également d'un pool local.

L'objectif principal de ce chapitre est de présenter les principes qui gouvernent le développement de chacune de ces trois approches. La mise en oeuvre de ces principes, appliqués à la résolution du problème de localisation multi-produits avec équilibrage, fera l'objet du prochain chapitre.

Rappelons qu'au chapitre précédent, nous avons décrit une stratégie d'affectation initiale commune aux trois approches: un seul travailleur explore l'arborescence jusqu'à ce qu'une feuille soit atteinte. De même, nous avons présenté un mécanisme de gestion de la borne supérieure, similaire dans les trois approches, qui consiste à diffuser les meilleures bornes supérieures dès qu'elles sont identifiées. Toutefois, nous avons vu que les stratégies d'affectation et les mécanismes de détection de la fin différaient d'une approche à l'autre. Nous consacrons les trois dernières sections de ce chapitre à une description détaillée des stratégies d'affectation (en particulier, les politiques d'information, de déclenchement, de sélection et de localisation) et des mécanismes de détection de la fin pour chacune

des approches de parallélisation. Dans la section suivante, nous précisons d'abord les hypothèses et le cadre conceptuel qui ont servi de base au développement des stratégies d'affectation.

3.1 Cadre conceptuel

Au chapitre précédent, nous avons mentionné qu'une stratégie d'affectation dynamique est mieux adaptée aux méthodes de SÉP qu'une stratégie statique. À la section 1.2.2, nous avons vu que la conception d'une stratégie d'affectation efficace dépend non seulement de facteurs reliés aux caractéristiques de l'application, mais aussi à celles de l'environnement parallèle.

Dans notre cas, il s'agit d'un réseau distribué de stations de travail (une architecture MIMD à échange de messages). Nos expérimentations sont réalisées avec un réseau dédié: aucune autre application que la nôtre (outre celles reliées au système d'exploitation) n'occupe les ressources du réseau. De plus, le réseau est constitué de machines identiques (pour plus de détails voir section 5.2). Ce contexte justifie les hypothèses suivantes, qui servent de base au développement conceptuel des stratégies d'affectation:

- le coordonnateur possède un lien de communication direct avec tous les travailleurs;
- tous les travailleurs sont interconnectés directement entre eux;
- chaque lien de communication est bidirectionnel;
- les délais de communication entre chaque paire de processus sont les mêmes (cette hypothèse n'est pas pleinement réaliste dans notre contexte, mais cela n'invalide en rien la conception des stratégies d'affectation);
- chaque processus a une puissance de traitement équivalente.

3.2 Approche centralisée

L'approche centralisée utilise une organisation de la mémoire à un seul pool. Ce pool est géré par le coordonnateur. Le coordonnateur garde un contrôle total sur l'exploration de l'arborescence: il sélectionne et affecte les sous-problèmes du pool aux travailleurs, ces derniers étant responsables de l'examen des sous-problèmes, puis il récupère les résultats d'examen et détecte la fin de l'exploration. Concrètement, le coordonnateur exécute l'algorithme de SÉP en déléguant l'examen des sous-problèmes aux travailleurs. Cette approche de parallélisation est mieux connue sous l'appellation *maître/esclave*. La figure 1 illustre schématiquement l'approche centralisée.

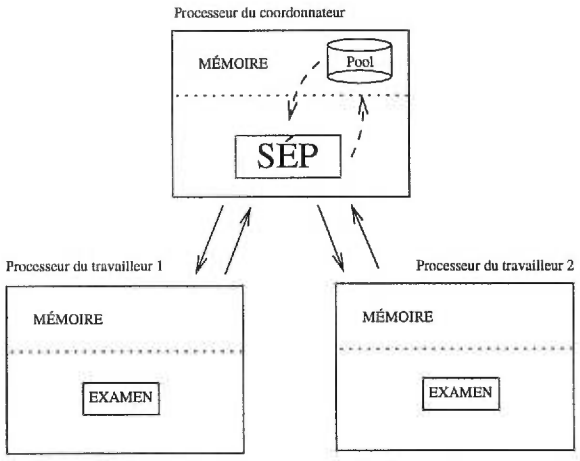

FIGURE 1: Schéma de l'approche centralisée

3.2.1 Stratégie d'affectation

La stratégie d'affectation est dynamique et sans requête. Tant que le pool est non vide, le coordonnateur sélectionne un sous-problème et localise un travailleur, parmi ceux qui sont inactifs, pour le lui affecter. Une fois l'examen du sous-problème terminé, le travailleur envoie les résultats au coordonnateur: les sous-problèmes générés ou un message d'élagage. La réception des résultats indique au coordonnateur que le travailleur est de nouveau disponible pour examiner un sous-problème.

Politique d'information

La totalité de l'information nécessaire à la stratégie d'affectation est gérée par le coordonnateur. C'est à partir de l'information associée aux sous-problèmes contenus dans le pool (Ω) qu'il sélectionne les sous-problèmes à affecter (politique de sélection). Étant responsable des affectations effectuées en cours d'exécution, le coordonnateur connaît l'état d'activité des travailleurs (A): soit J l'ensemble des travailleurs, un travailleur $j \in J$ est soit en cours d'examen de sous-problème ($A_j = ACTIF$), soit en attente de sous-problème ($A_j = INACTIF$). Ainsi, $J = J^+ \cup J^-$ où $J^+ = \{j | A_j = ACTIF\}$ et $J^- = \{j | A_j = INACTIF\}$.

Politique de déclenchement

Puisque les travailleurs ne possèdent pas de pool local, ils peuvent recevoir au plus un sous-problème à la fois et doivent également retourner au coordonnateur les sous-problèmes générés par l'opération de séparation. Par conséquent, le coordonnateur peut affecter des sous-problèmes lorsque le pool est non-vide et qu'au moins un travailleur est disponible. Le critère de déclenchement de la stratégie d'affectation est formulé ainsi:

$$(\Omega \neq \emptyset \ \wedge \ J^- \neq \emptyset).$$

Politique de sélection

De façon analogue à l'algorithme séquentiel, une règle de sélection doit être définie de façon à diriger la construction de l'arborescence. Cette règle de sélection, appliquée par le coordonnateur, détermine, d'une part, l'ordre des opérations exécutées par un travailleur et fournit, d'autre part, un critère de choix de sous-problème parmi ceux déjà conservés dans le pool. Jusqu'ici, nous avons vu deux règles de sélection: la règle "meilleur d'abord" et la règle "profondeur d'abord" (voir section 1.1).

Formellement, il est possible de représenter ces deux règles de sélection de la façon suivante: notons $Q_1, Q_2, \ldots, Q_m$ les sous-problèmes contenus dans un pool

Ω. On cherche à déterminer un sous-problème $Q_s \in \Omega$ tel que:

$$s = \arg \min_{1 \leq i \leq m} (Z^l(Q_i)) \text{ (règle meilleur d'abord)}$$

où $Z^l(Q_i)$ est la borne inférieure du sous-problème Q_i. De la même façon, on a:

$$s = \arg \min_{1 \leq i \leq m} (-D(Q_i)) \text{ (règle profondeur d'abord)}$$

où $D(Q_i)$ est la profondeur du sous-problème Q_i. Au chapitre 1, nous avons vu que la règle meilleur d'abord permet de minimiser le nombre de sous-problèmes décomposés lorsque, notamment, le pool ne contient pas deux sous-problèmes ayant la même borne inférieure. Nous avons vu également que la règle profondeur d'abord permet de trouver des solutions réalisables plus rapidement et facilite la réoptimisation.

Ces mêmes règles peuvent aussi être utilisées par le coordonnateur dans l'approche centralisée. Si le coordonnateur utilise la règle de sélection meilleur d'abord, il choisira parmi les sous-problèmes du pool, celui qui possède la borne inférieure la plus petite. Si le coordonnateur utilise par contre la règle de sélection profondeur d'abord, il choisira le sous-problème ayant la profondeur la plus élevée.

Cependant, avec une parallélisation de type 2, l'ordre de génération des sous-problèmes est passablement perturbé par rapport à l'ordre de génération de l'algorithme séquentiel. En particulier, les règles classiques présentées précédemment n'ont pas la capacité de discriminer deux sous-problèmes ayant des bornes inférieures identiques ou la même profondeur. Un mauvais choix peut alors entraîner un surcoût de travail, c'est-à-dire un plus grand nombre de sous-problèmes examinés. C'est pourquoi la sélection constitue un rôle très important du coordonnateur. De façon à pouvoir discriminer, il semble donc plus approprié de sélectionner les sous-problèmes en utilisant plus d'un critère à la fois. Par exemple, entre deux sous-problèmes ayant la même borne inférieure, on favorisera celui dont la profondeur est la plus élevée, permettant ainsi de déterminer rapidement une solution réalisable. Inversement, entre deux sous-problèmes ayant la même

profondeur, on favorisera celui dont la borne inférieure est la moins élevée. Dans ce cas, le choix de Q_s est tel que:

$$s = \arg \min_{1 \leq i \leq m} (p_1 \, Z^l(Q_i) + p_2 \, (-D(Q_i)))$$

où p_1 et p_2 sont ajustés de façon à pondérer l'importance de la borne inférieure et de la profondeur. Lorsque l'ordre des opérations de l'algorithme de SÉP séquentiel parallélisé est défini de façon à faire la gestion de sous-problèmes non évalués (comme dans l'ordre des opérations défini par la règle profondeur d'abord), la valeur $Z^l(Q)$ pourra être définie égale à $Z^l(Q^p)$, où le sous-problème Q^p est le sous-problème parent de Q.

Ainsi, dépendant de l'application à résoudre, on peut raffiner la sélection des sous-problèmes en considérant une somme pondérée de plusieurs critères. Notamment, on peut ajouter par exemple un critère permettant de favoriser l'exploration des sous-problèmes selon l'ordre où ils ont été générés lors de l'opération de séparation.

En pratique, plutôt que de chercher Q_s parmi tous les sous-problèmes du pool à chaque sélection, il est courant d'associer une *priorité* (P) à chacun d'eux de manière à en faciliter la gestion et la sélection. Par conséquent, on peut généraliser la notion de priorité par l'expression suivante (impliquant K critères de sélection):

$$P(Q) = \sum_{j=1}^{K} p_j C_j(Q)$$

où $C_j(Q)$ correspond à un critère associé à Q et p_j est un poids permettant de pondérer l'importance du critère j.

Politique de localisation

Une fois que le coordonnateur a sélectionné un sous-problème, il doit localiser un travailleur pour l'examiner. Le problème du coordonnateur se résume à choisir le travailleur inactif le plus approprié pour examiner le sous-problème sélectionné. Au chapitre 1, nous avons mentionné des facteurs contraignant ce choix: délais

de communication, puissance des processeurs et quantité de travail associée aux tâches. Rappelons que notre cadre conceptuel suppose des processus ayant une puissance de traitement équivalente et des délais de communication identiques entre chaque paire de processus. Le seul aspect qui reste à considérer est la quantité de travail associée aux sous-problèmes. Par conséquent, nous analysons deux cas: le cas où la quantité de travail est constante d'un sous-problème à l'autre et le cas où cette quantité est variable.

On rappelle que la stratégie d'affectation vise à balancer le travail sur l'ensemble des travailleurs. Dans le cas où la quantité de travail associée à chaque sous-problème est constante, il semble indiqué d'affecter les sous-problèmes à chaque travailleur de façon cyclique. En supposant que p travailleurs numérotés de 1 à p sont disponibles pour examiner les sous-problèmes, le coordonnateur affecte les sous-problèmes sélectionnés en respectant l'ordre numérique des travailleurs (on suppose que le travailleur suivant p est le travailleur 1). Puisque la quantité de travail effectuée par chacun des travailleurs est la même, ces derniers termineront l'examen de leur sous-problème respectif dans le même ordre que celui de leur affectation. Ainsi, le coordonnateur pourra poursuivre son affectation cyclique jusqu'à ce qu'il n'y ait plus de sous-problèmes.

Dans le cas où la quantité de travail associée à chaque sous-problème varie de l'un à l'autre, les travailleurs termineront l'examen de leur sous-problème respectif dans un ordre qui diffère de celui de leur affectation. Néanmoins, en vue de répartir également le travail, une affectation cyclique semble indiquée dans ce cas aussi. Concrètement, le choix d'un travailleur s'effectuera en balayant les travailleurs dans l'ordre numérique, mais en ne considérant que les travailleurs inactifs.

Formellement, on peut exprimer le choix de travailleur par affectation cyclique de la façon suivante. Soit j_p^* le travailleur ayant reçu le dernier sous-problème, et sachant que les processus sont numérotés de 0 à $|J-1|$, alors le coordonnateur

choisit un travailleur j^* tel que:

$$j^* = \min\{j \in J^- \mid j \ mod \ |J| > j_p^* \ mod \ |J|\}.$$

3.2.2 Détection de la fin

L'ensemble des sous-problèmes étant stockés dans un seul pool comme dans l'algorithme séquentiel, l'algorithme doit s'arrêter lorsque le pool est vide. Cependant, puisque le coordonnateur délègue l'examen des sous-problèmes aux travailleurs, il doit tenir compte du délai impliqué par ces examens. Le délai d'examen d'un sous-problème débute au moment où il est affecté à un travailleur et prend fin lorsque parviennent au coordonnateur les sous-problèmes générés ou un message indiquant un élagage. La condition d'arrêt de l'exploration peut être formulée ainsi:

$$(\Omega = \emptyset \wedge J^+ = \emptyset).$$

3.3 Approche décentralisée

Dans cette parallélisation, l'organisation de la mémoire est à multiples pools ou "collégiale", où chaque processus travailleur possède son propre pool local. Puisque le rôle premier d'un travailleur demeure l'examen de sous-problèmes et qu'en plus, il dispose d'un pool lui permettant d'y sélectionner et d'y insérer des sous-problèmes, il exécute l'algorithme de SÉP à partir des sous-problèmes de son pool. Par rapport à l'approche centralisée, dans laquelle l'information générée au cours de l'exploration se retrouve dans un seul pool, l'information générée est divisée sur l'ensemble des pools locaux. Puisque les sous-problèmes générés ne sont pas tous aussi "prometteurs", un débalancement de la quantité de travail contenue sur l'ensemble des pools est inévitable. Une stratégie d'affectation doit donc être utilisée afin de rebalancer le travail et d'exploiter pleinement le parallélisme.

Le rôle du coordonnateur consiste essentiellement à diriger la stratégie d'af-

fectation et à détecter la fin de l'exploration. À partir d'informations envoyées par les travailleurs sur l'évolution de leur recherche respective, le coordonnateur gère les affectations de sous-problèmes entre les travailleurs de façon à les maintenir actifs durant l'exploration.

En plus de son travail d'exploration, un travailleur est chargé de diffuser aux autres la valeur des meilleures solutions qu'il évalue. De plus, sous la direction du coordonnateur, il participe activement à la stratégie d'affectation en sélectionnant des sous-problèmes dans son pool local et en les affectant à des travailleurs en manque. La figure 2 représente schématiquement l'allure de la parallélisation selon l'approche décentralisée.

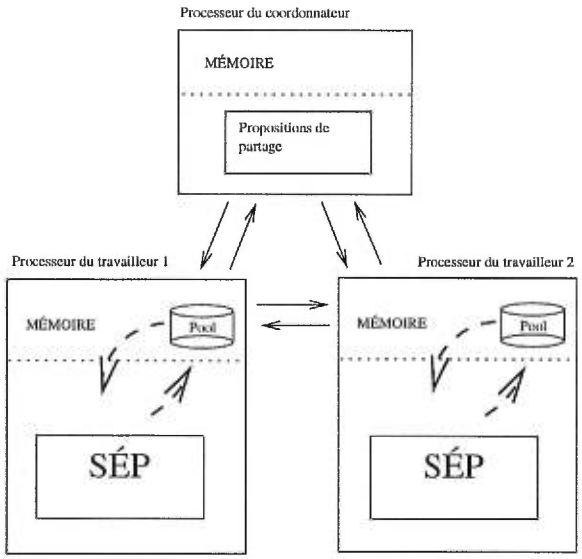

FIGURE 2: Schéma de l'approche décentralisée

Avant de décrire la stratégie d'affectation, nous présentons d'abord la notion de charge, sur laquelle elle est basée et qui tente d'évaluer la quantité de travail associée à un ensemble de sous-problèmes.

3.3.1 Notion de charge

Soit Ω un ensemble de sous-problèmes. La charge $L(\Omega)$ correspond à une estimation relative de la quantité de travail nécessaire pour explorer les sous-

problèmes de Ω.

En considérant un ensemble de sous-problèmes, il est difficile de déterminer à l'avance lesquels seront séparés ou non, et de connaître la quantité de travail associé à chacun. Il est donc nécessaire d'estimer ce travail. D'une part, un sous-problème Q_i a plus de chance d'être décomposé qu'un sous-problème Q_j si $Z^l(Q_i) < Z^l(Q_j)$. D'autre part, un sous-problème Q_i est susceptible d'être le sous-problème ancêtre d'un plus plus grand nombre de sous-problèmes qu'un sous-problème Q_j si $D(Q_i) < D(Q_j)$. Évidemment, les deux aspects sont dépendants l'un de l'autre. Il semble donc qu'une définition de charge devrait tenir compte des deux aspects.

De plus, il n'est pas nécessaire de déterminer de façon absolue la valeur de ce travail, une valeur relative basée sur une échelle est suffisante pour comparer l'effort de calcul entre deux ensembles de sous-problèmes.

Bien que plusieurs définitions puissent être imaginées, souvent dépendantes de l'application traitée, quelques définitions ont été proposées. Un grand nombre de ces définitions se généralisent par l'expression suivante (Eckstein 1996): soient $Q_1, Q_2, \ldots, Q_m$ les sous-problèmes contenus dans Ω,

$$L(\Omega) = \sum_{i=1}^{m} (\overline{Z^u} - Z^l(Q_i))^{exp}$$

où *exp* est une valeur non négative. Cette généralisation ne tient compte que de l'aspect *décomposition* des sous-problèmes: on accorde une charge plus grande à un sous-problème dont la borne inférieure est "éloignée" de la meilleure borne supérieure. À partir de la définition générale, nous en donnons deux qui ont été utilisés dans plusieurs parallélisations:

- $exp = 0$: Cette définition propose une estimation optimiste de la quantité de travail associée à Ω. En effet, cette définition associe à l'ensemble contenant m sous-problèmes la valeur m ($L(\Omega) = m$) comme si tous les sous-problèmes allaient être élagués.

- $exp = 2$: Kröger et Vornberger (1990) ont proposé cette définition où la charge est définie à partir de la variance de la borne inférieure associée aux sous-problèmes par rapport à la meilleure borne supérieure connue. Cette définition associe une plus grande quantité de travail à un sous-problème dont la borne inférieure est loin de la meilleure borne supérieure, ceci pour tenir compte de la taille de la sous-arborescence issue du sous-problème.

Intuitivement, une définition de charge devrait reposer sur un compromis entre la valeur de la borne inférieure et la profondeur. Nous ne connaissons cependant pas à ce jour de définition qui combine les deux aspects à la fois. Par contre, il nous semble indiqué d'exploiter notre définition de priorité introduite pour l'approche centralisée afin de proposer une définition de charge basée sur une somme pondérée, dépendant de la borne inférieure et de la profondeur. À partir de la définition générale de charge donnée précédemment, on aurait:

$$L(\Omega) = \sum_{i=1}^{m} \{p_1(\overline{Z^u} - Z^l(Q_i))^{exp}) + p_2(D_{\max} - D(Q_i))\}$$

où $D_{\max}$ correspond à la profondeur maximum de l'arborescence (nombre de variables à fixer) et p_1 et p_2 correspondent aux facteurs de pondération.

Cette somme pondérée permet d'associer une charge élevée à un sous-problème ayant une borne inférieure faible par rapport à la valeur de la meilleure solution réalisable, mais également d'associer en plus un "travail potentiel" au sous-problème dépendant de sa position dans l'arborescence: plus le sous-problème est profond dans l'arborescence, moins sa descendance sera grande.

3.3.2 Stratégie d'affectation

La stratégie d'affectation utilisée est dynamique et avec requêtes. Les travailleurs envoient régulièrement au coordonnateur la charge de leur pool. De plus, lorsqu'un travailleur est en manque de travail, il envoie une requête de travail au coordonnateur. Ainsi, ayant une connaissance des charges locales et des be-

soins des travailleurs, le coordonnateur est en mesure d'envoyer des propositions d'affectation. Avant d'envoyer une proposition, le coordonnateur sélectionne la requête d'un travailleur faiblement chargé et localise un travailleur hautement chargé susceptible de lui envoyer des sous-problèmes. Recevant une proposition, un travailleur vérifie s'il est en mesure de donner une partie de ses sous-problèmes, auquel cas il sélectionne un groupe de sous-problèmes et les affecte au travailleur spécifié par la proposition.

La figure 3 illustre schématiquement les étapes de la stratégie d'affectation. La stratégie d'affectation est déclenchée suite à l'envoi de requêtes et de charges des travailleurs au coordonnateur (C). Suivant la réception de la requête, le coordonnateur localise un travailleur donneur (T_i) et lui envoie une proposition d'affectation ce qui implique un envoi de sous-problèmes vers un travailleur requéreur.

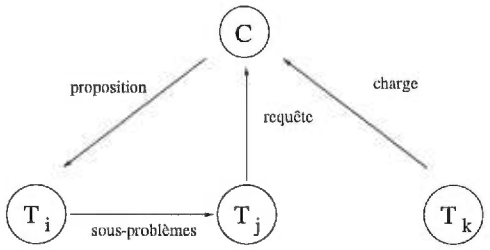

FIGURE 3: Approche décentralisée: schéma d'affectation

Politique d'information

Pour la localisation, le coordonnateur a besoin des requêtes des travailleurs (stockés dans une liste R), ainsi que de leur charge respective (L). Afin d'être en mesure de proposer des affectations aux travailleurs, le coordonnateur doit pouvoir distinguer les travailleurs hautement chargés des travailleurs faiblement chargés. Cette distinction est possible en définissant un seuil de charge (L_s). De cette façon, l'ensemble des travailleurs J peut être divisé en deux groupes. Ainsi, $J = J^+ \cup J^-$ où $J^+ = \{j | L_j > L_s\}$ (les travailleurs hautement chargés) et $J^- = \{j | L_j \leq L_s\}$ (les travailleurs faiblement chargés).

Pour la sélection, un travailleur doit connaître le seuil d'affectation (L_s) ainsi

que les critères de choix des sous-problèmes à affecter. Les critères de sélection des sous-problèmes sont discutés dans la section portant sur la politique de sélection.

Politique de déclenchement

Si un travailleur détecte que son pool est vide, ou qu'après avoir été hautement chargé, sa charge locale devient inférieure à un seuil d'avertissement (L_a), il envoie une requête au coordonnateur. Le mécanisme d'avertissement permet de démarrer la stratégie d'affectation rapidement, en espérant que le temps requis pour recevoir de nouveaux sous-problèmes recoupe celui d'examiner les derniers sous-problèmes du pool local. Ainsi, un travailleur devrait passer moins de temps inactif, favorisant ainsi une meilleure exploitation du parallélisme.

De plus, un travailleur communique périodiquement la valeur de sa charge locale au coordonnateur. Après chaque opération d'insertion ou de suppression de sous-problèmes, un travailleur vérifie si la valeur de sa charge courante remplit une condition d'envoi de charge, auquel cas il la communique au coordonnateur.

Connaissant les requêtes des travailleurs et suivant l'évolution de leur charge, le coordonnateur est en mesure d'envoyer des propositions d'affectation aux travailleurs hautement chargés. Le critère de déclenchement de la stratégie peut donc être formulé ainsi:

$$(J^+ \neq \emptyset \ \wedge \ R \neq \emptyset).$$

Politique de localisation

Comme dans l'approche centralisée, où le coordonnateur devait choisir un travailleur en mesure de recevoir un sous-problème, dans l'approche décentralisée, le coordonnateur doit choisir un travailleur en mesure de recevoir une proposition d'affectation. Dans ce contexte, le coordonnateur doit sélectionner un travailleur $j^* \in J^+$. Nous proposons deux politiques de localisation permettant de remplir cette tâche. Chaque politique proposée contient le critère de sélection du travailleur j^*, ainsi que le critère d'envoi de charge adapté à chacune.

- **Politique 1**

Selon la première politique, le coordonnateur choisit le travailleur ayant la charge la plus élevée parmi les travailleurs hautement chargés. Formellement, le critère de choix est:

$$j^* = \arg\max_{j \in J^+} L_j.$$

Un travailleur communique sa charge courante (L_c) dès qu'elle prend une valeur qui est supérieure ou inférieure d'une quantité ΔL de la dernière valeur de charge communiquée au coordonnateur (L_p). Formellement, le critère d'envoi de charge est:

$$|L_c - L_p| \geq \Delta L.$$

La valeur de ΔL est ajustée de façon à contrôler la fréquence d'envoi de charge se traduisant par un suivi plus ou moins serré de l'évolution des charges. En effet, une valeur faible de ΔL contribue à augmenter la fréquence d'envoi des charges, permettant au coordonnateur d'avoir une connaissance relativement fidèle de la valeur de charge des pools locaux. Cependant, selon le type d'application traitée et l'environnement parallèle utilisé, il n'est pas nécessairement souhaitable que la fréquence d'envoi soit élevée. Par exemple, si la charge d'un pool varie très fréquemment, en raison d'une faible quantité de travail requise par sous-problème, un nombre très élevé de charges seraient envoyées au coordonnateur, congestionnant ainsi le réseau de communication. Par contre, une connaissance peu fidèle de l'évolution des charges, en raison d'une fréquence d'envoi très faible, contribue à ce que le coordonnateur fassent des propositions d'affectations qui ne pourront être satisfaites, ce qui n'est pas souhaitable non plus.

- **Politique 2**

Selon la deuxième politique, le coordonnateur choisit le travailleur à l'aide d'un parcours cyclique sur l'ensemble des travailleurs hautement chargés. Formellement, le critère de choix est le suivant. Soit j_p^* le travailleur ayant reçu la

dernière proposition, le critère de choix de travailleur est:

$$j^* = \min\{j \in J^+ \mid j \ mod \ |J| > j_p^* \ mod \ |J|\}.$$

Comme dans la politique précédente, le critère de sélection de travailleur repose sur la connaissance de l'ensemble J^+, l'ensemble des travailleurs hautement chargés. Par conséquent, le critère d'envoi de charge précédent pourrait être utilisé par les travailleurs, afin que le coordonnateur puisse les identifier. Cependant, contrairement au critère de sélection de la politique 1, le choix de travailleur ne dépend pas de la valeur des charges. Par conséquent, un travailleur a besoin de communiquer sa charge courante (L_c) si elle vérifie la condition d'appartenance à l'ensemble J^+ ($L_c > L_s$) alors que la dernière valeur de charge qu'il a communiquée au coordonnateur (L_p) vérifiait la condition d'appartenance à l'ensemble J^- ($L_p \leq L_s$), et inversement. Concrètement, cette condition signifie que tant que la charge courante d'un pool demeure supérieure au seuil d'affectation L_s, un travailleur n'a pas à envoyer sa charge et inversement. Par rapport au critère d'envoi de la politique 1, celui-ci possède le net avantage de diminuer considérablement le nombre d'envois de charge nécessaire à la stratégie d'affectation. Formellement, le critère d'envoi de charge est:

$$((L_c > L_s) \ \wedge \ (L_p \leq L_s)) \ \vee \ ((L_c \leq L_s) \ \wedge \ (L_p > L_s)).$$

Politique de sélection

Si un travailleur reçoit une proposition d'affectation, il vérifie à l'aide du seuil d'affectation L_{min} s'il possède au moins la charge minimale pour donner un ou plusieurs sous-problèmes. Si c'est le cas, il doit sélectionner dans son pool local les sous-problèmes à affecter.

Idéalement, on souhaiterait que les travailleurs restent occupés durant tout le processus de résolution, travaillant sur des sous-problèmes "prometteurs", tout en minimisant le nombre de rééquilibrages de charge entre les pool locaux. De façon à donner du travail "prometteur" aux travailleurs, on souhaiterait respecter le plus

possible l'ordre d'exploration des sous-problèmes défini par la règle de sélection de l'algorithme séquentiel. Dans une architecture à échange de messages, un délai est associé à la communication d'un sous-problème. Ce délai dépend de la taille du sous-problème (la quantité d'information qu'il contient), ainsi que des caractéristiques physiques des liens de communication inter-processeurs. De plus, chaque sous-problème constitue la racine d'une sous-arborescence dont la taille peut varier. La taille de cette arborescence est difficile à déterminer, mais peut, dans certains cas, être estimée à l'aide d'une définition de charge appropriée. Le choix d'une politique de sélection repose donc sur trois aspects:

- l'ordre d'exploration;
- les délais de communication;
- le travail potentiel associé aux sous-problèmes.

Afin d'illustrer l'impact de ces aspects, considérons le cas où la politique de sélection correspond à la règle de sélection de l'algorithme de SÉP.

Du point de vue d'une stratégie d'affectation, respecter l'ordre d'exploration des sous-problèmes consiste à éviter d'affecter des sous-problèmes dont l'examen se ferait plus tardivement que s'ils étaient conservés localement. Autrement dit, la sélection des sous-problèmes doit tenir compte du délai séparant le moment de l'envoi des sous-problèmes du moment de leur examen. Entre ces deux moments, le délai est déterminé par le temps de communication des sous-problèmes et par le temps de stockage dans le pool du travailleur destinataire.

Considérons les deux cas suivants:

1. Les délais de communication sont négligeables. Puisque dans notre stratégie d'affectation, un travailleur requéreur a terminé ou est sur le point de terminer l'exploration de ses sous-problèmes locaux, les sous-problèmes affectés seront examinés rapidement, ce qui tend à conserver l'ordre d'exploration.
2. Les délais de communication sont non négligeables. L'examen des sous-problèmes est alors retardé, résultant en un changement de l'ordre d'ex-

ploration. Pour pallier cette situation, il faudrait modifier la politique de sélection de façon à sélectionner des sous-problèmes dont l'examen peut être retardé d'un temps au moins égal au temps de communication de ces sous-problèmes.

Le travail potentiel associé aux sous-problèmes constitue un facteur non négligeable de la politique de sélection. D'une part, affecter un sous-problème dont le travail potentiel est faible fait en sorte qu'un équilibrage sera nécessaire à court terme, étant donné que le travailleur récepteur, ayant peu de sous-problèmes à examiner, risque d'être occupé pendant une courte période de temps. D'autre part, affecter plusieurs sous-problèmes à la fois, dont le travail potentiel associé à chacun est élevé, peut retarder grandement l'examen de certains de ces sous-problèmes. Donc, on utilise la notion de travail potentiel de façon à contrôler le choix et la quantité des sous-problèmes à affecter.

La politique de sélection de la stratégie d'affectation repose donc sur un compromis entre la qualité d'un sous-problème, définie par la règle de sélection "locale" d'un travailleur, ainsi que sur le travail potentiel associé au sous-problème.

À chaque sous-problème Q, on peut associer une priorité qui est une somme pondérée de la forme:

$$P(Q) = \sum_{j=1}^{K} p_j C_j(Q) + p_{K+1} L(Q)$$

où $C_j(Q)$ est le j^{eme} critère de sélection associé à Q, $L(Q)$ représente le travail potentiel de Q, et les facteur p_j correspondent aux poids permettant de pondérer l'importance du critère j.

Mentionnons les travaux de Dowaji (1995) dans lesquels on définit une notion de priorité de sélection tenant compte de la borne inférieure, du travail potentiel et de la profondeur des sous-problèmes. Pour les applications testées, le travail potentiel est défini comme le nombre de variables non-fixées.

3.3.3 Détection de la fin

Afin de décréter la fin du processus de résolution, le coordonnateur doit s'assurer que toute activité a cessé dans le réseau: aucun travailleur n'est actif et aucun message n'est en circulation. Avant de formuler un critère d'arrêt, il faut se demander sur quelles informations le coordonnateur peut appuyer sa décision. À tout moment durant le processus de résolution, le coordonnateur sait:

- quels travailleurs sont en attente de travail;
- s'il a envoyé une proposition à un travailleur donneur t_i pour une requête du travailleur demandeur t_j;
- s'il a reçu une réponse positive ou négative du travailleur demandeur t_j.

De plus, on sait que le coordonnateur possède de l'information sur la charge des travailleurs. Cependant, cette information n'est pas tout à fait fiable en raison de la rapidité de diffusion de l'information associée à la charge des travailleurs, cette rapidité dépendant d'une part du critère d'envoi de charge et d'autre part des délais de transmission. Autrement dit, à tout moment du processus de résolution, le coordonnateur ne peut jamais être certain de la charge exacte des travailleurs.

Puisque le coordonnateur connaît les travailleurs en attente, il sait quels travailleurs sont actifs ou inactifs. La résolution ne peut donc pas se terminer avant que le nombre de travailleurs inactifs soit égal au nombre total de travailleurs. Cependant, cette condition représente une condition nécessaire mais non suffisante à l'arrêt de la résolution. En effet, le coordonnateur doit s'assurer également que la procédure de formulation de propositions, qui implique la circulation d'un certain nombre de messages (propositions, sous-problèmes, réponses) a pris fin. C'est pourquoi, tant que le coordonnateur n'a pas de confirmation sur l'issue des propositions envoyées (réussite ou échec), il ne met pas fin à l'exploration.

C'est pourquoi le coordonnateur peut décréter la fin du processus de résolution lorsque deux conditions sont remplies:

1. l'issue de toutes les propositions d'affectation de sous-problèmes est connue;
2. tous les travailleurs sont en attente de travail.

3.4 Approche hybride

L'approche hybride utilise une organisation de la mémoire qui combine les organisations précédentes: un pool global géré par le coordonnateur, accessible par tous les travailleurs, et un pool local à chacun des travailleurs. Comme dans l'approche décentralisée, un travailleur exécute l'algorithme de SÉP sur les sous-problèmes de son pool. Le coordonnateur et les travailleurs participent activement à la stratégie d'affectation. Contrairement à l'approche décentralisée, les mouvements de sous-problèmes ne sont pas permis entre les travailleurs: pendant son exploration locale, un travailleur identifie un certain nombre de sous-problèmes qu'il affecte au coordonnateur et, à son tour, le coordonnateur sélectionne les sous-problèmes les plus susceptibles de mener à la solution optimale et les réaffecte aux travailleurs en manque de travail. Un travailleur diffuse la valeur des meilleures solutions qu'il évalue.

L'approche hybride peut être vue comme une approche centralisée à laquelle on ajoute un pool à chacun des travailleurs, permettant ainsi au coordonnateur d'affecter plus d'un sous-problème à la fois et aux travailleurs la possibilité de conserver localement des sous-problèmes générés. On peut aussi voir l'approche hybride comme une approche décentralisée à laquelle on ajoute un pool géré par le coordonnateur et dont la stratégie d'affectation est modifiée. La figure 4 représente schématiquement l'allure de la parallélisation selon l'approche hybride.

3.4.1 Stratégie d'affectation

La stratégie d'affectation est dynamique et combinée. Elle se divise en deux phases: une phase travailleur-coordonnateur, dans laquelle les travailleurs envoient des sous-problèmes au coordonnateur et une phase coordonnateur-

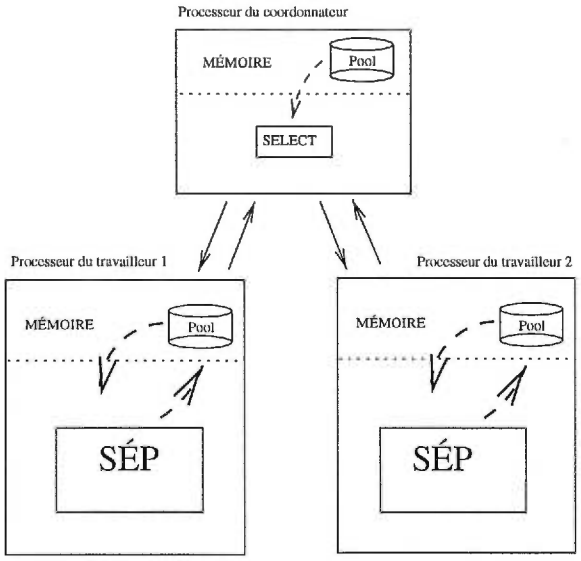

FIGURE 4: Schéma de l'approche hybride

travailleur, dans laquelle le coordonnateur envoie des sous-problèmes aux travailleurs suite à des requêtes de ces derniers. La figure 5 illustre schématiquement cette stratégie d'affectation. Chaque phase constitue en elle-même une stratégie d'affectation complète. Nous décrivons chacune d'elles.

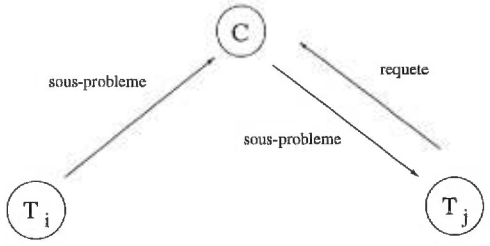

FIGURE 5: Approche hybride: schéma d'affectation

3.4.1.1 Phase travailleur-coordonnateur

La première phase est une stratégie d'affectation dynamique sans requête à l'initiative du donneur, en l'occurrence un travailleur. À mesure qu'il insère des sous-problèmes dans son pool local, il en sélectionne un certain nombre qu'il envoie au coordonnateur. Ces sous-problèmes dits de *deuxième qualité* sont des sous-problèmes "prometteurs", mais dont l'examen peut être retardé. En recevant

ces sous-problèmes, le coordonnateur les insère dans le pool central.

Politique d'information

Pour la politique de sélection, les critères associés aux sous-problèmes du pool local de chacun des travailleurs (Ω_j) sont utilisés. Pour la politique de localisation, aucune information n'est nécessaire, puisque le travailleur n'affecte des sous-problèmes qu'au coordonnateur.

Politique de déclenchement

Après chaque opération de séparation, le travailleur insère les sous-problèmes générés dans le pool local Ω_j, sauf un qu'il conserve pour examen. Suite à l'insertion, il détermine la présence ou l'absence de sous-problèmes de deuxième qualité en appliquant la politique de sélection. Si un ou plusieurs sous-problèmes sont identifiés, ils sont envoyés illico au coordonnateur. Autrement dit, c'est l'issue de la politique de sélection qui sert de critère de déclenchement.

Politique de localisation

Dans cette phase, la politique de localisation est triviale: après avoir sélectionné les sous-problèmes de "deuxième qualité", un travailleur les affecte uniquement au coordonnateur.

Politique de sélection

Le but de la politique de sélection, dans le cadre de la phase travailleur-coordonnateur, consiste à choisir des sous-problèmes destinés à des travailleurs en manque de travail. On rappelle les trois aspects essentiels à l'établissement d'une politique de sélection:

- l'ordre d'exploration;
- les délais de communication;
- le travail potentiel associé aux sous-problèmes.

D'une part, on ne voudrait pas affecter des sous-problèmes de mauvaise qualité, ce qui résulterait en un surcoût de travail. D'autre part, on souhaiterait

également respecter l'ordre d'exploration des sous-problèmes défini par la règle de sélection de l'algorithme de SÉP ("profondeur d'abord", "meilleur d'abord" ou une combinaison des deux).

Dans ce contexte, même si le temps de communication est inférieur au temps d'examen d'un sous-problème, le début de l'examen est retardé en raison du stockage intermédiaire dans le pool central, retard qui dépend de l'état d'activité des travailleurs. Afin de tenir compte de ce retard, on choisit des sous-problèmes dont l'examen serait retardé par la sélection locale. On appelle une telle sélection de sous-problème, une sélection de "deuxième qualité".

La notion de travail potentiel n'a pas à intervenir dans la présente politique de sélection. Étant donné qu'un travailleur envoie des sous-problèmes de "deuxième qualité" à mesure qu'ils sont disponibles et que chaque travailleur n'a aucune connaissance de l'état d'activité des autres travailleurs, il n'a pas avantage à tenir compte du travail potentiel associé aux sous-problèmes qu'il affecte.

La politique de sélection peut être intégrée de deux façons: soit elle est appliquée à chaque opération de séparation, alors qu'un travailleur sélectionne un ensemble de sous-problèmes qui seraient insérés dans le pool local, soit le travailleur sélectionne un sous-problème du pool local seulement lorsqu'un critère de sélection est satisfait.

Par exemple, dans un contexte où l'opération de séparation est dichotomique (l'opération de séparation génère deux sous-problèmes), et que l'ordre des opérations est défini selon la règle "profondeur d'abord", un sous-problème de deuxième qualité pourrait correspondre au sous-problème complémentaire à celui sélectionné pour examen, c'est-à-dire celui qui est inséré dans le pool central en attente d'examen. Dans ce cas, la politique de sélection est appliquée à chaque opération de séparation.

3.4.1.2 Phase coordonnateur-travailleur

La deuxième phase est une stratégie d'affectation dynamique avec requêtes à l'initiative du receveur. Les travailleurs qui sont en manque de travail envoient une requête au coordonnateur. Lorsque le pool central est non vide, le coordonnateur sélectionne une requête, puis un groupe de sous-problèmes et les affecte au travailleur requéreur spécifié par la requête.

Politique d'information

Comme dans l'approche centralisée, le coordonnateur reçoit des sous-problèmes et des requêtes des travailleurs. D'une part, les critères associés à chacun des sous-problèmes du pool central (Ω) lui permettent de les sélectionner. D'autre part, les requêtes de l'ensemble R lui permettent de localiser les travailleurs receveurs de sous-problèmes.

Politique de déclenchement

Lorsque le coordonnateur reçoit une requête d'un travailleur, il l'insère dans l'ensemble R. De même, lorsqu'il reçoit des sous-problèmes, il les insère dans le pool central. Lorsqu'il détecte que la liste R et que le pool Ω sont non vides, il sélectionne un groupe de sous-problèmes, sélectionne une requête et les affecte au travailleur requéreur. Formellement, le critère de déclenchement est:

$$(\Omega \neq \emptyset \ \wedge \ R \neq \emptyset).$$

Politique de localisation

Pour cette phase, la politique de localisation est simple. Elle consiste à localiser les receveurs de sous-problèmes. Cette localisation est implicite au choix de la requête de l'ensemble R.

Politique de sélection

Lorsqu'un travailleur fait une demande de travail, le coordonnateur tente de lui affecter le ou les sous-problèmes qui sont les plus "prometteurs". Comme nous le

mentionnions dans la section traitant de l'approche centralisée, il est courant d'associer une priorité (P) à chacun des sous-problèmes du pool central Ω de manière à en faciliter la sélection. Par conséquent, le coordonnateur utilise une sélection "multi-critères" de façon à choisir les sous-problèmes les plus prometteurs. On rappelle l'expression de la priorité de sélection:

$$P(Q) = \sum_{j=1}^{K} p_j C_j(Q)$$

où $C_j(Q)$ correspond à un critère associé à Q et p_j est un poids permettant de pondérer l'importance du critère j.

3.4.2 Détection de la fin

La détection de la fin est identique à celle utilisée pour l'approche centralisée. Afin de décréter la fin du processus de résolution, le coordonnateur doit s'assurer que tous les travailleurs sont inactifs: ils ont examiné tous les sous-problèmes de leur pool local. Il est nécessaire également que tous les sous-problèmes aient été examinés: le pool global est vide et aucun sous-problème n'est en route vers un travailleur.

Puisque le coordonnateur sait quels travailleurs sont en attente de travail et connaît l'état du pool global, le coordonnateur peut décréter la fin du processus de résolution lorsque deux conditions sont remplies:

1. le pool global est vide;
2. tous les travailleurs sont en attente de travail.

Formellement, le critère d'arrêt est le suivant:

$$\Omega = \emptyset \ \wedge \ |R| = p.$$

CHAPITRE 4

Mise en oeuvre

Le présent chapitre décrit l'implantation des trois parallélisations de type 2 de l'algorithme de SÉP, présentées au chapitre précédent. Ces implantations sont appliquées à la parallélisation de l'algorithme séquentiel de SÉP permettant de résoudre le problème de localisation multi-produits avec équilibrage.

Ce chapitre est divisé en six sections. La première expose l'environnement de développement des parallélisations. La seconde présente la structure du code parallèle. La troisième décrit l'interface Fortran/C qui a été nécessaire pour fusionner nos parallélisations, écrites en langage C, avec l'algorithme séquentiel de SÉP, réalisé en langage Fortran. Les trois sections suivantes concernent l'implantation des trois parallélisations.

4.1 Environnement de développement

Afin de réaliser nos implantations, nous utilisons le langage de programmation structuré C, langage multi-usage ne possédant pas de fonctions particulières adaptées à un environnement parallèle.

Afin de pallier ce manque de fonctionnalité, nous utilisons la bibliothèque de fonctions PVM (Geist, Beguelin, Dongarra, Jiang, Manchek et Sunderam 1993). Cette bibliothèque de fonctions permet de simuler un environnement parallèle sur un environnement distribué. La bibliothèque permet l'échange de messages entre processus s'exécutant sur un réseau de stations de travail hétérogènes. Nous décrivons brièvement son fonctionnement.

Chacun des processus est un *code exécutable* qui réside sur une machine. Un tel code est obtenu après création d'un *code source* contenant les instructions en langage C auxquelles on a ajouté des appels aux fonctions PVM. Après compilation de ce code source, on obtient un *code objet.* Ce dernier, joint à la bibliothèque PVM précompilée, permet d'obtenir le code exécutable suite à une phase d'édition des liens. La figure 6 illustre les étapes de création d'un code exécutable.

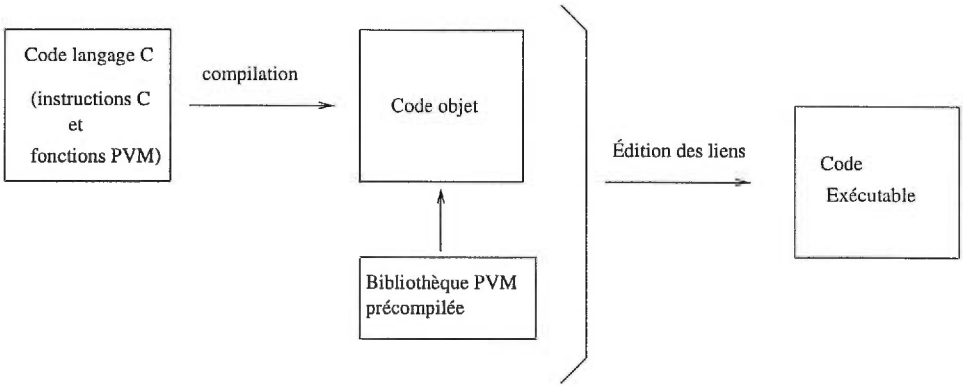

FIGURE 6: Création d'un code exécutable

PVM permet, d'une part, de démarrer des processus communicants sur un réseau de stations de travail et, d'autre part, de gérer les échanges de messages entre ces processus. Afin de démarrer la machine parallèle virtuelle, on confie généralement à un processus la responsabilité de démarrer les autres, en spécifiant le code exécutable à démarrer, ainsi que la machine d'exécution.

Des fonctions sont responsables de la gestion de messages entre les processus: des fonctions d'envoi de messages et des fonctions de réception de messages. Un envoi de message peut donc s'établir entre deux processus sachant que le premier fait un appel à une fonction d'envoi de message et que le deuxième fait un appel à une fonction de réception de message. Il est important de noter que l'envoi et la réception ne sont pas tenus de s'effectuer au même moment. Un message envoyé est stocké temporairement dans un espace mémoire réservé ("buffer") de la machine de destination. La machine de destination peut être la même que celle d'où l'envoi a été fait. Ceci est possible lorsque la communication s'établit

entre deux processus s'exécutant sur la même machine. L'appel à la fonction de réception permet donc de vérifier le contenu de la zone mémoire réservée. La figure 7 illustre schématiquement l'envoi d'un message P entre un processus A s'exécutant sur une station de travail i et un processus B s'exécutant sur une station de travail j.

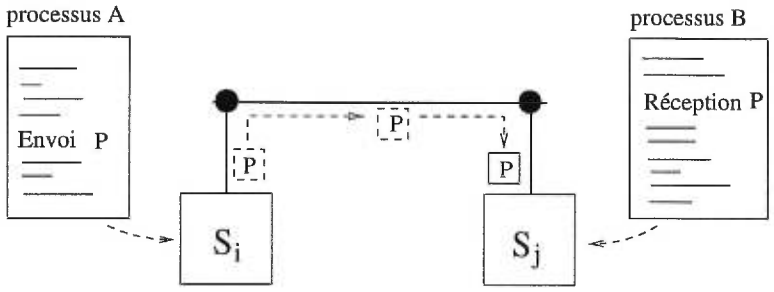

FIGURE 7: Processus communicants

L'envoi d'un message se fait en trois étapes:

- *initialisation*: Étape qui permet de réserver l'espace mémoire nécessaire à l'établissement d'une communication entre processus.
- *empaquetage*: Considérant les types de données de base manipulés par le langage C (les nombres entiers, les nombres réels, les caractères), PVM contient des fonctions d'empaquetage adaptées à chacun de ces types. Le rôle de l'empaquetage consiste à uniformiser la structure des messages véhiculés dans la "machine virtuelle". L'empaquetage est particulièrement nécessaire lorsque les communications s'établissent entre des stations hétérogènes dont la représentation interne des données peut différer. Dans notre contexte, les stations utilisées sont homogènes.
- *envoi*: Cette étape permet de spécifier l'adresse du processus de destination du message.

La réception d'un message se fait en deux étapes:

- *vérification de réception*: Afin de vérifier la réception d'un message, un processus fait appel à une fonction qui inspecte la zone mémoire réservée aux messages reçus. PVM offre deux types de fonctions de vérification de

réception: les fonctions bloquantes et non bloquantes. Lorsqu'un processus fait appel à une fonction bloquante, son exécution est suspendue jusqu'à ce que la zone mémoire réservée contienne un message. Par contre, l'appel à une fonction non bloquante inspecte si la zone mémoire contient un message et retourne immédiatement l'état de la zone.

- *dépaquetage*: Une fois que l'étape de vérification de réception confirme la présence d'un message dans la zone réservée, le processus fait appel à une fonction de dépaquetage. Symétriquement à l'étape d'empaquetage de l'envoi, l'étape de dépaquetage permet de convertir les informations contenus dans un message en types en données de base.

La figure 8 illustre la communication par message entre un processus A et un processus B à l'aide d'un exemple d'entité (X) et d'un pseudo-code utilisant les fonctions PVM.

Soit une entité X formée des attributs suivants:

info1: ..... un scalaire
info2[10]: ..... un tableau de 10 nombres réels
info3[5]: ..... un tableau de 5 caractères

Soit adr(B): l'adresse du processus B fournie par PVM à la configuration

code du processus A

```
{initialisation}
 pvm_init()
{empaquetage}
 pvm_pkint(info1,1)
 pvm_pkfloat(info2,10)
 pvm_pkstring(info3,5)
{envoi}
 pvm_send(adr(B))
```

code du processus B

```
{vérification de réception}
  pvm_recv()
si (entité X reçue) alors
   {dépaquetage}
    pvm_upkint(info1,1)
    pvm_upkfloat(info2,10)
    pvm_upkstring(info3,5)
```

FIGURE 8: Communication par messages (fonctions PVM)

4.2 Structure du code parallèle

4.2.1 Description d'un code général

En accord avec le schéma conceptuel que nous avons proposé au chapitre précédent qui définit l'environnement parallèle comme un ensemble de processus communicants, nous spécifions un code exécutable correspondant au processus coordonnateur, ainsi qu'un code exécutable correspondant au processus travailleur.

La structure de code du coordonnateur et d'un travailleur est la même. Celle-ci est divisée en cinq parties:

1. *configuration de l'environnement parallèle*: Dans le cas du coordonnateur, cette partie lui permet de lire le fichier contenant la configuration de la machine parallèle, ainsi que l'emplacement des fichiers de données et de paramètres. C'est lui qui démarre les processus travailleurs sur les machines désignés par le fichier de configuration. Cette étape de démarrage lui permet de bâtir un tableau des adresses des travailleurs qui seront nécessaires pour communiquer avec eux. Une fois le tableau bâti, il le communique à tous les travailleurs. Dans le cas d'un travailleur, lorsqu'il est créé, il fait appel à une fonction PVM afin de connaître l'adresse de son "créateur", en l'occurrence le coordonnateur. Cette adresse lui permettra notamment de recevoir le tableau des adresses des travailleurs.

2. *lecture des données et paramètres*: Le coordonnateur aura à lire l'ensemble ou une partie des données concernant le problème à résoudre de façon à définir au moins le problème original s'il gère un pool de sous-problèmes. De plus, il doit lire, d'une part, les paramètres propres à la résolution du problème et, d'autre part, les paramètres nécessaires à la parallélisation (par exemple pour la stratégie d'affectation). Étant donné que ce sont les travailleurs qui sont responsables de l'examen des sous-problèmes, ils doivent lire les données et paramètres du problème afin de définir le problème original et

de pouvoir appliquer les opérations d'évaluation et de séparation. Dans certains cas, il devront lire un certain nombre de paramètres supplémentaires reliés à la parallélisation.

3. *initialisation des variables*: Une fois les données et paramètres lus, chaque processus initialise les variables nécessaires à son traitement.
4. *bloc principal*: Cette partie comprend le code principal de chacun des processus. Elle contient les actions effectuées par chacun des processus pendant l'exploration de l'arborescence. La structure du bloc principal est identique d'un processus à l'autre: elle est formée d'une boucle principale à l'intérieur de laquelle on retrouve les instructions du traitement local parmi lesquelles on retrouve également les appels aux fonctions nécessaires aux communications. Cette boucle s'arrête lorsque la condition d'arrêt est vérifiée.
5. *affichage des résultats*: Une fois la fin de l'exploration déclarée, il est nécessaire de connaître la solution au problème considéré.

La figure 9 présente la structure générale du code d'un processus.

4.2.2 Messages

Des messages particuliers sont échangés entre les processus démarrés afin de mettre en oeuvre les stratégies de parallélisation. De façon à faciliter davantage l'échange des informations, nous avons défini nos propres fonctions de communications basées sur celles de la bibliothèque PVM. À partir de regroupements des types de données de base du langage C, on crée de nouveaux types de données plus complexes. Ces nouveaux types correspondent à des entités qui doivent être véhiculées entre les processus participant à la résolution: d'une part, on retrouve des entités concernant l'algorithme de SÉP tels que les sous-problèmes et les bornes, d'autre part on retrouve des entités nécessaires à la gestion des parallélisations tels que les requêtes et les messages de fin. Exploitant la structure de chacune de ces entités, nous avons développé un mécanisme d'envoi et de réception de messages:

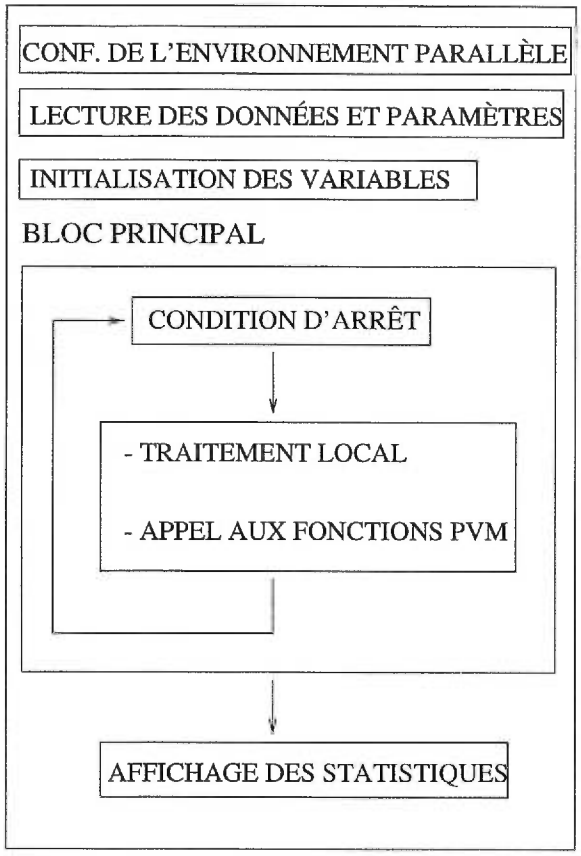

FIGURE 9: Structure du code d'un processus

- Envoi de message:

  1. fonction d'initialisation: En plus de réserver l'espace mémoire nécessaire à la communication par l'appel à la fonction d'initialisation de PVM, cette fonction permet aussi de créer un entête de message adapté au type d'entité communiqué. L'entête de message contient une identification du type d'entité contenu dans le message ainsi que le nombre d'entités de même type contenu dans le message. En effet, nous supposons qu'un message ne concerne qu'un seul type d'entité à la fois. La création de cet entête se fait par empaquetage des données propres à l'entête.
  2. fonction d'empaquetage: Pour une entité donnée, définie à partir des types de données de base, cette fonction d'empaquetage correspond à une séquence déterminée d'appels aux fonctions d'empaquetage de

PVM.

3. fonction d'envoi: Une fois le message empaqueté, cette fonction permet d'envoyer le message à destination. Bien que nous ayons redéfini cette fonction, elle ne contient qu'un appel à la fonction d'envoi définie par PVM.

- Réception de message:

1. fonction de réception: Fonction qui contient un appel à une fonction de vérification de réception définie par PVM, suivi du dépaquetage de l'entête dans le cas où une réception est détectée.
2. fonction de dépaquetage: Symétriquement à la fonction d'empaquetage, la fonction de dépaquetage associée à une entité donnée permet de reconstituer une entité à partir des données empaquetées dans le message. Une séquence d'appels aux fonctions de dépaquetage de PVM correspondant à la séquence appliquée lors de l'empaquetage permet de reconstituer l'entité contenue dans le message.

Reprenant notre exemple précédent, en utilisant nos propres fonctions de communications, la communication se fait selon le pseudo-code illustré à la figure 10.

Soient N entités de type X: (x1, x2,...,xN)

code du processus A

```
{formation de l'entête}
 Init_Envoi(type X, N)
{empaquetage}
 pour i=1 à N faire
  Empaq_EntiteX(xi)
{envoi}
  Envoi_message(adr(B))
```

code du processus B

```
{vérification de réception}
 Message_recu(type, N)
si (message reçu) alors
  si(type = type X) alors
    pour i= 1 à N faire
     {dépaquetage}
      Depaq_EntiteX(xi)
```

FIGURE 10: Communication par messages: nos fonctions

4.3 Interface Fortran/C

L'algorithme de SÉP séquentiel pour la résolution du problème de localisation multi-produits avec équilibrage décrit au chapitre 2, était originalement écrit en langage Fortran. Afin de l'intégrer à nos parallélisations, nous avons traduit partiellement l'algorithme en langage C. Nous avons conservé en langage Fortran les procédures d'évaluation et de séparation, procédures qui auraient nécessité un travail de traduction fastidieux. À la place, nous avons *interfacé* notre code écrit en C avec les procédures écrites en Fortran. Cet interfaçage permet de faire des appels aux procédures Fortran directement à partir du code C.

Après compilation du code Fortran, le code objet obtenu contient des étiquettes correspondant aux "commons" (blocs de variables globales) ainsi qu'aux procédures du code Fortran. Le code source en langage C peut faire usage de ces étiquettes afin d'appeler les procédures Fortran et d'accéder aux zones mémoire réservées aux "commons" du code Fortran. Afin de créer le code exécutable, les codes objet obtenus des codes sources en C et en Fortran seront liés ensembles après compilation.

L'appel d'une procédure Fortran à partir du code C se fait en trois étapes:

1. *mise à jour des "commons"*: Avant d'appeler la procédure Fortran, le code C doit mettre à jour les données d'entrée nécessaires à son exécution.
2. *appel de la procédure*: Une fois les "commons" intialisés pour l'exécution de la procédure, un appel à la fonction Fortran est fait.
3. *récupération des résultats*: Une fois la procédure Fortran complétée, le code C peut disposer des résultats de la procédure en accédant aux "commons" modifiés par la procédure.

La figure 11 illustre les trois étapes de l'interfaçage Fortran/C.

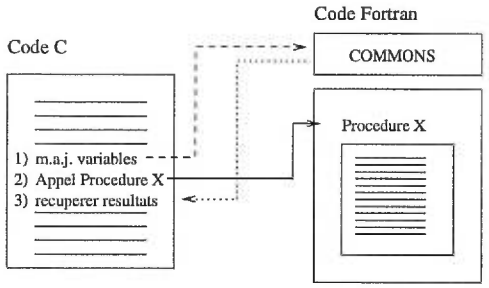

FIGURE 11: Schéma de l'interfaçage Fortran/C

4.4 Approche centralisée

Comme nous l'avons déjà mentionné, le rôle principal d'un travailleur consiste à examiner des sous-problèmes. Le second rôle des travailleurs consiste à diffuser la valeur des meilleures solutions réalisables qu'il détermine. Ce rôle fait partie de la stratégie de gestion de la borne supérieure.

4.4.1 Fonctionnement

Initialement, le coordonnateur possède le problème original dans son pool et tous les travailleurs sont inactifs. Il démarre l'exécution de la parallélisation en affectant le problème original à un des travailleurs. À mesure que les sous-problèmes générés à partir du problème original deviennent disponibles, ces derniers sont affectés aux travailleurs. Étant donné qu'un travailleur ne possède pas de pool local, un travailleur ne peut recevoir qu'un sous-problème à la fois. Un travailleur recevant un sous-problème est considéré actif.

Au terme de l'examen d'un sous-problème (ou du problème original pour le travailleur qui débute l'exploration), un travailleur envoie les résultats de l'examen au coordonnateur: les sous-problèmes obtenus suite à l'application de l'opération de séparation ou un message d'élagage indiquant que le sous-problème ne peut être séparé. La réception de ces résultats indique au coordonnateur que le travailleur est de nouveau inactif.

Dans le cas où le coordonnateur reçoit des sous-problèmes, il les insère dans son pool. Tant que le pool est non vide et qu'au moins un travailleur est inactif, le coordonnateur déclenche la stratégie d'affectation. Après chaque affectation, le coordonnateur vérifie l'arrivée de nouveaux messages: des résultats d'examen ou de nouvelles bornes.

Le coordonnateur et les travailleurs conservent en mémoire la valeur de la meilleure solution réalisable connue (une borne supérieure). Si un travailleur obtient une meilleure borne suite à l'application de l'opération d'évaluation, sa valeur remplace celle qu'il a déjà en mémoire et il la diffuse au coordonnateur et aux autres travailleurs.

L'exécution se poursuit ainsi jusqu'à ce que le pool soit vide et que tous les travailleurs soient inactifs. À ce moment, le coordonnateur diffuse un message de fin aux travailleurs.

4.4.2 Messages

Dans cette approche, on retrouve les types de messages suivants:

- *sous-problèmes* (spbms): Des communications de sous-problèmes sont nécessaires entre le coordonnateur et les travailleurs: d'une part, le coordonnateur rend accessibles les sous-problèmes du pool en les affectant aux travailleurs inactifs et, d'autre part, les travailleurs renvoient au coordonnateur les sous-problèmes générés obtenus après séparation.
- *bornes* (borne): Les travailleurs diffusent aux autres processus les meilleures bornes qu'ils évaluent.
- *messages d'élagage* (élag): Si l'examen d'un sous-problème révèle qu'il n'a pas à être séparé, soit parce qu'il est résolu, soit parce que le test de la borne inférieure a permis de l'éliminer, un travailleur envoie un message d'élagage au coordonnateur.

- *fin*: Lorsque le critère d'arrêt de l'exploration est satisfait, le coordonnateur diffuse un message de fin aux travailleurs.

La figure 12 illustre les types de messages échangés entre les processus dans l'approche centralisée.

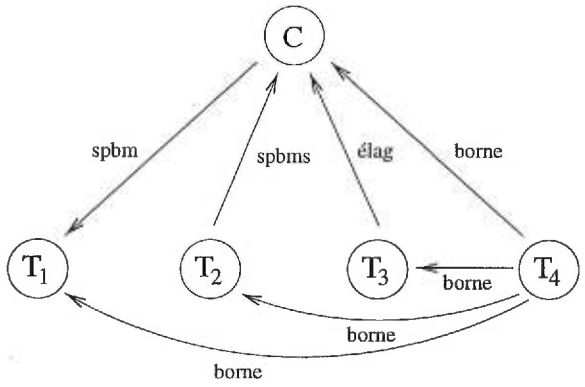

FIGURE 12: Approche centralisée: types de messages

4.4.3 Code du coordonnateur

Les rôles du coordonnateur comprennent la gestion du pool, l'application de la stratégie d'affectation et la détection de la fin de l'exploration.

4.4.3.1 Données et paramètres

- Données:
  - Q^*: problème original;
  - Ω: pool central;
  - $\overline{Z^u}$: meilleure borne supérieure;
  - $A[j]$: tableau des états d'activité ($ACTIF$ ou $INACTIF$) des travailleurs;
  - A^+: nombre de travailleurs actifs;
  - j^*: numéro du travailleur localisé.
- Paramètres:

- T: nombre de travailleurs;
- p_1, p_2, p_3: pondération des critères de la règle de sélection (ces paramètres seront introduits ci-dessous).

4.4.3.2 Bloc principal

A) Critère d'arrêt

Comme dans l'algorithme séquentiel, le coordonnateur étant le processus qui exécute la méthode de SÉP, il ne peut mettre fin à l'exploration tant que le pool est non vide. Cependant, un pool vide ne constitue pas une condition suffisante pour décréter la fin. En effet, si un travailleur est actif, son examen peut générer de nouveaux sous-problèmes à examiner. Le critère d'arrêt s'exprime donc comme suit:

$$A^+ = 0 \ \wedge \ \Omega = \emptyset.$$

B) Gestion du pool

À la réception de sous-problèmes, le coordonnateur évalue leur priorité et les insère dans le pool. Afin de faciliter l'insertion et la sélection des sous-problèmes, le pool possède une structure de *monceau*. À la sélection d'un problème du pool, le coordonnateur tente de l'éliminer en appliquant le test de la borne inférieure. Le coordonnateur utilise la valeur $\overline{Z^u}$ pour appliquer ce test. Étant donné que le pool est constitué de sous-problèmes non évalués, la borne inférieure du sous-problème parent est utilisée ($Z^l(Q^p) < \overline{Z^u}$).

C) Stratégie d'affectation

Le coordonnateur sélectionne un sous-problème (politique de sélection), localise un travailleur receveur (politique de localisation) et affecte le sous-problème au travailleur.

- Politique de déclenchement: Tant que des travailleurs sont disponibles et que des sous-problèmes sont disponibles dans le pool, le coordonnateur est en mesure de réaliser une affectation. En terme des données définies, le

critère de déclenchement de la stratégie d'affectation s'exprime ainsi:

$$(A^+ < T) \ \wedge \ (\Omega \neq \emptyset).$$

- Politique de sélection: Trois critères associés aux sous-problèmes sont utilisés afin de leur attribuer une priorité. Soit Q un sous-problème de Ω, on considère la profondeur de Q $(D(Q))$, la borne inférieure du sous-problème parent $(Z^l(Q^p))$ et la valeur de la variable fixée pour obtenir Q (pour le problème de localisation multi-produits avec équilibrage, $F(Q) = 0$ ou 1). L'expression de la priorité est la suivante:

$$P(Q) = p_1(-D(Q)) + p_2(Z^l(Q^p)) + p_3(F(Q)).$$

- Politique de localisation: On utilise un parcours cyclique des travailleurs pour localiser un travailleur récepteur. Une variable j^* contient le numéro du dernier travailleur affecté. Une fois le travailleur localisé, le tableau A et le scalaire A^+ sont mis à jour:

  $j^* \leftarrow (j^* + 1) \bmod T$
  Tant que $(A[j^*] = ACTIF)$ faire
  $\quad j^* \leftarrow (j^* + 1) \bmod T$
  $A[j^*] \leftarrow ACTIF$
  Mise à jour de A^+.

D) Communications

Tant que le coordonnateur possède un pool non vide et qu'au moins un travailleur est inactif, il peut affecter des sous-problèmes. Nous avons dit qu'entre chaque affectation, il devait vérifier la réceptions de messages. Ce test de vérification doit être non bloquant de façon à retourner le plus rapidement à la prochaine affectation.

Par contre, lorsque tous les travailleurs sont actifs ($A^+ = T$) ou que le pool est vide et qu'un certain nombre de travailleurs sont actifs ($\Omega = \emptyset \ \wedge \ A^+ > 0$), il ne peut déclencher la stratégie d'affectation, le critère de déclenchement étant violé. Il doit cependant continuer de vérifier la réception de messages. Dans ce cas, le test de réception peut être bloquant.

4.4.3.3 Synopsis du coordonnateur

- *Initialisations*:

  $\Omega \leftarrow \emptyset$

  $\overline{Z^u} \leftarrow \infty$

  $A[j] \leftarrow INACTIF \ \forall j$

  $A^+ \leftarrow 0$

  $j^* \leftarrow 0$ (on suppose que les travailleurs sont numérotés de 0 à $T-1$).

- *Bloc principal*:

Tant que (critère d'arrêt non satisfait) faire
    Politique de déclenchement
    Si (déclenchement) alors
        Politique de sélection (choisir un sous-problème Q)
        Politique de localisation (localiser un travailleur T_j)
        Envoi spbm (Q) à T_j
        Réception non bloquante (spbms, borne, élag)
            Si (spbms) alors
                Insérer les spbms dans Ω
                Mettre à jour A^+
            Si (borne) alors
                Mettre à jour $\overline{Z^u}$ (si nécessaire)
            Si (élag) alors
                Mettre à jour A^+
    Sinon

Réception bloquante (spbms, borne, élag)

Si (spbms) alors

Insérer les spbms dans Ω

Mettre à jour A^+

Si (borne) alors

Mettre à jour $\overline{Z^u}$ (si nécessaire)

Si (élag) alors

Mettre à jour A^+

Diffusion de la fin

4.4.4 Code d'un travailleur

Dans l'approche centralisée, les rôles du travailleur comprennent l'examen des sous-problèmes et la gestion de la borne supérieure.

4.4.4.1 Données et paramètres

- Données:
  - $\overline{Z^u}$: meilleure borne supérieure.
- Paramètres:
  - T: nombre de travailleurs.

4.4.4.2 Bloc principal

A) Critère d'arrêt

Dès que le coordonnateur détecte la fin de l'exploration, il diffuse un message de fin. C'est la réception de ce message de fin qui met un terme aux activités du travailleur.

B) Examen

De l'algorithme de localisation multi-produits avec équilibrage, on tire les opérations suivantes avec les envois de messages qui s'y rattachent.

(*prétraitement*): Tenter de fixer certaines variables.
(*opération d'évaluation*): Effectuer la procédure d'évaluation sur Q (on évalue $Z^l(Q)$ et $Z^u(Q)$).
(*mise à jour de* $\overline{Z^u}$): Si ($Z^u < \overline{Z^u}$) alors $\overline{Z^u} \leftarrow Z^u(Q)$ et
diffuser $\overline{Z^u}$ aux autres processus.
(*règle d'élimination*)
Si ($Z^l(Q) > \overline{Z^u}$) alors Q est élagué.
Si (Q n'est pas élagué) alors
(*opération de séparation*): Choisir $j^* \in T_{01}$ et générer $Q_0^{j^*}$ et $Q_1^{j^*}$.

C) Communications

Un travailleur est toujours en attente bloquante de messages: soit il reçoit un sous-problème à examiner, soit une borne permettant de mettre $\overline{Z^u}$ à jour, ou un message de fin, lui indiquant que l'exploration est terminée.

4.4.4.3 Synopsis d'un travailleur

- *Initialisations*:
  $\overline{Z^u} \leftarrow \infty$
- *Bloc principal*:

Tant que (fin non reçu) faire
  Réception bloquante (spbm, borne, fin)
  Si (borne) alors
    Mettre à jour $\overline{Z^u}$
  Si (spbm) alors
    le message spbm devient le sous-problème Q
    Examen de Q

Diffusion borne ($\overline{Z^u}$ aux autres processus si nécessaire)
Si (spbm n'est pas élagué) alors
  Envoi spbms (sous-problèmes fils) au coordonnateur.
Sinon
  Envoi élag au coordonnateur

4.5 Approche décentralisée

L'approche décentralisée est basée sur une organisation de la mémoire à plusieurs pools. Contrairement à l'approche centralisée, chaque travailleur gère son propre pool de sous-problèmes, alors que le coordonnateur n'en gère aucun. Globalement, chaque travailleur dirige partiellement l'exploration de l'arborescence. Le rôle principal du coordonnateur consiste à balancer le travail sur l'ensemble des travailleurs de façon à maximiser le parallélisme. Le coordonnateur est ainsi responsable de la stratégie d'affectation.

4.5.1 Fonctionnement

Un des travailleurs possède le problème original dans son pool local et démarre l'exploration. Les autres travailleurs, ayant un pool vide, envoient une requête de travail au coordonnateur.

Si l'examen d'un sous-problème a permis sa séparation, un travailleur conserve un des sous-problèmes générés pour l'examiner et insère l'autre dans son pool. Par contre, si un sous-problème est élagué ou résolu, le travailleur sélectionne dans son pool le prochain sous-problème à examiner. À chaque fois qu'un travailleur fait des insertions ou des retraits dans son pool, il vérifie si sa charge satisfait une condition d'envoi, auquel cas il la communique au coordonnateur. De plus, lorsque cette charge devient inférieure à un seuil d'avertissement et lorsque celle-ci devient nulle, une requête de travail est envoyée au coordonna-

teur. Entre chaque examen de sous-problème, un travailleur vérifie la réception de messages: des bornes, des propositions d'affectation et aussi des sous-problèmes, si on permet à un travailleur d'envoyer des requêtes de travail avant que son pool local soit totalement vide.

Le coordonnateur reçoit et conserve en mémoire les charges et les requêtes communiquées par les travailleurs. Si la charge d'un travailleur dépasse un seuil fixé (le seuil d'affectation), il le considère hautement chargé. Lorsque le coordonnateur a une ou plusieurs requêtes à satisfaire et qu'il détecte qu'au moins un travailleur est hautement chargé, il déclenche la stratégie d'affectation.

Similairement à l'approche centralisée, les travailleurs conservent en mémoire la valeur de la meilleure solution réalisable connue. Si un travailleur évalue une meilleure borne supérieure, sa valeur remplace celle qu'il a en mémoire et il la diffuse aux autres travailleurs.

L'exécution se poursuit jusqu'à ce que l'issue de toutes les propositions d'affectation soit connue et que tous les travailleurs soient en attente de travail.

4.5.2 Messages

Dans cette approche, on retrouve les types de messages suivants:

- *bornes*: Les travailleurs diffusent aux autres processus travailleurs les meilleures bornes qu'ils évaluent.
- *charges* (charge): Périodiquement, un travailleur communique au coordonnateur la charge de son pool local. Ces charges sont utiles à la politique de localisation de la stratégie d'affectation.
- *requêtes* (req): Lorsque la charge locale du pool d'un travailleur baisse en deça d'un certain seuil ou lorsque le pool local d'un travailleur est épuisé, il envoie une requête de travail au coordonnateur.

- *propositions* (prop): Afin de satisfaire les requêtes de travail, le coordonnateur envoie des propositions d'affectation de sous-problèmes à des travailleurs hautement chargés.
- *sous-problèmes* (spbms): Lorsqu'un travailleur hautement chargé reçoit une proposition d'affectation, il sélectionne un certain nombre de sous-problèmes dans son pool local et les affecte à un travailleur requéreur spécifié par la proposition.
- *réponses négatives* (rép): Un travailleur a reçu une proposition d'affectation mais sa charge locale est insuffisante pour la satisfaire, il envoie alors une réponse négative au travailleur requéreur spécifié par la proposition.
- *confirmations* (conf): Dans le cas où un travailleur requéreur reçoit des sous- problèmes ou une réponse négative d'un autre travailleur, il envoie une confirmation d'affectation informant le coordonnateur sur l'issue de sa proposition.
- *fin*: Lorsque le critère d'arrêt de l'exploration est satisfait, le coordonnateur diffuse un message de fin aux travailleurs.

La figure 13 illustre les types de messages échangés entre les processus dans l'approche décentralisée.

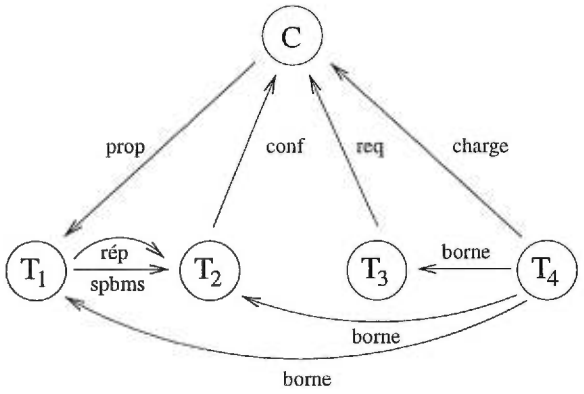

FIGURE 13: Approche décentralisée: types de messages

4.5.3 Stratégie d'affectation et paramètres

La stratégie d'affectation intégrée à l'approche décentralisée comporte un bon nombre de paramètres permettant de la contrôler. Nous consacrons cette sous-section à leur description avant de décrire en détails les codes du coordonnateur et d'un travailleur.

Nous rappelons que la stratégie d'affectation exploitée ici est dynamique avec requêtes à l'initiative du receveur. Pendant la résolution, le coordonnateur reçoit périodiquement des travailleurs deux types d'informations: des requêtes de travail et la charge de leur pool local. La charge d'un pool correspond à une mesure relative du travail contenu dans ce pool. Dans notre implantation de l'approche décentralisée, la charge d'un pool correspond au nombre de sous-problèmes contenus dans ce pool. À partir de ces deux types d'informations, le coordonnateur tente de satisfaire les requêtes de travail en proposant des mouvements de sous-problèmes à partir de travailleurs hautement chargés vers des travailleurs faiblement chargés. Le but de tels mouvements permet de répartir le travail afin de maintenir les travailleurs actifs.

Les paramètres associés à cette approche permettent de contrôler cette stratégie d'affectation:

- L_s: paramètre permettant de distinguer les travailleurs hautement chargés des travailleurs faiblement chargés. Lorsque la charge d'un pool local du travailleur i est supérieure à L_s ($L(\Omega_i) \geq L_s$), le travailleur i est considéré hautement chargé. Dans la figure 14, les travailleurs 1, 2 et 4 sont hautement chargés.
- L_a: paramètre permettant de contrôler l'envoi de requêtes. Bien qu'un travailleur envoie systématiquement une requête de travail lorsque son pool est vide, un travailleur a également la possibilité d'envoyer des requêtes anticipées, permettant au coordonnateur de déclencher le processus d'affectation avant même que le travailleur ait épuisé son pool. Lorsque la charge

du pool local du travailleur i passe à une valeur inférieure ou égale à L_a, il envoit une requête de travail ($L(\Omega_i) \leq L_a$). Dans la figure 14, le processeur 5 aurait déjà envoyé une requête de travail.

- *Loc*: paramètre permettant de choisir la politique de localisation. Comme nous l'avons mentionné au chapitre précédent, nous avons proposé deux politiques de localisation de travailleurs donneurs:
  - Politique H: consiste à localiser le travailleur le plus chargé de l'ensemble de ceux qui sont hautement chargés. Dans notre exemple, le coordonnateur localiserait le travailleur 4.
  - Politique R: selon cette politique, le coordonnateur procède par cyclage sur l'ensemble des travailleurs hautement chargés afin de localiser un donneur. Dans notre exemple, en supposant que le dernier donneur localisé par le coordonnateur était le travailleur 1, le prochain donneur localisé serait le travailleur 2.
- ΔL: paramètre permettant de contrôler l'envoi de charges lorsque la politique de localisation H est utilisée. Lorsque la charge du pool d'un travailleur i varie, à la hausse ou à la baisse, d'une valeur supérieure ou égale à ΔL par rapport à la dernière valeur de charge envoyée par ce même travailleur au coordonnateur, le travailleur i envoie sa charge locale. On rappelle que lorsque la politique de localisation R est utilisée, un travailleur envoie sa charge uniquement lorsque la dernière envoyée au coordonnateur l'incluait parmi les travailleurs hautement chargés, alors que la valeur courante de sa charge l'inclut parmi les travailleurs faiblement chargés (et inversement).
- L_{min}: paramètre permettant de contrôler le seuil d'affectation. Lorsqu'un travailleur reçoit une proposition d'affectation, il vérifie s'il possède au moins la valeur de charge minimale pour effectuer une affectation.
- N: paramètre permettant de contrôler le nombre de sous-problèmes transférés lors d'une affectation. Lorsqu'un travailleur a reçu une proposition d'affectation et qu'il a vérifié qu'il possède bien la charge minimale pour effectuer une affectation, il tente de sélectionner N sous-

problèmes sans violer le seuil de charge minimum; le travailleur sélectionne $\min(L_c - L_{min} + 1, N)$ sous-problèmes où L_c est la charge courante du pool;

La figure 14 illustre un exemple de configuration de pools ainsi que les seuils de charge importants.

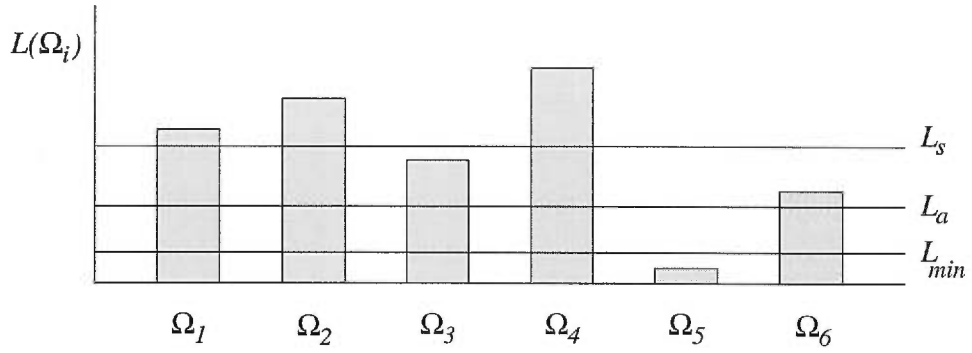

FIGURE 14: Approche décentralisée: seuils de la stratégie d'affectation

Il est important de noter que l'ajustement de la stratégie d'affectation à l'aide des paramètres implique un impact sur la répartition du travail ainsi que sur l'ordre d'exploration des sous-problèmes.

4.5.4 Code du coordonnateur

Les rôles du coordonnateur comprennent l'application d'une partie de la stratégie d'affectation et la détection de la fin.

4.5.4.1 Données et paramètres

- Données:
  - $L[j]$: charge du travailleur j;
  - j_{max}: numéro du travailleur le plus chargé;
  - $P[j]$: proposition d'affectation pour le travailleur requéreur j;
  - P^+: nombre de propositions envoyées et non confirmées;

- R: file de requêtes des travailleurs.

- Paramètres:
  - T: nombre de travailleurs;
  - L_s: seuil de charge.

4.5.4.2 Bloc principal

A) Critère d'arrêt

Comme nous le mentionnions au chapitre 2, la fin de l'exploration ne peut être décrétée tant que des messages sont en circulation entre les processus. Ce sont les messages nécessaires à la stratégie d'affectation qui permettent au coordonnateur de détecter la fin. En effet, le coordonnateur possède une certaine connaissance de l'état d'activité des travailleurs par la communication des charges et des requêtes, mais en raison des délais de transfert de messages et des durées d'examen des sous-problèmes, cette connaissance est approximative. Afin d'éviter des situations où, en raison de cette mauvaise connaissance, la fin de l'exploration soit décrétée trop tôt, le coordonnateur s'assure que toutes les propositions d'affectation ont pris fin par la réception des confirmations et qu'il a reçu des requêtes de travail de tous les travailleurs. En termes des données introduites plus tôt, cette condition s'exprime comme suit:

$$|R| = T \ \wedge \ P^{+} = 0.$$

B) Stratégie d'affectation

Le coordonnateur sélectionne une requête, localise un travailleur donneur, appartenant au groupe des travailleurs hautement chargés (politique de localisation), et lui envoie une proposition d'affectation. À la réception de la proposition, le travailleur vérifie d'abord s'il est toujours hautement chargé en comparant sa charge courante au seuil d'affectation (L_{min}). Si tel est le cas, il sélectionne les sous-problèmes à affecter (politique de sélection) et les envoie au travailleur requéreur spécifié par la requête. Sinon, il envoie une réponse négative au tra-

vailleur requéreur. En recevant des sous-problèmes, un travailleur requéreur les insère dans son pool local et envoie une confirmation au coordonnateur lui indiquant la réussite de la proposition. En recevant une réponse négative, un travailleur requéreur envoie une confirmation au coordonnateur lui indiquant l'échec de la proposition et refait une requête de travail. Entre chaque envoi de proposition, le coordonnateur vérifie l'arrivée de messages: des charges, des requêtes et des confirmations.

- Politique de déclenchement: Pour déclencher l'envoi d'une proposition d'affectation, le coordonnateur doit posséder en mémoire une ou plusieurs requêtes à satisfaire et connaître l'existence d'un travailleur hautement chargé. En termes des données définies ci-dessus, le critère de déclenchement s'exprime comme suit:

$$(L[j_{max}] > L_s) \ \wedge \ (R \neq \emptyset).$$

- Politique de localisation: On décrit deux politiques possibles pour le choix d'un travailleur hautement chargé.
  - Politique 1: On choisit le travailleur le plus chargé comme travailleur donneur. Par la politique de déclenchement, on sait que ce travailleur fait partie des travailleurs hautement chargés. Les étapes de la localisation sont:

    Sélectionner une requête de R (requête d'un travailleur k)
    $j^* \leftarrow j_{max}$
    $P[k] \leftarrow VRAI$
    Envoyer une proposition au travailleur j^* pour le travailleur k

  - Politique 2: On utilise une méthode par cyclage pour localiser un travailleur donneur. On balaie en ordre numérique les travailleurs en ne considérant que les travailleurs hautement chargés. Les étapes de la localisation sont:

Sélectionner une requête de R (requête d'un travailleur k)

$j^* \leftarrow (j^* + 1) \bmod T$

Tant que $(L[j^*] < L_s)$ faire

$j^* \leftarrow j^* + 1 \bmod T$

$P[k] \leftarrow VRAI$

Envoyer une proposition au travailleur j^* pour le travailleur k

C) Communications

Comme dans l'approche centralisée, où le coordonnateur doit surveiller l'arrivée de messages entre chaque affectation, le coordonnateur de l'approche décentralisée surveille l'arrivée de messages à l'aide d'un test non bloquant entre chaque envoi de proposition.

Si le coordonnateur détecte qu'il ne peut envoyer de proposition, soit parce que l'ensemble de requêtes est vide, soit parce qu'aucun travailleur n'est hautement chargé, il n'a d'autre choix que de se placer en position d'attente de messages; il fait alors usage d'un test de réception bloquant.

4.5.4.3 Synopsis du coordonnateur

- *Initialisations*:

  $L[j] \leftarrow 0 \ \forall j$;

  $L[j^*] \leftarrow 1$ où j^* est le travailleur qui débute l'exploration;

  $j_{max} \leftarrow j^*$;

  $P[j] \leftarrow FAUX \ \forall j$;

  $R \leftarrow \emptyset$.

- *Bloc principal*:

Tant que (critère d'arrêt non satisfait) faire

Politique de déclenchement

Si (déclenchement) alors

Réception non bloquante (charge, req, rep)

Politique de localisation (choisir T_j)

Envoi de proposition à T_j

Sinon

Réception bloquante (charge, req, rep)

Diffusion de la fin

4.5.5 Code d'un travailleur

Les rôles d'un travailleur comprennent la gestion du pool local, l'examen de sous-problèmes, la participation à la stratégie d'affectation et la gestion de la borne supérieure.

4.5.5.1 Données et paramètres

- Données:
  - Ω_j: pool local (travailleur j);
  - Q^*: problème original;
  - $\overline{Z^u}$: meilleure borne supérieure.
- Paramètres:
  - T: nombre de travailleurs;
  - L_s: seuil de charge;
  - L_{min}: seuil d'affectation;
  - L_a: seuil d'avertissement;
  - N: nombre de sous-problèmes affectés à la fois;
  - ΔL: filtre d'envoi de charge pour la politique 1.

4.5.5.2 Bloc principal

A) Critère d'arrêt

Dès que le coordonnateur détecte la fin de l'exploration, il diffuse un message de

fin. C'est la réception de ce message de fin qui met un terme aux activités du travailleur.

B) Gestion du pool

Le pool local (Ω_j) est géré à l'aide d'une structure de pile comme dans l'algorithme séquentiel initial. L'utilisation d'une pile permet de respecter l'ordre d'exploration "profondeur d'abord".

À chaque opération effectuée sur la pile (insertion ou retrait), le travailleur évalue la charge locale (qui correspond dans notre cas à la taille de la pile), vérifie une condition d'envoi de charge adaptée à la politique de localisation utilisée par le coordonnateur. Selon la politique 1, la condition est:

$$|L_c - L_p| > \Delta L$$

où L_p est la dernière valeur de charge envoyée au coordonnateur. Selon la politique 2, la condition est:

$$((L_c > L_s) \ \wedge \ (L_p \leq L_s)) \vee ((L_c \leq L_s) \ \wedge (L_p > L_s)).$$

C) Examen

L'examen est identique à celui implanté dans la stratégie centralisée (voir section 4.4.4.2).

D) Stratégie d'affectation

La définition de charge que nous utilisons pour fournir une mesure relative de la quantité de travail associée au pool local Ω_j est la suivante:

$$L(\Omega_j) = m$$

où m correspond au nombre de sous-problèmes stockés dans le pool.

À la réception d'une proposition, un travailleur vérifie si sa charge locale (L_c) est assez élevée pour affecter une partie de ses sous-problèmes, en appliquant le

test $L_c > L_{min}$. Si le test n'est pas vérifié, il envoie une réponse négative au travailleur requéreur. Sinon, il fera une sélection de sous-problèmes dans le pool local: la priorité de sélection est la même que celle utilisée pour le choix des sous-problèmes examinés localement. Le nombre de sous-problèmes sélectionnés à la fois est contrôlé par le paramètre N: le travailleur retire de la pile $\min\{L_c - L_{min} + 1, N\}$ sous-problèmes.

E) Communications

Tant que le travailleur possède un pool non vide, il doit examiner les sous-problèmes qui y sont stockés. Entre ces examens, il doit surveiller l'arrivée de messages en utilisant un test de réception non bloquant afin de retourner rapidement au prochain examen si aucun message ne lui est parvenu.

Par contre, si son pool est vide, il est alors en attente de travail, et ne peut faire autrement que se placer en position de recevoir de nouveaux messages. Il utilise alors un test bloquant de messages.

4.5.5.3 Synopsis d'un travailleur

- *Initialisations*:

  Q^*: problème original;

  $\Omega_j \leftarrow \emptyset$;

  $\Omega_j \leftarrow \{Q^*\}$ si $j = j^*$ (où j^* est le numéro du travailleur qui débute l'exploration);

  $\overline{Z^u} \leftarrow \infty$.

- *Bloc principal*:

Tant que (message fin non reçu) faire

  Tant que ($\Omega_j \neq \emptyset$) faire

    Si (Q est élagué) alors

      Sélectionner Q dans Ω_j

      Appliquer règle de la borne inférieure

Si (Q n'est pas élagué) alors

Examen de Q

Diffusion de borne ($\overline{Z^u}$) si nécessaire

Réception non bloquante (prop, borne)

Si (prop (pour T_i)) alors

Politique de sélection

Si (spbms sélectionnés) alors

Envoi spbms à T_i

Sinon

Envoi rép négative à T_i

Si (borne) alors

Mise à jour de $\overline{Z^u}$ (si nécessaire)

Tant que (($\Omega_j = \emptyset$) $\wedge$ (message fin non reçu)) faire

Envoi req au coordonnateur

Répéter

Réception bloquante (spbms, rep, prop, borne, fin)

Si (spbms) alors

Insérer spbms dans Ω_j

Si (rép) alors

Envoi réponse négative au coordonnateur

Si (prop (pour T_i)) alors

Envoi rép (négative) à T_i.

Si (borne) alors

Mise à jour de $\overline{Z^u}$ (si nécessaire)

Tant que (on reçoit message borne)

4.6 Approche hybride

L'approche hybride est basée sur une organisation de la mémoire qui recoupe les deux précédentes: un pool global accessible par tous les processus et un pool

local à chacun des travailleurs. Comme dans l'approche décentralisée, chacun des travailleurs est responsable d'une partie de l'exploration de l'arborescence. Les travailleurs affectent une partie des sous-problèmes qu'ils génèrent au coordonnateur. Ce dernier est responsable de réaffecter ces sous-problèmes aux travailleurs qui font des requêtes. Bien que le rôle du coordonnateur consiste à balancer le travail en affectant des sous-problèmes aux travailleurs requéreurs, il participe activement au processus de construction de l'arborescence en choisissant "intelligemment" les sous-problèmes qu'il affecte. Par conséquent, le coordonnateur et les travailleurs sont responsables de la stratégie d'affectation et participent activement à l'exploration de l'arborescence.

4.6.1 Fonctionnement

Comme dans l'approche décentralisée, tous les travailleurs font une requête de travail au coordonnateur sauf un qui possède le problème original dans son pool et qui débute l'exploration.

Si l'examen d'un sous-problème a permis sa séparation, un travailleur conserve un des sous-problèmes générés et affecte l'autre au coordonnateur (ce dernier sous-problème correspond à ce qu'on a appelé un sous-problème de "deuxième qualité"). Cette simple affectation constitue la phase travailleur-coordonnateur. Par contre, si un sous-problème examiné est élagué ou résolu, il envoie une requête de travail au coordonnateur.

Le coordonnateur insère dans son pool les sous-problèmes qu'il reçoit. Il conserve également en mémoire les requêtes des travailleurs. Lorsque le coordonnateur a une ou plusieurs requêtes à satisfaire et que son pool est non vide, il déclenche la phase coordonnateur-travailleur.

Le coordonnateur et les travailleurs conservent en mémoire la valeur de la meilleure solution réalisable connue. Si un travailleur évalue une meilleure borne, sa valeur remplace celle qu'il a en mémoire et il la diffuse au coordonnateur et

aux autres travailleurs.

Lorsque le pool central est vide et que tous les travailleurs sont en attente de travail, le coordonnateur détecte la fin de l'exploration et diffuse un message de fin aux travailleurs.

4.6.2 Messages

Dans cette approche, on retrouve les types de messages suivants:

- *requêtes* (req): Lorsqu'un travailleur détecte que son pool local est vide, il envoie une requête de travail au coordonnateur.
- *sous-problèmes* (spbms): D'une part, un travailleur affecte régulièrement des sous-problèmes de "deuxième qualité" au coordonnateur. D'autre part, le coordonnateur affecte aux travailleurs requéreurs des sous-problèmes qu'il sélectionne dans le pool central.
- *bornes* (borne): Les travailleurs diffusent aux autres processus les meilleures bornes qu'ils évaluent.
- *fin*: Lorsque le critère d'arrêt de l'exploration est satisfait, le coordonnateur diffuse un message de fin aux travailleurs.

La figure 15 illustre les types de messages échangés entre les processus dans l'approche hybride.

4.6.3 Code du coordonnateur

Les rôles du coordonnateur comprennent la gestion du pool, l'application de la phase coordonnateur-travailleur de la stratégie d'affectation et la détection de la fin de l'exploration.

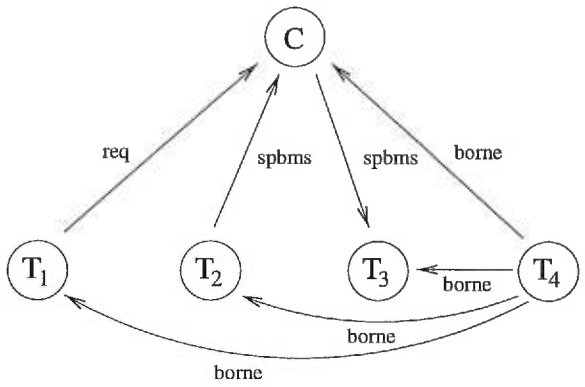

FIGURE 15: Approche hybride: types de messages

4.6.3.1 Données et paramètres

- Données:
  - Ω: pool central;
  - $\overline{Z^u}$: meilleure borne supérieure;
  - R: file de requêtes.
- Paramètres:
  - T: nombre de travailleurs;
  - p_1, p_2, p_3: pondération des critères de la règle de sélection.

4.6.3.2 Bloc principal

A) Critère d'arrêt

Similairement à l'approche décentralisée, le coordonnateur utilise des requêtes de travail pour contrôler sa stratégie d'affectation. Lorsqu'il reçoit une requête d'un travailleur j, il sait que ce dernier est inactif et en attente de travail. Étant donné que le coordonnateur gère un pool, il ne peut pas mettre fin à l'exploration tant qu'il n'a pas affecté tous ses sous-problèmes. Par conséquent, à partir des données définies, le critère d'arrêt s'exprime comme suit:

$$|R| = T \ \wedge \ \Omega = \emptyset.$$

B) Gestion du pool

La gestion du pool central de l'approche hybride est identique à celle de l'approche centralisée. À la réception de sous-problèmes, le coordonnateur évalue leur priorité et les insère dans le pool. Afin de faciliter l'insertion et la sélection des sous-problèmes, le pool possède une structure de *monceau*. À la sélection d'un problème du pool, le coordonnateur tente de l'éliminer en appliquant le test de la borne inférieure. Le coordonnateur utilise la valeur $\overline{Z^u}$ pour appliquer ce test. Étant donné que le pool est constitué de sous-problèmes non évalués, la borne inférieure du sous-problème parent est utilisée ($Z^l(Q^p) \geq \overline{Z^u}$).

C) Stratégie d'affectation: phase coordonnateur-travailleur

Le coordonnateur sélectionne un sous-problème (politique de sélection), localise un travailleur receveur (politique de localisation) en sélectionnant une requête de R et affecte le sous-problème au travailleur.

- Politique de déclenchement: Lorsque le pool central est non vide et que le coordonnateur a au moins une requête de travail à satisfaire, il déclenche la stratégie d'affectation. En terme des données définies, le critère de déclenchement s'exprime comme suit:

$$R \neq \emptyset \ \wedge \ \Omega \neq \emptyset.$$

- Politique de sélection: Elle est identique à celle de la stratégie centralisée (voir section 4.4.3.2)
- Politique de localisation: Le travailleur receveur est déterminé par la requête sélectionnée.

D) Communications

Si le coordonnateur gère un pool non vide et qu'il possède en mémoire au moins une requête à satisfaire, il peut affecter des sous-problèmes. Nous avons dit

qu'entre chaque affectation, il devait vérifier la réception de messages. Ce test de vérification doit être non bloquant de façon à retourner le plus rapidement à la prochaine affectation.

Par contre, lorsqu'il n'a aucune requête à satisfaire et que son pool est non vide, il ne peut déclencher la stratégie d'affectation, le critère de déclenchement étant violé. Il doit cependant continuer de vérifier la réception de messages. Dans ce cas, le test de réception peut être bloquant.

4.6.3.3 Synopsis du coordonnateur

- *Initialisations*:

  $\Omega \leftarrow \emptyset$;

  $\overline{Z^u} \leftarrow \infty$;

  $R \leftarrow \emptyset$.

- *Bloc principal*:

Tant que (critère d'arrêt non satisfait) faire

  Politique de déclenchement (phase coordonnateur-travailleur)

  Si (déclenchement) alors

    Politique de localisation (T_j)

    Politique de sélection (sélectionner spbms)

    Envoi spbms à T_j

    Réception non bloquante (spbms, req, borne)

  Sinon

    Réception bloquante (spbms, req, borne)

Diffusion de la fin

4.6.4 Code d'un travailleur

Les rôles d'un travailleur comprennent la gestion d'un pool, l'examen de sous-problèmes, l'application de la phase travailleur-coordonnateur de la stratégie

d'affectation et la gestion de la borne supérieure.

4.6.4.1 Données et paramètres

- Données:
  - Q^*: problème original;
  - Ω_j: pool local (travailleur j);
  - $\overline{Z^u}$: meilleure borne supérieure.
- Paramètres:
  - T: nombre de travailleurs.

4.6.4.2 Bloc principal

A) Critère d'arrêt

Dès que le coordonnateur détecte la fin de l'exploration, il diffuse un message de fin. C'est la réception de ce message de fin qui met un terme aux activités du travailleur.

B) Gestion du pool

Comme dans l'approche décentralisée, le pool local Ω_j est géré à l'aide d'une structure de pile.

C) Examen

L'examen est identique à celui implanté dans la stratégie centralisée (voir section 4.4.4.2).

D) Stratégie d'affectation: phase travailleur-coordonnateur

- Politique de déclenchement: L'opération de séparation du travailleur précède chaque envoi de sous-problème au coordonnateur.
- Politique de sélection: Après une opération de séparation, un travailleur sélectionne le sous-problème complémentaire à celui conservé pour examen

comme sous-problème de "deuxième qualité".

- Politique de localisation: Pour la phase travailleur-coordonnateur, la politique de localisation est triviale: tous les sous-problèmes sont affectés au coordonnateur.

E) Communications

Dans notre implantation actuelle de l'approche hybride, un travailleur n'utilise pas son pool local. Comme nous l'avons vu, après chaque application de l'opération de séparation, un travailleur n'insère jamais de sous-problèmes dans son pool local: soit il en conserve un pour examen, soit il envoie une requête de travail au coordonnateur. En effet, on rappelle que pour le problème de localisation multi-produits avec équilibrage, la règle de séparation est dichotomique (Gendron et Crainic 1995).

Tant que le travailleur arrive à séparer un sous-problème, il examine des sous-problèmes. Entre chaque examen, il doit surveiller l'arrivée de messages en utilisant un test de réception non bloquant afin de retourner rapidement au prochain examen si aucun message ne lui est parvenu.

Par contre, après avoir envoyé une requête, il n'a d'autre choix que de se placer en position de recevoir des messages. Il utilise alors un test bloquant de messages.

4.6.4.3 Synopsis d'un travailleur

- *Initialisations*:

  $\Omega_j \leftarrow \emptyset$;

  $\Omega_j \leftarrow Q^*$ si $j = j^*$ (où j^* est le numéro du travailleur qui débute l'exploration);

  $\overline{Z^u} \leftarrow \infty$.

- *Bloc principal*:

Tant que (message fin non reçu) faire

Tant que ($\Omega_j \neq \emptyset$) faire

Si (Q est élagué) alors

Sélectionner Q dans Ω_j

Appliquer règle de la borne inférieure

Si (Q n'est pas élagué) alors

Examen de Q

Diffusion de $\overline{Z^u}$ (si nécessaire)

Politique de déclenchement (phase travailleur-coordonnateur)

Si (déclenchement) alors

Politique de sélection

Politique de localisation

Envoi de spbm au coordonnateur

Réception non bloquante (borne)

Si (borne) alors

Mise à jour de $\overline{Z^u}$ (si nécessaire)

Envoi req au coordonnateur

Réception bloquante (spbms, borne, fin)

Si (spbms) alors

Insérer spbms dans Ω_j

Si (borne) alors

Mise à jour de $\overline{Z^u}$ (si nécessaire)

CHAPITRE 5

Expérimentations numériques et analyses

Ce chapitre présente un certain nombre d'expérimentations numériques et d'analyses de performance des implantations parallèles décrites au chapitre précédent. Ces implantations sont appliquées à la résolution d'exemplaires du problème de localisation multi-produits avec équilibrage. Ces expérimentations visent deux objectifs principaux:

- pour chacune des approches de parallélisation, estimer l'impact de différentes stratégies d'affectation, en mesurant les performances en fonction du nombre de processeurs;
- pour des problèmes aux caractéristiques différentes, en termes de granularité de calcul à chacun des sous-problèmes et de taille des arborescences générées, qualifier les performances relatives des trois approches de parallélisation.

La première section de ce chapitre décrit les jeux de données utilisés pour nos expérimentations. La seconde section décrit l'environnement dans lequel les expérimentations ont été réalisées. Afin de comparer entre elles les parallélisations, de même qu'avec l'algorithme séquentiel, la troisième section dresse une liste des mesures de performances nécessaires à nos analyses. Les trois sections suivantes exposent les résultats obtenus avec chacune des trois approches de parallélisation. La septième section compare ces trois approches entre elles. Enfin, la dernière section expose quelques conclusions tirées de l'ensemble des expérimentations.

5.1 Jeux de données

Afin de tester nos parallélisations, quelques jeux de données possédant des caractéristiques variées ont été générés aléatoirement. Également, un jeu de données provenant d'une application réelle de gestion de conteneurs a également été utilisé. À ceux-ci s'ajoute un jeu de données généré à partir de l'application réelle en faisant varier aléatoirement les coûts fixes ainsi que les coûts de transport des produits sur les arcs.

Chaque jeu de données contient la représentation d'un réseau composé de certaines entités. D'abord, trois ensembles de noeuds sont définis: les noeuds clients origines (O), les noeuds clients destinations (D) et les noeuds dépôts (T). Implicitement, trois ensembles d'arcs permettent de relier ces noeuds: les arcs client-dépôt (A_{OT}), les arcs dépôt-client (A_{TD}) et les arcs dépôt-dépôt (A_{TT}). On retrouve également un ensemble de produits pouvant circuler dans le réseau (P) pour lesquels des offres et des demandes sont associées aux noeuds clients. De plus, des coûts fixes d'ouverture sont associés à chacun des dépôts et des coûts de transport unitaire par arc et par produit sont définis.

Dans la suite de ce chapitre, nous définissons la taille (n) d'un problème égale au nombre de sous-problèmes examinés par l'algorithme séquentiel de référence (voir section 2.1) pour le résoudre: n = nombre de sous-problèmes examinés par l'algorithme séquentiel. Le tableau 1 dresse une liste des jeux de données. Chacun d'eux est caractérisé par la cardinalité de ses entités, sa taille (n), ainsi que le temps de résolution séquentiel (en secondes) requis par l'algorithme de référence pour le résoudre ($T_s(n)$).

Afin de comparer les versions dans un cadre quelque peu différent, nous avons résolu le problème $P1$ à l'aide d'une procédure d'évaluation basée sur une relaxation faible du problème de localisation multi-produits avec équilibrage (Crainic, Dejax et Delorme 1993). Ainsi, la granularité de calcul des sous-problèmes diminue considérablement et la qualité des bornes est inférieure. Par conséquent,

les arborescences générées en utilisant une telle procédure d'évaluation sont de très grande taille. Ceci explique pourquoi la taille du problème $P1$ est si élevée (tableau 1).

Prob	$\|P\|$	$\|O\|$	$\|D\|$	$\|T\|$	$\|A_{OT}\|$	$\|A_{TD}\|$	$\|A_{TT}\|$	n	$T_s(n)$
$P1$	3	124	124	26	871	871	650	300563	19141
$P2$	1	219	219	44	2630	2630	1892	221	185
$P3$	2	219	219	44	2629	2629	1892	565	281
$P4$	2	219	219	44	2629	2629	1892	145	339
$P5$	1	219	219	44	2631	2631	1892	145	38
$P6$	2	220	220	43	2647	2647	1806	247	125
$P7$	2	220	220	43	2647	2647	1806	99	121
$P8$	12	289	289	130	1914	1914	890	3681	3013
$P9$	12	289	289	130	1914	1914	890	10825	8146

TABLEAU 1: Dimensions des problèmes

5.2 Environnement

Un environnement parallèle à mémoire distribuée composé de 16 stations de travail de type Sparc Ultra 1/140, disposant chacune de 64 méga-octets de mémoire vive, reliées par la technologie Fast-Ethernet (100 Mbits/s) a été utilisé pour les expérimentations. Ces dernières se sont déroulées sur un réseau entièrement dédié à la résolution des problèmes, sans être gênées par l'exécution de processus externes. De plus, les processus coordonnateur et travailleurs ont été disposés sur le réseau de façon à ce que le coordonnateur (C) et un des travailleurs (T_1) soient exécutés sur une seule station. La figure 16 illustre l'organisation des processus sur le réseau de stations de travail.

Les procédures d'évaluation et de séparation de l'algorithme de SÉP sont en langage Fortran, compilées à l'aide du compilateur *f77* en utilisant l'option

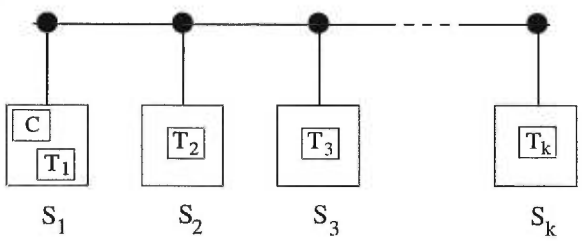

FIGURE 16: Position des processus sur le réseau de stations

d'optimisation $-O4$. Le code parallèle est écrit en langage C et compilé à l'aide du compilateur *gcc* en utilisant l'option d'optimisation $-O3$. La bibliothèque de fonctions PVM assure l'échange de messages entre les processus démarrés sur le réseau de stations.

5.3 Mesures de performance

Aux sous-sections 1.2.3 et 1.3.3, nous avons mentionné un certain nombre de mesures de performance des algorithmes parallèles et plus particulièrement des parallélisations de type 2 de l'algorithme de SÉP. Afin de décrire le comportement et les performances de nos parallélisations, nous en avons sélectionné quelques-unes.

Afin d'avoir un indice de la qualité du parcours de la recherche, nous utilisons le *facteur de pénalité de recherche* ($P(n,p)$) qui est défini par le rapport du nombre de sous-problèmes évalués par une parallélisation sur le nombre de ceux explorés par l'algorithme séquentiel de référence décrit à la section 2.1. Formellement, le facteur de pénalité s'exprime comme suit:

$$P(n,p) \;=\; N(n,p)/n.$$

En vue de mesurer la qualité de la répartition du travail sur les travailleurs, nous utilisons le *facteur d'équilibrage de travail* ($F^{t}(n,p)$) qui est défini par le rapport du temps d'utilisation minimum sur le temps d'utilisation maximum de

l'ensemble des processus:

$$F^t(n,p) \;=\; \min_{1\leq j\leq p} U_j^t(n,p) / \max_{1\leq j\leq p} U_j^t(n,p)$$

(on rappelle que $U_j^t(n,p) = W_j^t(n,p)/T(n,p)$ où $W_j^t(n,p)$ est le travail du processus j).

Bien que le facteur d'équilibrage décrit bien la répartition du travail sur les travailleurs, il n'indique pas si ces derniers ont été bien utilisés. Afin de pallier ce manque, nous ajoutons une autre mesure qui constitue une mesure en pire cas de l'utilisation des travailleurs. L'*utilisation minimum* ($U_{min}^t(n,p)$) est définie comme suit:

$$U_{min}^t(n,p) \;=\; \min_{1\leq j\leq p} U_j^t(n,p).$$

Cette mesure permet principalement de détecter les situations où une parallélisation répartit très bien le travail ($F^t(n,p) \cong 1$) mais qui, par contre, utilise très mal les processus ($U_{min}^t \ll 1$).

Enfin, nous utilisons également l'*accélération* ($S(n,p)$), mesure largement utilisée, qui permet d'avoir une idée globale de la performance d'une parallélisation. L'accélération est définie par le rapport du temps séquentiel de résolution de l'algorithme de référence sur celui d'une parallélisation:

$$S(n,p) \;=\; T_s(n)/T(n,p).$$

Dans le but de diminuer les pénalités de recherche, rappelons que chacune des parallélisations entreprend son traitement par l'application de la stratégie d'affectation initiale qui consiste à affecter la racine à un des travailleurs afin qu'il effectue une recherche en profondeur jusqu'à l'obtention d'une feuille. La stratégie d'affectation initiale est donc purement séquentielle, ce qui nous permet de valider les valeurs d'accélération obtenues en calculant l'accélération limite selon la loi d'Amdahl (Amdahl 1967), en spécifiant la fraction sérielle ainsi que

le nombre de processeurs utilisés. Dans notre cas, la fraction sérielle est définie par le rapport du temps entre le début de l'exécution et la réception d'un sous-problème par un travailleur autre que celui qui effectue l'exploration initiale ($\tilde{T}$), sur le temps séquentiel ($T_s(n)$) requis pour la résolution du problème concerné. L'accélération limite s'exprime alors de la façon suivante:

$$S_A(n,p) = p\left(\frac{T_s(n)}{T_s(n) + (p-1)\tilde{T}}\right).$$

La valeur donnée par cette expression est une limite théorique sur l'accélération possible à atteindre sans tenir compte des surcoûts impliqués par la parallélisation.

En plus de ces mesures, nous nous intéressons à estimer, pour chacun des problèmes considérés, une accélération limite de nos parallélisations sans spécifier le nombre de processeurs utilisés. Nous estimons ces accélérations selon trois méthodes (Chabini et Gendron 1995). Les deux premières découlent de la notion de surcoût de parallélisation relatif, alors que la dernière découle de la loi d'Amdahl.

Considérant les résultats obtenus suite à la résolution d'un problème spécifique à l'aide d'une parallélisation, en faisant varier à chaque fois le nombre de processeurs, il est possible de tracer la courbe du surcoût de parallélisation relatif. On sait que la courbe de surcoût de parallélisation capture tous les types de surcoût d'une parallélisation. Un avantage du surcoût de parallélisation relatif est qu'il permet de prédire, en considérant la pente en n'importe quel point de la courbe, le nombre de processeurs pour lequel le temps est minimum (si un tel minimum existe). À la section 1.2.3, on avait exprimé le temps $T(n,p)$ en fonction du surcoût de parallélisation: $T(n,p) = T_s(n)(1/p + B(n,p))$.

La première méthode consiste à considérer la dérivée partielle de $T(n,p)$ par rapport à p et de considérer la valeur de p pour laquelle le temps est minimum:

$$\frac{\partial T(n,p)}{\partial p} = T_s(n)\left(-\frac{1}{p^2} + \frac{\partial B(n,p)}{\partial p}\right)$$

et, par conséquent, si $\tilde{p} > 1$ minimise le temps, alors

$$\left.\frac{\partial B(n,p)}{\partial p}\right|_{p=\tilde{p}} = \frac{1}{\tilde{p}^2}.$$

Sachant qu'avec $\tilde{p}$ processeurs, on arrive à la parallélisation la moins coûteuse, et considérant que la parallélisation est idéale, on déduit qu'une borne sur l'accélération atteignable est donnée par $\tilde{p}$.

La deuxième méthode est déterminée en exprimant l'accélération en fonction du surcoût de parallélisation. En reconsidérant l'expression de $T(n,p)$ en fonction du surcoût, on arrive à: $S(n,p) = 1/(1/p + B(n,p))$. Ainsi, on peut déduire l'inégalité suivante, en supposant que $B(n,p) > 0$:

$$S(n,p) \leq 1/B(n,p).$$

En prenant le minimum de $B(n,p)$, on obtient une borne supérieure sur l'accélération. Il est important de noter que le minimum de la courbe de surcoût de parallélisation ne peut être déduit que lorsqu'on considère toutes les valeurs de p. Toutefois, en considérant la courbe partielle dont on dispose, on peut déduire deux situations réalistes: soit la courbe poursuit sa décroissance avant d'atteindre son minimum, soit la dernière valeur obtenue constitue le minimum, après laquelle la courbe deviendra croissante. Dans le premier cas, aucune estimation ne peut être fournie par la première méthode et la valeur fournie par la deuxième méthode sous-estime l'accélération limite. Dans le second cas, une estimation peut être fournie par la première méthode et la valeur fournie par la seconde méthode constitue bien une borne supérieure sur l'accélération limite. Pour l'évaluation, nous avons approximé la dérivée en calculant le pente entre deux points de la courbe obtenus pour 8 processeurs et pour 16 processeurs.

La troisième méthode est bâtie à partir de la loi d'Amdahl, qui tient compte de la fraction sérielle d'une parallélisation. À partir de l'expression de la loi d'Amdahl donnée à la section 1.2.3, on peut également donner une borne supérieure

sur l'accélération pour une taille donnée. Dans ce cas, il faut faire tendre la valeur de p vers l'infini pour déterminer la limite théorique:

$$S(n,p) = \frac{1}{1/p + f_s(n)(p-1)/p} \underset{p \to +\infty}{\longrightarrow} \frac{1}{f_s(n)}.$$

5.4 Approche centralisée

Afin de tester cette approche, nous avons utilisé trois critères de sélection des sous-problèmes en sélectionnant tantôt les sous-problèmes selon leur profondeur ($D(Q)$), tantôt selon la valeur de la borne inférieure de leur parent ($Z^l(Q^p)$) (on rappelle que les sous-problèmes qui se retrouvent dans le(s) pool(s) sont non-évalués), tantôt en considérant les deux aspects à la fois. Le troisième critère ($F(Q)$) est utilisé dans le but de favoriser les sous-problèmes branchés à zéro.

En modifiant ainsi la sélection des sous-problèmes selon plusieurs combinaisons de critères, on espère déterminer des parcours d'arborescences qui permettront de diminuer significativement la quantité de travail nécessaire à la résolution des exemplaires du problème.

De la sous-section 4.4.3, on rappelle la priorité de sélection ($P(Q)$) utilisée par le coordonnateur (on suppose que le sous-problème le plus prioritaire est celui ayant la plus petite valeur $P(Q)$):

$$P(Q) = p_1(-D(Q)) + p_2(Z^l(Q^p)) + p_3(F(Q)).$$

Dans le but de reproduire le type de sélection de l'algorithme séquentiel, nous avons créé une version de la priorité en mettant tout le poids sur le critère $D(Q)$. Dans ce cas, la valeur des poids est: $p_1 = 1$; $p_2 = p_3 = 0$. De cette façon, un sous-problème est favorisé si sa profondeur associée est élevée. Nous identifions cette version par la lettre D. Dans le but de perturber l'ordre d'exploration précédent, nous avons créé une deuxième version de priorité en mettant tout le poids sur le critère de borne inférieure, $Z^l(Q^p)$. Dans ce cas, la valeur des poids

varie quelque peu: $p_1 = p_3 = 0$; $p_2 = 1$. Ainsi, un sous-problème est favorisé si la borne inférieure de son parent est faible. Nous identifions cette version par la lettre B. Enfin, dans le but de raffiner les priorités de sélection précédentes, nous avons combiné les trois critères, en conservant toutefois dans un cas, le poids sur le critère de profondeur, et dans l'autre cas, le poids sur le critère de borne inférieure. Donc, une troisième version, issue de la première, est définie par la somme pondérée formée des poids suivants: $p_1 = 10^{10}$; $p_2 = 1$; $p_3 = 1$. Étant donné que la valeur de $Z^l(Q^p)$ est souvent très grande, il est nécessaire de mettre un poids très élevé sur $D(Q)$ afin qu'il demeure le critère principal. Par conséquent, nous fixons p_1 à une valeur arbitrairement grande alors que nous laissons les deux autres poids à 1. Nous identifions cette version par les lettres DB. De façon similaire, nous créons une quatrième version, issue de la seconde, en favorisant le critère $Z^l(Q^p)$. Dans ce cas, nous modifions quelque peu la définition de la priorité de sélection de façon à ce que la profondeur représente le deuxième critère. En supposant que le nombre de variables binaires des problèmes à résoudre ne dépasse pas 10^α, où α est un entier positif, on prendra comme poids: $p_1 = 1$, $p_2 = 10^{\alpha+1}$, $p_3 = 1$. Nous identifions cette dernière version par les lettres BD.

5.4.1 Pénalité de recherche

On rappelle que les procédures d'évaluation de bornes et les règles de séparation utilisées par un travailleur sont les mêmes que celles de l'algorithme séquentiel. Par conséquent, une pénalité de recherche supérieure à 1 signifie que l'exploration est réalisée selon un parcours moins efficace que celui de l'algorithme séquentiel et qui défavorise la découverte rapide de solutions réalisables de bonne qualité. Inversement, une pénalité inférieure à 1 signifie que le parcours réalisé par la parallélisation est plus efficace que celui de l'algorithme séquentiel et favorise la découverte de solutions réalisables de bonne qualité. Dans ce dernier cas, on observe une diminution de la quantité totale de sous-problèmes à examiner pour arriver à la solution optimale. Les tableaux 2 et 3 donnent un aperçu des

pénalités obtenues en utilisant respectivement 16 et 8 processeurs.

Prob	D	B	DB	BD
$P2$	1.104	1.181	1.109	1.195
$P3$	0.984	0.984	0.979	0.984
$P4$	0.945	0.945	0.945	0.890
$P5$	0.966	0.966	0.966	0.966
$P6$	1.251	1.393	1.287	1.389
$P7$	1.000	1.000	1.000	1.000
$P8$	1.034	2.340	1.024	2.342
$P9$	0.997	1.081	0.997	1.081

TABLEAU 2: Approche centralisée: facteurs de pénalité (p=16)

On constate que pour la majorité des problèmes que nous avons testés, les versions qui favorisent la borne inférieure (B et BD) génèrent une pénalité de recherche plus élevée ou égale à celles qui favorisent la profondeur (D et DB), et ce, peu importe les caractéristiques de l'exemplaire et le nombre de processeurs utilisés. Seule la résolution du problème $P4$ à l'aide de 16 processeurs fait exception à cette affirmation.

En favorisant la borne inférieure, le coordonnateur affecte en priorité aux travailleurs, des sous-problèmes pour lesquels la borne inférieure du père est faible. On rappelle que dans une méthode de SÉP, la borne inférieure d'un sous-problème est toujours plus grande ou égale à celle de son sous-problème *parent*. Par conséquent, sélectionner des sous-problèmes dont la borne inférieure est faible correspond à sélectionner des sous-problèmes situés près de la racine. Globalement, ce type d'exploration correspond à une recherche en largeur. Ce type d'exploration diffère grandement de celui défini par l'algorithme séquentiel, retarde la découverte de solutions réalisables de bonne qualité et entraîne, dans certains cas, une pénalité de recherche importante. Notamment, on remarque les

Prob	D	B	DB	BD
$P2$	1.054	1.190	1.045	1.181
$P3$	0.994	0.982	0.995	0.982
$P4$	0.890	0.890	0.890	0.890
$P5$	0.966	0.966	0.966	0.966
$P6$	0.911	1.377	0.907	1.405
$P7$	1.000	1.000	1.000	1.000
$P8$	1.020	2.338	1.011	2.336
$P9$	0.999	1.082	0.997	1.082

TABLEAU 3: Approche centralisée: facteurs de pénalité (p=8)

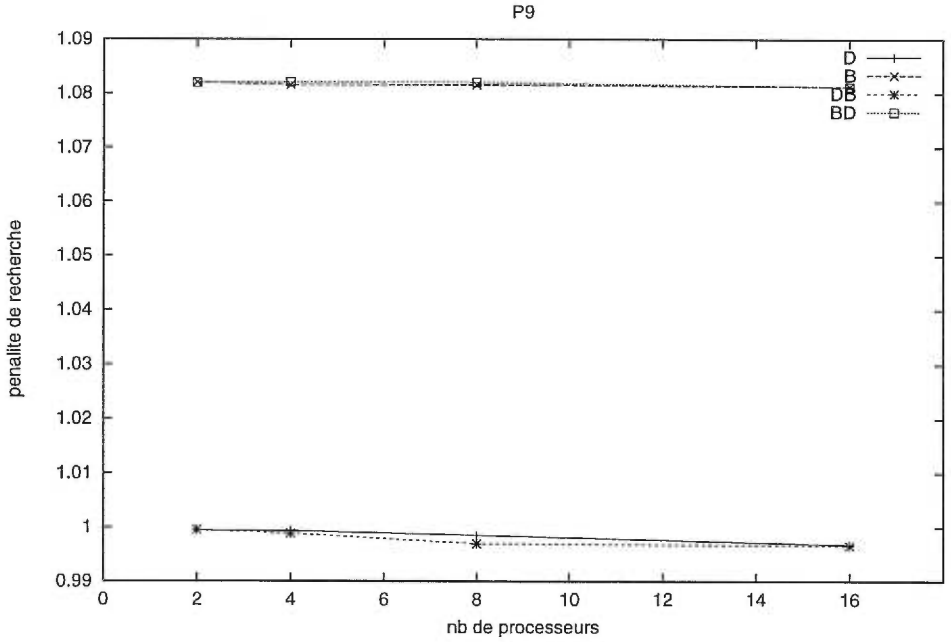

FIGURE 17: Approche centralisée: pénalité de recherche

grandes pénalités générées par les versions B et BD appliquées à la résolution du problème $P8$.

Par contre, en affectant en priorité des sous-problèmes dont la profondeur est grande, il semble que le coordonnateur tend à reproduire le type de parcours de l'algorithme séquentiel. Par ce type de sélection, le coordonnateur dirige clairement l'exploration vers les sous-problèmes feuilles. D'une part, la découverte de solutions réalisables est favorisée et, d'autre part, on favorise la mise à jour de la meilleure solution réalisable. Toutefois, on observe que les versions D et DB peuvent aussi générer des pénalités de recherche importantes comme c'est le cas pour la résolution du problème $P6$ avec 16 processeurs.

Il semble qu'en combinant les critères de profondeur et de borne inférieure dans l'expression de la priorité, on n'observe pas d'amélioration significative du parcours.

Malgré que certaines versions produisent une pénalité de recherche, les courbes de pénalité démontrent que ces parallélisations sont extensibles: en faisant varier le nombre de processeurs, les pénalités demeurent stables. La figure 17 illustre un exemple typique du genre de courbes obtenues.

5.4.2 Équilibrage de travail

Les facteurs d'équilibrage de travail sont définis par le rapport entre l'utilisation du travailleur étant resté actif le moins longtemps sur celui étant resté actif le plus longtemps. Les valeurs de facteur d'équilibrage prennent leurs valeur dans l'intervalle [0,1], où une valeur de 0 indique un fort déséquilibrage alors qu'une valeur près de 1 indique un bon équilibrage. Les tableaux 4 et 5 dressent une liste des résultats obtenus avec 16 et 8 processeurs respectivement.

De ces résultats, on observe que la qualité de l'équilibrage fourni par les versions qui favorisent la profondeur est similaire à celle fournie par les versions

Prob	D	B	DB	BD
$P2$	0.882	0.937	0.871	0.908
$P3$	0.919	0.964	0.932	0.950
$P4$	0.595	0.660	0.544	0.632
$P5$	0.828	0.823	0.835	0.856
$P6$	0.885	0.960	0.890	0.915
$P7$	0.647	0.559	0.647	0.656
$P8$	0.969	0.977	0.977	0.976
$P9$	0.983	0.981	0.983	0.981

TABLEAU 4: Approche centralisée: facteurs d'équilibrage de travail (p=16)

Prob	D	B	DB	BD
$P2$	0.933	0.967	0.911	0.960
$P3$	0.971	0.967	0.972	0.976
$P4$	0.895	0.901	0.788	0.907
$P5$	0.884	0.881	0.879	0.846
$P6$	0.896	0.966	0.895	0.954
$P7$	0.775	0.789	0.826	0.813
$P8$	0.978	0.981	0.982	0.982
$P9$	0.988	0.985	0.987	0.984

TABLEAU 5: Approche centralisée: facteurs d'équilibrage de travail (p=8)

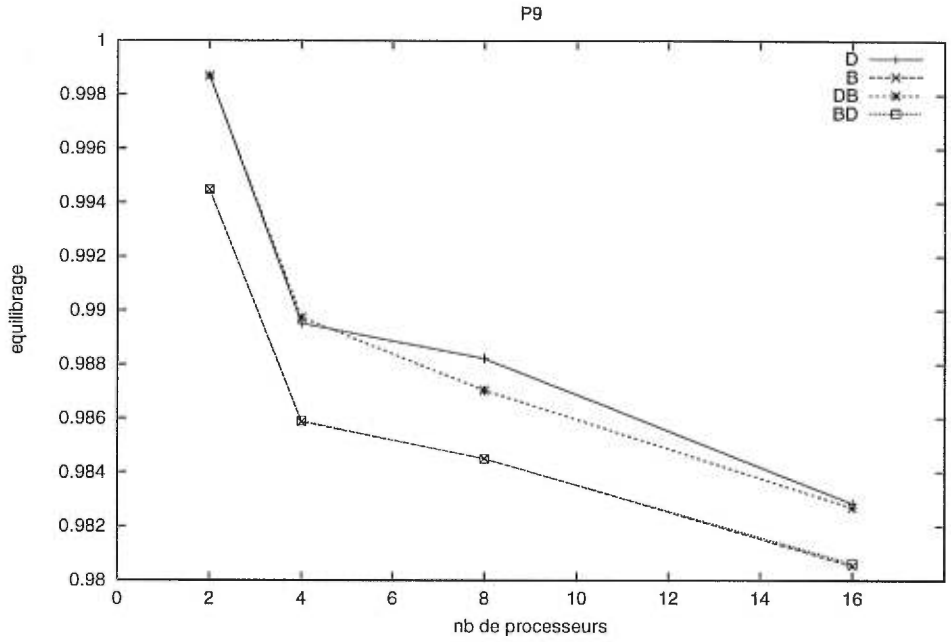

FIGURE 18: Approche centralisée: facteur d'équilibrage de travail

qui favorisent la borne inférieure, mais avec un léger avantage pour ces dernières. De plus, on observe que la qualité de l'équilibrage augmente lorsque le nombre de processeurs diminue. *A priori*, on accepte le fait que plus le nombre de sous-problèmes examinés est élevé, plus il est facile d'équilibrer le travail. D'une part, on a vu à la sous-section précédente que les approches qui favorisaient la borne inférieure généraient des pénalités de recherche plus importantes que les approches favorisant la profondeur. D'autre part, la diminution du nombre de processeurs contribue à augmenter le ratio du nombre de sous-problèmes examinés par chacun des travailleurs. Par conséquent, on comprend dans ces deux situations pourquoi la qualité de l'équilibrage augmente.

On remarque que pour la résolution des problèmes $P4$ et $P7$, la stratégie d'affectation de l'approche centralisée a beaucoup de mal à équilibrer le travail. Pour comprendre cet aspect, il est nécessaire de considérer un aspect supplémentaire: la granularité de calcul associée aux sous-problèmes. Le tableau 6 contient les

granularités de calcul moyennes associées à chaque problème, c'est-à-dire le rapport entre le temps de résolution sur le nombre de sous-problèmes explorés de l'algorithme séquentiel $(T_s(n)/n)$.

Prob	n	$T_s(n)$	$T_s(n)/n$
$P2$	221	185	0.837
$P3$	565	281	0.497
$P4$	145	339	2.338
$P5$	145	38	0.262
$P6$	247	125	0.506
$P7$	99	121	1.222
$P8$	3681	3013	0.819
$P9$	10825	8146	0.753

TABLEAU 6: Granularités moyennes

À partir du tableau 6, on constate la forte variation de l'effort de calcul requis pour résoudre les sous-problèmes. Il est important de noter également que ces granularités sont des valeurs moyennes, où l'écart entre le sous-problème qui a nécessité le moins d'efforts de calcul et celui qui en a nécessité le plus peut être très grand. Pour appuyer ce fait, mentionnons notamment que la procédure d'évaluation définie par l'algorithme séquentiel dépend de la valeur de la meilleure borne supérieure trouvée: plus sa valeur est faible, plus le travail requis est faible. Donc, l'évaluation d'un sous-problème associé à une solution spécifique peut nécessiter un temps de calcul qui dépend de la vitesse de détermination des meilleures solutions réalisables.

On remarque notamment les fortes granularités moyennes associées aux problèmes $P4$ et $P7$. De plus, le nombre de sous-problèmes explorés pour ces problèmes est relativement faible. Une étude plus détaillé des expérimentations a montré que pour ces problèmes, on observe une grande variation de l'effort de calcul requis pour le calcul des bornes. Ainsi, la granularité moyenne n'est pas

représentative de la granularité de l'ensemble des sous-problèmes. En effet, pour un groupe de sous-problèmes incluant la racine, le temps de calcul est beaucoup plus long que celui requis par les autres sous-problèmes.

En résumé, la qualité de l'équilibrage est favorisée lorsque:

- $N(n,p)$ est élevé;
- p est faible;
- la variation de la granularité est faible.

5.4.3 Utilisation minimum

Comme on l'a mentionné à la section 5.3, l'utilisation minimum permet de confirmer ou d'infirmer la valeur fournie par le facteur d'équilibrage. Les tableaux 7 et 8 donnent les utilisations minimum pour tous les problèmes traités à l'aide de 16 et 8 processeurs respectivement.

Prob	D	B	DB	BD
$P2$	0.663	0.904	0.661	0.877
$P3$	0.802	0.868	0.841	0.850
$P4$	0.394	0.382	0.366	0.505
$P5$	0.545	0.532	0.563	0.579
$P6$	0.633	0.889	0.687	0.853
$P7$	0.413	0.408	0.629	0.421
$P8$	0.903	0.976	0.910	0.976
$P9$	0.966	0.980	0.965	0.980

TABLEAU 7: Approche centralisée: utilisation minimum (p=16)

En accord avec les valeurs d'équilibrage présentées à la sous-section précédente, les valeurs obtenues pour l'utilisation minimum ne font que confirmer la difficulté pour certains problèmes à équilibrer le travail. On remarque notamment

Prob	D	B	DB	BD
$P2$	0.855	0.963	0.873	0.953
$P3$	0.923	0.933	0.927	0.931
$P4$	0.803	0.885	0.780	0.888
$P5$	0.731	0.839	0.754	0.839
$P6$	0.733	0.941	0.747	0.934
$P7$	0.620	0.721	0.629	0.722
$P8$	0.952	0.981	0.981	0.981
$P9$	0.982	0.984	0.981	0.984

TABLEAU 8: Approche centralisée: utilisation minimum (p=8)

que pour le problème $P2$ résolu à l'aide de 16 processeurs, bien que l'équilibrage fourni par les versions D et DB était près de 88%, l'utilisation minimum est près de 66%. Pour le même problème, les versions B et BD sont beaucoup plus efficaces avec un équilibrage supérieur à 90% et une utilisation minimum légèrement inférieure. Il faut ajouter que cette meilleure efficacité est expliquée en partie par les pénalités de recherche plus importantes observées précédemment. Considérant la résolution des problèmes $P5$ et $P7$ pour lesquels toutes les versions ont généré la même pénalité de recherche, on constate que les versions sont comparables en terme d'utilisation, avec un léger avantage pour la version DB à la résolution de $P7$.

Une simple analyse de notre stratégie d'affectation permet de se rendre compte que son efficacité est fortement influencée par la variabilité du temps requis par l'examen des sous-problèmes. Comme nous l'avons vu à la section 3.2.1, le cas idéal correspond à celui où le temps requis pour l'examen des sous-problèmes est constant. Dans ce cas, les travailleurs se libèrent dans l'ordre où ils ont été affectés. Dans le cas contraire, où le temps requis par l'examen des sous-problèmes varie grandement d'un sous-problème à l'autre, la séquence d'affectation des sous-problèmes est continuellement brisée et peut créer des débalancements de travail.

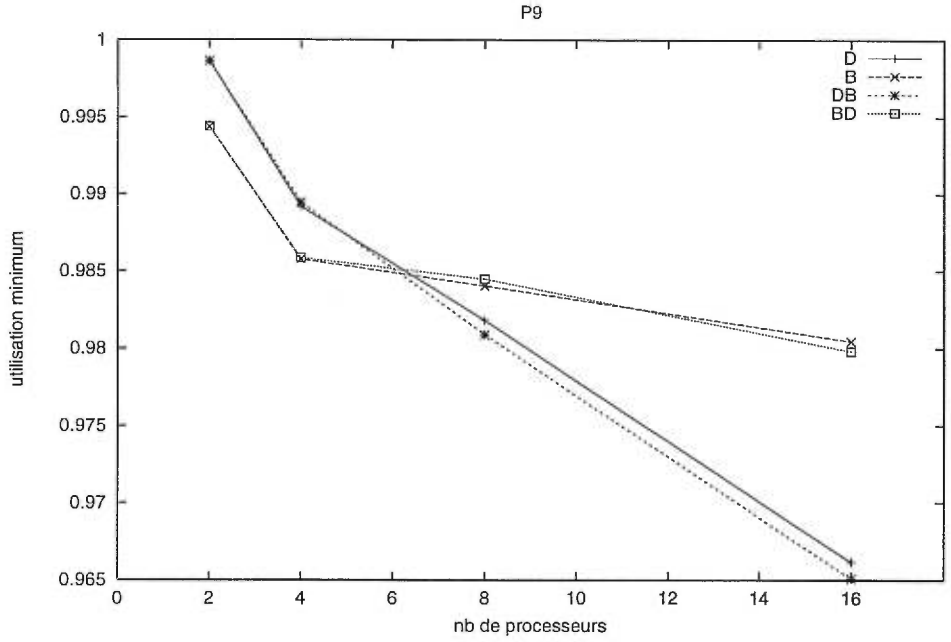

FIGURE 19: Approche centralisée: utilisation minimum

Le fait que le coordonnateur affecte séquentiellement les sous-problèmes à chacun des travailleurs entraîne également que plus le nombre de travailleurs augmente, plus le délai entre deux affectations de sous-problèmes au même travailleur augmente.

Comme pour le facteur d'équilibrage, l'utilisation minimum est favorisée lorsque $N(n,p)$ et p sont faibles et lorsque la variation de la granularité est faible.

Considérant les courbes d'utilisation illustrées par la figure 19, on remarque la chute progressive de l'utilisation pour les versions favorisant la profondeur, par rapport à celles qui favorisent la borne inférieures, celles-ci générant une forte pénalité de recherche pour le problème $P9$.

5.4.4 Accélération

Compte tenu des mesures précédentes en termes de pénalité de recherche, d'équilibrage et d'utilisation, nous sommes en mesure d'interpréter les performances atteintes par nos parallélisations. Les tableaux 9 et 10 présentent les résultats obtenus à partir des problèmes $P2$ à $P9$ pour 16 et 8 processeurs respectivement où S_A correspond à l'accélération limite évaluée par la loi d'Amdahl.

Prob	D	S_A	B	S_A	DB	S_A	BD	S_A
$P2$	3.88	4.77	4.09	4.76	3.86	4.74	4.02	4.73
$P3$	6.83	7.34	7.02	7.37	7.00	7.95	7.00	7.98
$P4$	1.88	2.10	1.93	2.16	1.87	2.09	1.96	2.10
$P5$	2.48	3.16	2.50	3.14	2.48	3.14	2.49	3.15
$P6$	3.59	4.85	3.83	4.79	3.62	4.81	3.76	4.80
$P7$	1.88	2.15	1.93	2.18	1.90	2.17	1.93	2.17
$P8$	12.87	13.57	6.52	13.50	12.89	13.38	6.67	13.52
$P9$	13.79	14.93	12.99	14.92	14.14	14.95	13.23	14.99

TABLEAU 9: Approche centralisée: accélérations (p=16)

À première vue, les accélérations peuvent paraître faibles. Il faut bien comprendre que l'approche centralisée, ainsi que les autres approches, font usage d'une stratégie d'affectation initiale dans laquelle un seul travailleur fait une recherche séquentielle à partir de la racine, les autres travailleurs demeurant inactifs. Les écarts observés avec les accélérations limites sont attribuables aux différents surcoûts de parallélisation. En fait, ce calcul théorique est basé sur une parallélisation idéale dans laquelle le temps de résolution parallèle est égal à $pT_s(n)$. Cependant, il faut noter qu'il serait possible d'améliorer davantage les accélérations en utilisant une stratégie d'affectation alternative, où plusieurs processeurs seraient mis à profit pour effectuer l'exploration initiale. Gendron et Crainic (1997) ont fait une étude comparative de plusieurs stratégies d'affectation

Prob	D	S_A	B	S_A	DB	S_A	BD	S_A
$P2$	3.29	3.75	3.10	3.75	3.31	3.72	3.00	3.63
$P3$	4.86	5.36	4.92	5.41	4.93	5.39	4.95	5.42
$P4$	1.92	2.00	1.92	1.97	1.89	1.97	1.96	2.01
$P5$	2.32	2.73	2.44	2.75	2.26	2.65	2.37	2.67
$P6$	3.24	3.86	2.92	3.84	3.16	3.77	2.88	3.83
$P7$	1.80	1.98	1.85	1.99	1.71	1.90	1.83	1.98
$P8$	7.40	7.34	3.46	7.37	7.56	7.35	3.49	7.39
$P9$	7.20	7.73	6.79	7.74	7.27	7.75	6.77	7.76

TABLEAU 10: Approche centralisée: accélérations (p=8)

initiale, et ont démontré qu'il était possible de diminuer la pénalité.

On remarque qu'avec 16 processeurs, les accélérations obtenues avec les versions B et BD pour la résolution du problème $P8$ sont très inférieures aux accélérations théoriques fournies par la loi d'Amdahl, ce qui est en accord avec les fortes pénalités de recherche observées. Pour les autres versions et les autres problèmes, les accélérations obtenues sont légèrement inférieures aux accélérations théoriques.

Les courbes d'accélération illustrées à la figure 20 ont une allure quasi-linéaire étant donné que les versions examinent sensiblement le même nombre de sous-problèmes que l'algorithme séquentiel et que l'équilibrage est de très bonne qualité. Puisque la granularité est élevée, la procédure d'évaluation nécessitant beaucoup de calculs, et que les versions favorisant la borne inférieure génèrent une pénalité assez élevée, ces versions n'arrivent pas à atteindre la performance des versions D et DB.

Malgré la grande similarité entre les courbes, on déduit que l'accélération repose sur un compromis entre la qualité de l'équilibrage, l'utilisation des processeurs et la pénalité de recherche. D'une part, les versions favorisant le critère

de profondeur équilibrent bien le travail et ne produisent pas de pénalité de recherche, ce qui les rend plus performantes que les versions favorisant le critère de borne inférieure qui balancent le travail similairement mais qui, en contrepartie, sont pénalisées par une forte pénalité de recherche.

On peut interpréter les valeurs d'accélération obtenues pour un certain nombre de problèmes. Pour le problèmes $P8$, la faible performance des versions B et BD est une conséquence de la grande pénalité de recherche générée par celles-ci. Pour les problèmes $P2$ et $P6$, la faible performance des versions D et DB dépend du mauvais équilibrage.

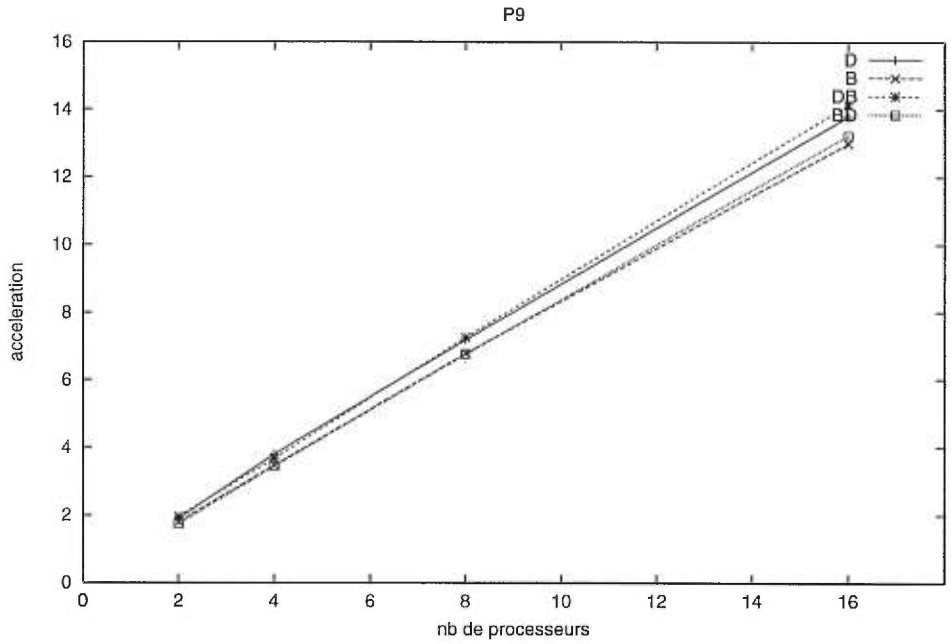

FIGURE 20: Approche centralisée: accélération

Le tableau 11 donne les valeurs d'accélération limite estimées pour les 8 problèmes testés.

Globalement, on observe que la valeur fournie par la première méthode $(1/B)$ donne la meilleure estimation. Dans les cas où la deuxième méthode (p) n'a pas fourni de valeurs (courbes décroissantes de surcoûts), on doit noter que

Prob	D			B			DB			BD		
	$1/B$	$\tilde{p}$	$1/f_s$	$1/B$	$\tilde{p}$	$1/f_s$	$1/B$	$\tilde{p}$	$1/f_s$	$1/B$	$\tilde{p}$	$1/f_s$
$P2$	6	23	7	6	-	7	6	21	7	6	-	7
$P3$	12	52	15	13	71	15	13	56	15	13	49	15
$P4$	3	11	3	3	12	3	3	11	3	3	12	3
$P5$	3	16	4	3	13	4	3	19	4	3	14	4
$P6$	5	16	7	6	-	7	5	19	7	5	-	7
$P7$	3	15	3	3	15	3	3	42	3	3	16	3
$P8$	66	40	84	12	-	81	67	33	77	12	-	82
$P9$	100	-	209	70	-	207	123	-	213	77	-	223

TABLEAU 11: Approche centralisée: accélérations limites estimées

l'estimation fournie par la première méthode constitue une sous-estimation de l'accélération limite.

On constate que ces estimations d'accélérations sont en corrélation avec la taille des problèmes. Selon les données obtenues, la version DB semble la plus prometteuse.

L'estimation selon la deuxième méthode ($\tilde{p}$) donne systématiquement des estimations qui dépassent celles fournies par les deux autres méthodes sauf pour l'exemplaire $P8$. Étant donné que nous avons réalisé les tests à l'aide de 2, 4, 8 et 16 processeurs uniquement, la courbe de surcoût de parallélisation est basée sur un nombre restreint de points, ce qui implique que l'estimation par la deuxième méthode est très grossière. Si les tests avait été réalisés en utilisant des nombres de processeurs intermédiaires, (3,5,6, etc.), la tendance aurait été plus claire, et l'estimation plus précise.

5.5 Approche décentralisée

À partir de l'approche décentralisée, nous avons créé plusieurs versions en faisant varier plusieurs paramètres qui touchent directement la stratégie d'affectation (voir section 4.5.3).

Nous avons testé les deux politiques de localisation: H, le coordonnateur choisit le donneur le plus chargé, et R, le coordonnateur choisit le donneur par cyclage. Lorsque la politique H est utilisée, les travailleurs envoient la valeur de leur charge locale lorsque celle-ci varie, à la hausse où la baisse, d'une valeur ΔL. Dans nos versions, ΔL est fixé à 1, de façon à ce que le coordonnateur soit informé de toute variation de charge. Lorsque la politique R est utilisée, un travailleur envoie sa charge au coordonnateur lorsque celle-ci passe d'une valeur inférieure à L_s à une valeur supérieure ou égale et vice-versa.

Nous avons fait varier le paramètre L_a successivement aux valeurs 0, 1 et 2 de façon à ce qu'un travailleur envoie une requête de travail respectivement lorsque son pool est vide, lorsque son pool ne contient qu'un sous-problème et lorsque son pool en contient deux.

Nous avons également fait varier le paramètre N, permettant de contrôler le nombre de sous-problèmes transférés lors d'une affectation, successivement aux valeurs 0, 1 et 2. On rappelle que le nombre exact de sous-problèmes affectés entre deux travailleurs est déterminé par le travailleur donneur selon la relation $\min(L_c - L_{min} + 1, N)$, où L_c correspond à la charge courante du pool.

Nous avons gardé constantes les valeurs des seuils L_s et L_{min} pour toutes les versions. L_s est fixé à la valeur 2, ce qui signifie qu'un travailleur est considéré "donneur" à partir du moment où son pool local contient au moins 2 sous-problèmes. L_{min} est également fixé à la valeur 2, ce qui signifie qu'un travailleur peut accepter d'affecter des sous-problèmes lorsque son pool contient au moins 2 sous-problèmes.

Nous avons donc testé huit versions de l'approche décentralisée: chacune des versions est identifiée à l'aide d'une lettre et de deux chiffres. La lettre indique la politique de sélection utilisée (H ou R), le premier chiffre indique la valeur du paramètre L_a utilisée et le dernier chiffre indique la valeur du paramètre N utilisée.

On rappelle que la politique de sélection de la stratégie d'affectation choisit ses sous-problèmes de la même façon que pour un examen local.

5.5.1 Pénalité de recherche

Les tableaux 12 et 13 donnent les facteurs de pénalité obtenus respectivement avec 16 et 8 processeurs.

Prob	$H01$	$H11$	$H21$	$H23$	$R01$	$R11$	$R21$	$R23$
$P2$	1.172	1.145	1.145	1.063	1.163	1.149	1.154	1.131
$P3$	0.986	0.986	0.986	0.968	0.984	0.984	0.984	0.972
$P4$	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.944
$P5$	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
$P6$	1.182	1.235	1.259	1.202	1.105	1.174	1.186	1.190
$P7$	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
$P8$	1.058	1.059	1.067	1.039	1.031	1.043	1.020	1.053
$P9$	0.998	0.998	0.998	1.007	0.998	0.999	0.998	1.242

TABLEAU 12: Approche décentralisée: facteurs de pénalité (p=16)

La figure 21 illustre les courbes de pénalité obtenues à partir du problème de plus grande taille ($P9$). On remarque que seules les versions $H23$ et $R23$ se démarquent de l'ensemble par leur pénalité de recherche plus élevée.

Considérant les valeurs de pénalité de recherche que l'on a obtenues, on peut diviser les problèmes en deux catégories. Ceux qui ont généré des pénalités

Prob	$H01$	$H11$	$H21$	$H23$	$R01$	$R11$	$R21$	$R23$
$P2$	1.054	1.054	1.054	1.054	1.036	1.054	1.054	1.054
$P3$	0.979	0.979	0.979	0.968	0.979	0.977	0.977	0.968
$P4$	0.944	0.944	0.944	0.944	0.944	0.944	0.944	0.944
$P5$	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
$P6$	1.089	1.146	1.202	1.243	1.109	1.162	1.190	1.154
$P7$	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
$P8$	1.027	1.032	1.026	1.016	1.020	1.026	1.033	1.046
$P9$	0.998	0.998	0.998	1.006	0.998	0.998	0.998	1.099

TABLEAU 13: Approche décentralisée: facteurs de pénalité (p=8)

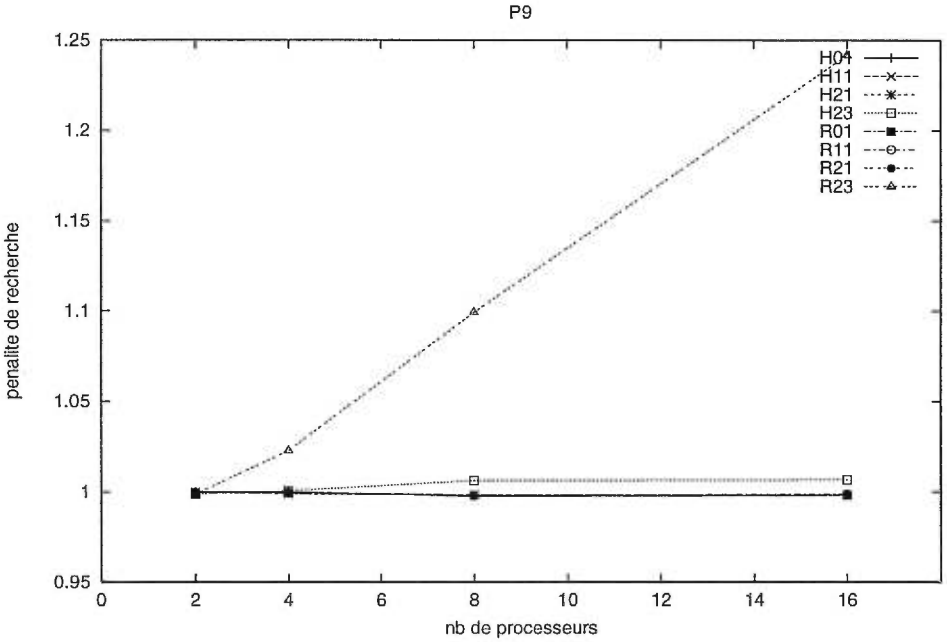

FIGURE 21: Approche décentralisée: pénalité de recherche

favorables (ou pas de pénalité du tout) ($P3$, $P4$, $P5$ et $P7$) et ceux qui ont généré des pénalités défavorables. De plus, il ne semble pas exister de corrélation entre la variation des paramètres et les facteurs obtenus.

L'augmentation du paramètre N a un impact important sur la pénalité de recherche. En effet, les versions $H23$ et $R23$ génèrent parfois des pénalités de recherche importantes et parfois ne génèrent pas de pénalité. Par exemple, le problème $P9$ résolu à l'aide de la version $R23$ sur 16 processeurs a généré une pénalité défavorable au-delà de 24%. Ce phénomène dépend de la gestion des sous-problèmes effectuée par les travailleurs. Lorsqu'un travailleur reçoit des sous-problèmes, il les insère dans son pool local (une pile) et effectue une recherche en profondeur sur ceux-ci. Supposons qu'il a reçu trois sous-problèmes. Il sélectionne le dernier inséré dans la pile et débute la recherche en profondeur en y insérant (au-dessus de la pile) les sous-problèmes générés, repoussant ainsi ultérieurement l'examen des deux autres sous-problèmes reçus au moment de l'affectation. Autrement dit, l'examen de ces deux sous-problèmes étant retardé, on modifie grandement l'ordre d'exploration et on retarde possiblement la découverte de solutions réalisables de bonne qualité. Ce phénomène ne survient pas, ou du moins a moins d'impact lorsque la valeur de N est égale à 1. De plus, l'asynchronisme intrinsèque aux implantations, fait en sorte que d'une exécution à l'autre, les groupes de sous-problèmes peuvent différer, ce qui rend la pénalité de recherche difficilement prévisible.

5.5.2 Équilibrage de travail

Les tableaux 14 et 15 contiennent les facteurs d'équilibrage pour des tests effectués avec 16 et 8 processeurs respectivement.

L'impact du choix de la politique de localisation est difficile à mesurer. On constate par contre que les versions qui utilisent la politique de localisation par cyclage (R) ont plus de mal à équilibrer que celles utilisant la politique du plus chargé (H). Dans un cas comme dans l'autre, la performance de la version con-

Prob	$H01$	$H11$	$H21$	$H23$	$R01$	$R11$	$R21$	$R23$
$P2$	0.596	0.521	0.493	0.207	0.453	0.438	0.402	0.338
$P3$	0.650	0.673	0.610	0.409	0.680	0.647	0.679	0.582
$P4$	0.220	0.163	0.202	0.044	0.170	0.167	0.202	0.134
$P5$	0.220	0.344	0.343	0.166	0.132	0.404	0.386	0.268
$P6$	0.487	0.379	0.363	0.144	0.483	0.372	0.470	0.355
$P7$	0.287	0.366	0.323	0.111	0.239	0.309	0.329	0.103
$P8$	0.827	0.906	0.925	0.795	0.822	0.852	0.775	0.919
$P9$	0.875	0.908	0.929	0.985	0.788	0.815	0.821	0.953

TABLEAU 14: Approche décentralisée: facteurs d'équilibrage de travail (p=16)

Prob	$H01$	$H11$	$H21$	$H23$	$R01$	$R11$	$R21$	$R23$
$P2$	0.843	0.744	0.742	0.589	0.798	0.616	0.562	0.735
$P3$	0.823	0.805	0.766	0.756	0.818	0.691	0.734	0.884
$P4$	0.578	0.499	0.448	0.260	0.467	0.561	0.522	0.489
$P5$	0.690	0.716	0.676	0.582	0.409	0.485	0.483	0.334
$P6$	0.699	0.589	0.740	0.708	0.748	0.575	0.630	0.792
$P7$	0.572	0.344	0.380	0.466	0.374	0.532	0.465	0.282
$P8$	0.825	0.948	0.881	0.946	0.822	0.812	0.841	0.969
$P9$	0.874	0.963	0.949	0.981	0.841	0.858	0.862	0.985

TABLEAU 15: Approche décentralisée: facteurs d'équilibrage de travail (p=8)

sidérée dépend du pourcentage de réussite des propositions d'affectation envoyées par le coordonnateur. Avec la politique H, on maximise la réussite des propositions d'affectation. Avec la politique R cependant, la situation est différente et dépend fortement des seuils utilisés. Dans ce cas, le coordonnateur choisit les donneurs en cyclant sur les travailleurs hautement chargés mais sans suivre l'évolution de leur charge locale, donc sans savoir si leur charge est rapprochée ou éloignée du seuil L_s. Ceci implique par exemple que le coordonnateur est susceptible d'envoyer des propositions qui ne pourront être satisfaites si le travailleur donneur localisé était sur le point de baisser sa charge sous le seuil L_{min}. C'est pour cette raison que le seuil L_s devrait être assez élevé par rapport au seuil L_{min} afin de favoriser la réussite des propositions et tenir compte des délais de communication des informations. Avec les paramètres que nous avons fixés, les seuils L_s et L_{min} sont égaux, ce qui défavorise grandement les versions utilisant la politique de localisation R.

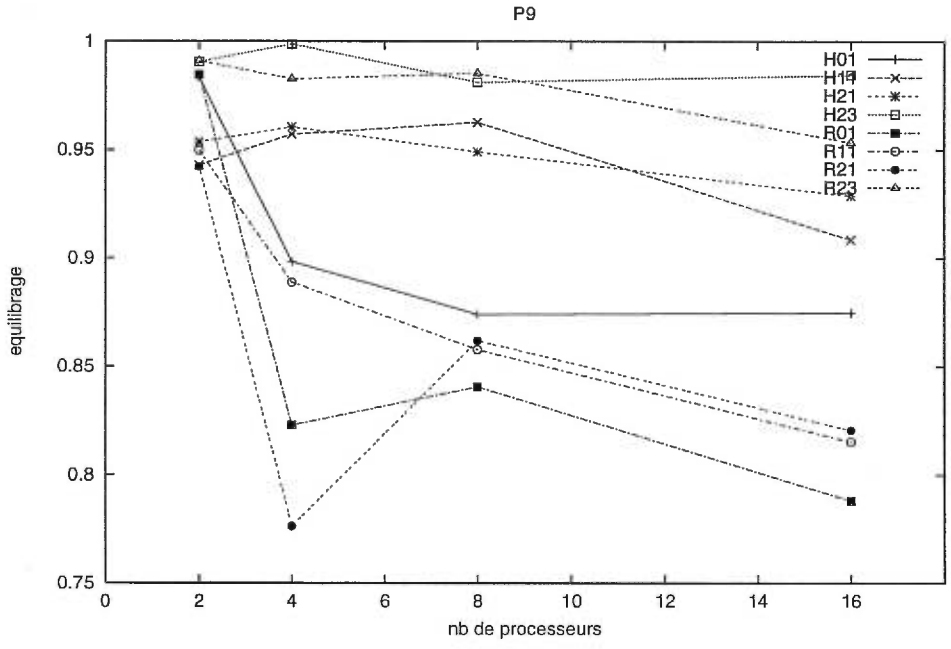

FIGURE 22: Approche décentralisée: facteur d'équilibrage de travail

L'augmentation du paramètre N implique des mouvements importants

de sous-problèmes entre les travailleurs et tend à minimiser la nécessité de rééquilibrer le travail. Globalement, les versions qui utilisent une grande valeur de N équilibrent bien le travail lorsque la taille est élevée.

5.5.3 Utilisation minimum

Les tableaux 16 et 17 donnent un aperçu des indices d'utilisation obtenus avec différents exemplaires.

Prob	$H01$	$H11$	$H21$	$H23$	$R01$	$R11$	$R21$	$R23$
$P2$	0.592	0.520	0.483	0.205	0.447	0.436	0.402	0.336
$P3$	0.642	0.670	0.602	0.403	0.652	0.645	0.677	0.580
$P4$	0.219	0.163	0.202	0.039	0.169	0.167	0.201	0.119
$P5$	0.214	0.335	0.333	0.163	0.131	0.396	0.377	0.263
$P6$	0.413	0.348	0.360	0.122	0.444	0.349	0.452	0.292
$P7$	0.250	0.353	0.314	0.110	0.236	0.288	0.295	0.102
$P8$	0.827	0.906	0.881	0.792	0.802	0.795	0.748	0.890
$P9$	0.875	0.907	0.924	0.983	0.788	0.815	0.817	0.953

TABLEAU 16: Approche décentralisée: utilisation minimum (p=16)

On observe une grande corrélation entre les valeurs d'utilisation et celles des facteurs d'équilibrage.

5.5.4 Accélération

Si on observe les courbes d'accélération (figure 23) on constate que l'ensemble des versions de l'approche décentralisée produisent une accélération quasi-linéaire. Ce comportement est explicable en raison de la qualité d'équilibrage produite par les versions et la faible pénalité de recherche générée par chacune. Ces courbes

Prob	$H01$	$H11$	$H21$	$H23$	$R01$	$R11$	$R21$	$R23$
$P2$	0.837	0.730	0.685	0.537	0.765	0.612	0.562	0.688
$P3$	0.807	0.805	0.764	0.739	0.765	0.684	0.733	0.877
$P4$	0.570	0.424	0.443	0.223	0.446	0.530	0.495	0.475
$P5$	0.658	0.661	0.596	0.577	0.405	0.470	0.472	0.333
$P6$	0.680	0.567	0.687	0.703	0.744	0.513	0.573	0.681
$P7$	0.561	0.335	0.371	0.465	0.350	0.484	0.465	0.281
$P8$	0.825	0.948	0.879	0.946	0.797	0.808	0.823	0.953
$P9$	0.872	0.963	0.949	0.979	0.838	0.858	0.862	0.982

TABLEAU 17: Approche décentralisée: utilisation minimum (p=8)

Prob	$H01$	S_A	$H11$	S_A	$H21$	S_A	$H23$	S_A
$P2$	3.92	4.60	3.69	4.59	3.76	4.63	3.37	4.49
$P3$	6.35	7.57	6.13	7.32	6.68	7.36	5.62	7.30
$P4$	1.83	2.09	1.82	2.07	1.87	2.09	1.81	2.14
$P5$	2.03	3.00	2.32	3.03	2.39	3.09	2.15	3.08
$P6$	3.25	4.74	3.35	4.69	3.01	4.15	2.79	4.59
$P7$	1.83	2.12	1.87	2.13	1.84	2.10	1.72	2.08
$P8$	12.45	13.47	12.97	13.46	12.66	13.30	11.97	13.43
$P9$	13.06	14.95	13.53	14.86	13.56	14.92	14.07	14.98

TABLEAU 18: Approche décentralisée: accélérations 1/2 (p=16)

Prob	$R01$	S_A	$R11$	S_A	$R21$	S_A	$R23$	S_A
$P2$	3.39	4.56	3.47	4.57	3.34	4.46	3.38	4.50
$P3$	6.60	7.66	6.66	7.65	6.41	7.30	6.10	7.36
$P4$	1.85	2.12	1.82	2.10	1.59	1.78	1.87	2.12
$P5$	1.66	3.05	2.37	3.08	2.26	3.00	2.14	3.04
$P6$	3.34	4.60	3.36	4.76	3.37	4.59	3.21	4.59
$P7$	1.74	2.07	1.86	2.16	1.87	2.16	1.77	2.13
$P8$	12.26	13.47	12.12	13.49	12.18	13.32	12.60	13.29
$P9$	12.32	14.88	12.22	14.75	12.39	14.96	11.21	14.95

TABLEAU 19: Approche décentralisée: accélérations 2/2 (p=16)

Prob	$H01$	S_A	$H11$	S_A	$H21$	S_A	$H23$	S_A
$P2$	3.23	3.61	3.15	3.55	3.09	3.55	2.98	3.68
$P3$	4.71	5.29	4.85	5.33	4.39	5.12	4.41	5.17
$P4$	1.85	1.97	1.82	1.99	1.85	1.99	1.78	1.97
$P5$	2.29	2.63	2.18	2.61	2.18	2.66	2.18	2.63
$P6$	2.97	3.71	2.78	3.75	2.90	3.75	2.86	3.71
$P7$	1.81	1.98	1.68	1.97	1.76	1.98	1.77	1.97
$P8$	6.72	7.32	7.57	7.35	7.44	7.36	7.43	7.30
$P9$	6.71	7.75	6.97	7.73	6.93	7.69	7.20	7.74

TABLEAU 20: Approche décentralisée: accélérations 1/2 (p=8)

Prob	$R01$	S_A	$R11$	S_A	$R21$	S_A	$R23$	S_A
$P2$	3.10	3.53	2.93	3.53	2.95	3.53	3.06	3.66
$P3$	4.64	5.31	4.54	5.31	4.65	5.28	4.92	5.30
$P4$	1.74	1.93	1.84	1.97	1.81	1.97	1.84	1.96
$P5$	1.80	2.54	2.19	2.67	2.12	2.64	1.84	2.70
$P6$	2.98	3.71	2.62	3.60	2.74	3.71	2.95	3.76
$P7$	1.73	1.97	1.74	1.96	1.73	1.95	1.53	1.75
$P8$	6.83	7.35	7.21	7.36	6.79	7.30	7.30	7.36
$P9$	6.54	7.74	6.59	7.74	6.38	7.73	6.75	7.74

TABLEAU 21: Approche décentralisée: accélérations 2/2 (p=8)

ont été produites après résolution d'un problème de taille relativement élevée nécessitant l'exploration de plusieurs sous-problèmes ($P9$).

Pour les problèmes ne nécessitant pas l'examen d'un grand nombre de sous-problèmes, la situation n'est pas tout à fait la même. D'après les tableaux 18 et 19, on constate par exemple la faible performance à la résolution du problème $P4$ en utilisant 16 processeurs. En effet, étant donné la courte durée de résolution de ce problème, les échanges d'information nécessaires à la stratégie d'affectation permettant la diffusion du travail sur l'ensemble des travailleurs prennent en proportion au temps total de résolution une place importante. On note en effet que plus la quantité de travail totale augmente, plus l'accélération obtenue devient importante.

Nous avons estimé également les accélérations limites avec les huit versions testées. Les tableaux 22 et 23 contiennent les estimations en utilisant les politiques de localisation H et R respectivement.

Encore une fois, on constate la corrélation des valeurs d'accélération estimées avec la quantité de sous-problèmes examinés pour la résolution de chacun des problèmes. Selon les valeurs obtenues, on constate que pour les problèmes qui

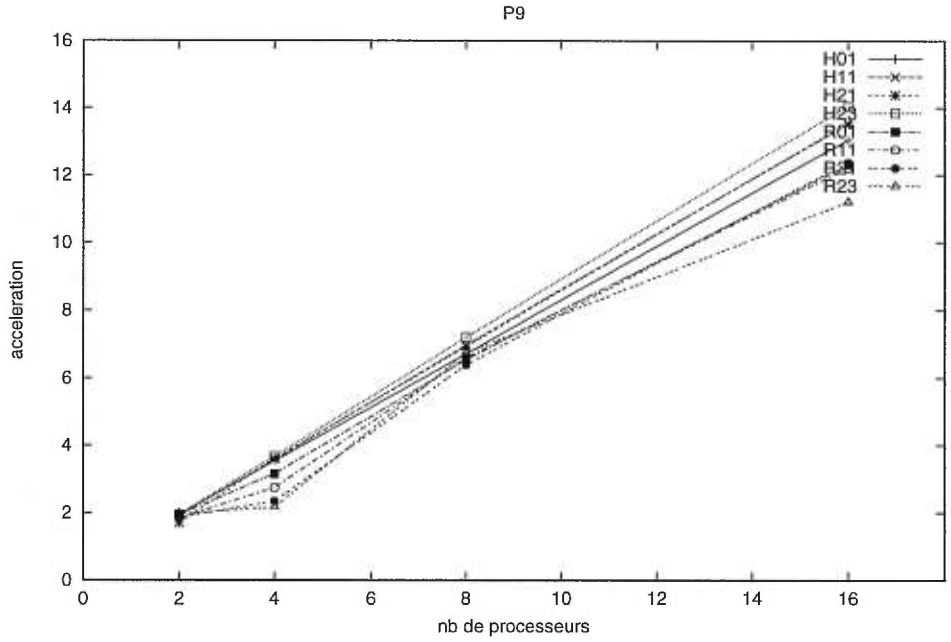

FIGURE 23: Approche décentralisée: accélération

	$H01$			$H11$			$H21$			$H23$		
Prob	$1/B$	$\tilde{p}$	$1/f_s$	$1/B$	$\tilde{p}$	$1/f_s$	$1/B$	$\tilde{p}$	$1/f_s$	$1/B$	p	$1/f_s$
$P2$	6	32	7	5	23	7	5	40	7	5	19	6
$P3$	11	32	14	10	21	13	12	-	15	9	25	14
$P4$	3	11	3	3	12	3	3	12	3	3	13	3
$P5$	3	9	4	3	16	4	3	20	4	3	11	4
$P6$	5	16	7	5	73	7	4	13	6	4	11	7
$P7$	3	13	3	3	55	3	3	15	3	2	11	3
$P8$	57	-	80	69	33	80	61	34	75	48	27	79
$P9$	72	-	215	88	-	195	89	-	208	117	-	220

TABLEAU 22: Approche décentralisée: accélérations limites estimées (1/2)

Prob	$R01$			$R11$			$R21$			$R23$		
	$1/B$	$\tilde{p}$	$1/f_s$	$1/B$	$\tilde{p}$	$1/f_s$	$1/B$	$\tilde{p}$	$1/f_s$	$1/B$	$\tilde{p}$	$1/f_s$
$P2$	5	15	6	5	29	7	5	19	6	5	16	6
$P3$	12	-	14	12	-	14	11	47	14	10	19	14
$P4$	3	17	3	3	11	3	2	8	2	3	13	3
$P5$	2	9	4	3	17	4	3	16	4	3	-	4
$P6$	5	18	7	5	-	7	5	-	7	5	16	7
$P7$	2	12	3	3	19	3	3	21	3	2	-	3
$P8$	53	-	81	50	36	81	52	-	75	60	41	74
$P9$	54	-	200	52	-	178	55	-	215	38	49	214

TABLEAU 23: Approche décentralisée: accélérations limites estimées (2/2)

nécessitent une exploration importante, les versions $H11$ et $H23$ semblent les plus prometteuses.

5.6 Approche hybride

Étant donné la grande similarité entre les approches centralisée et hybride, dans lesquelles le coordonnateur est responsable de la sélection de sous-problèmes, nous avons également testé, pour l'approche hybride, les quatre versions B, D, DB et BD. Ces versions touchent directement la politique de sélection de la stratégie d'affectation.

5.6.1 Pénalité de recherche

Les tableaux 24 et 25 contiennent les valeurs de facteurs de pénalité obtenus en utilisant 16 et 8 processeurs.

Considérant les versions D et DB de l'approche hybride, on constate que les

Prob	D	B	DB	BD
$P2$	1.095	1.140	1.086	1.127
$P3$	0.972	0.966	0.972	0.966
$P4$	0.945	0.945	0.945	0.945
$P5$	0.966	0.966	0.966	0.966
$P6$	1.287	1.538	1.283	1.534
$P7$	1.000	1.000	1.000	1.000
$P8$	1.022	3.885	1.019	3.885
$P9$	0.997	1.602	0.997	1.602

TABLEAU 24: Approche hybride: facteurs de pénalité (p=16)

Prob	D	B	DB	BD
$P2$	1.036	1.136	1.036	1.140
$P3$	0.968	0.970	0.964	0.972
$P4$	0.890	0.995	0.995	0.995
$P5$	0.966	0.966	0.966	0.966
$P6$	0.931	1.466	0.919	1.466
$P7$	1.000	1.000	1.000	1.000
$P8$	1.014	3.889	1.008	3.887
$P9$	0.997	1.603	0.997	1.604

TABLEAU 25: Approche hybride: facteurs de pénalité (p=8)

valeurs obtenues, pour la totalité des problèmes, sont pratiquement identiques à celles obtenues par l'approche centralisée. Avec les versions B et BD, la majorité des jeux de données testés donnent sensiblement les mêmes valeurs sauf pour les jeux de données $P8$ et $P9$ pour lesquels une très grande pénalité de recherche est observée. On constate que pour le problème $P8$, les versions B et BD génèrent près de quatre fois plus de sous-problèmes que l'algorithme séquentiel.

D'une part, la sélection en favorisant la borne inférieure entraîne une recherche en largeur. D'autre part, lorsqu'un travailleur reçoit un sous-problème, il explore la branche associée à ce sous-problème en conservant localement le sous-problème généré dont la variable fixée est nulle et retourne au coordonnateur le sous-problème complémentaire, et ce, jusqu'à ce qu'il atteigne une feuille.

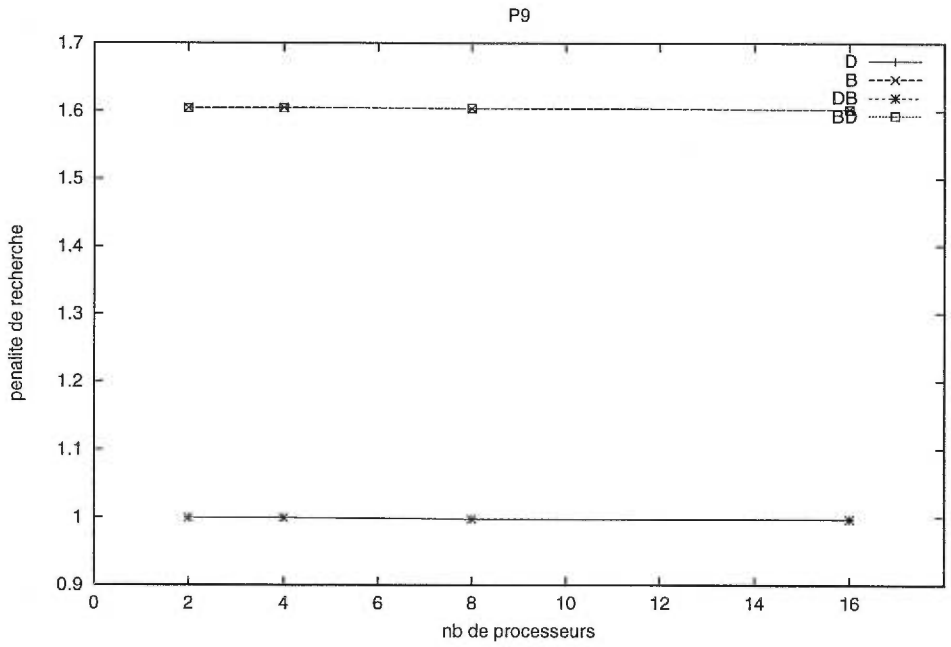

FIGURE 24: Approche hybride: pénalité de recherche

Par conséquent, si un sous-problème est choisi par le coordonnateur, et qu'il s'avère que le sous-problème en question est non-prometteur de mener à la solution optimale, un travailleur le recevant explorera la branche issue de celui-ci et

provoquera ainsi une pénalité de recherche.

5.6.2 Équilibrage de travail

Encore une fois, on remarque la difficulté à équilibrer pour les problèmes $P4$ et $P7$ en raison de l'effet combiné de la variation de la granularité et de la faible taille. Malgré cela, on constate qu'un équilibrage de bonne qualité est plus difficile à atteindre que dans le cas centralisé. On constate aussi l'effet bénéfique de la diminution du nombre de processeurs sur la qualité de l'équilibrage.

On remarque que celui des versions B et BD est meilleur en raison de la pénalité souvent plus élevée qu'avec les versions favorisant la profondeur et des caractéristiques de la procédure d'évaluation.

C'est ce principe d'ailleurs qui explique pourquoi dans les trois approches, les tableaux de facteurs d'équilibrage produits avec 8 processeurs possèdent des valeurs significativement plus élevées. Les tableaux 26 et 27 contiennent les facteurs d'équilibrage obtenus avec l'appproche hybride. Les tableaux 26 et 27 présentent les facteurs d'équilibrage obtenus avec 16 et 8 processeurs respectivement.

Prob	D	B	DB	BD
$P2$	0.606	0.903	0.601	0.936
$P3$	0.872	0.933	0.875	0.948
$P4$	0.349	0.360	0.350	0.382
$P5$	0.529	0.608	0.520	0.613
$P6$	0.851	0.887	0.819	0.911
$P7$	0.408	0.400	0.399	0.405
$P8$	0.878	0.984	0.986	0.986
$P9$	0.954	0.987	0.992	0.985

TABLEAU 26: Approche hybride: facteurs d'équilibrage de travail (p=16)

Prob	D	B	DB	BD
$P2$	0.918	0.952	0.923	0.962
$P3$	0.950	0.970	0.954	0.971
$P4$	0.806	0.784	0.803	0.763
$P5$	0.764	0.899	0.834	0.900
$P6$	0.838	0.957	0.911	0.943
$P7$	0.691	0.699	0.683	0.698
$P8$	0.964	0.990	0.991	0.989
$P9$	0.995	0.991	0.981	0.991

TABLEAU 27: Approche hybride: facteurs d'équilibrage de travail (p=8)

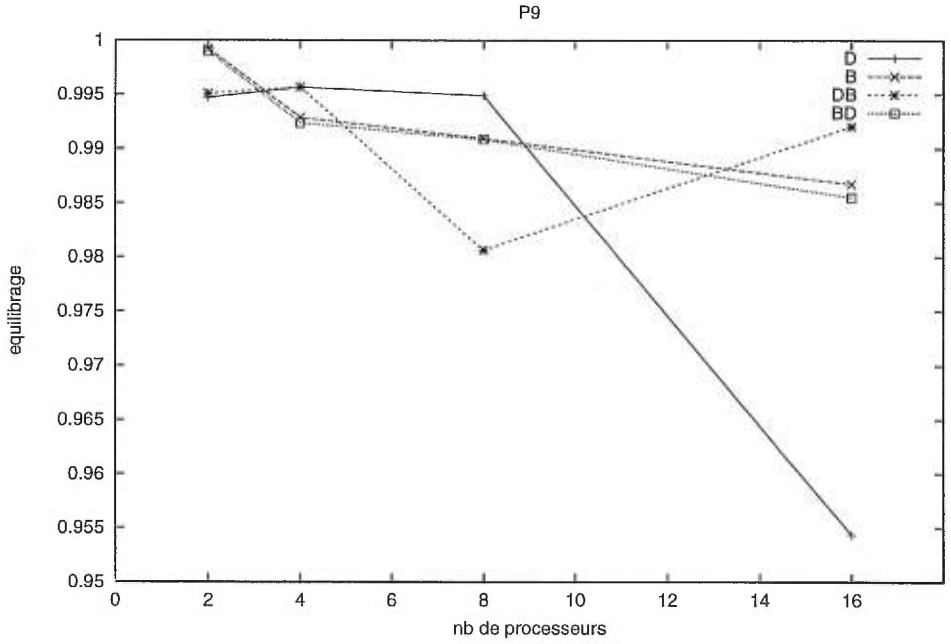

FIGURE 25: Approche hybride: facteur d'équilibrage de travail

5.6.3 Utilisation minimum

Les tableaux 28 et 29 contiennent les utilisations minimum obtenues en utilisant respectivement 16 et 8 processeurs.

Prob	D	B	DB	BD
$P2$	0.589	0.903	0.573	0.914
$P3$	0.801	0.858	0.802	0.869
$P4$	0.333	0.340	0.331	0.362
$P5$	0.515	0.545	0.509	0.546
$P6$	0.695	0.818	0.670	0.827
$P7$	0.398	0.398	0.388	0.396
$P8$	0.860	0.945	0.973	0.947
$P9$	0.948	0.980	0.991	0.979

TABLEAU 28: Approche hybride: utilisation minimum (p=16)

Similairement à l'approche décentralisée, les valeurs d'utilisation associées aux versions se rapprochent fortement de celles des facteurs d'équilibrage. Ceci signifie que la stratégie d'affectation contribue également à maintenir au moins un travailleur actif pendant presque toute la durée de la résolution. De plus, les tableaux 28 et 29 montrent la corrélation entre la quantité de travail et l'utilisation des processeurs: pour une même version, l'augmentation de la quantité de travail nécessaire à la résolution fait croître l'utilisation des processeurs alors que pour un même problème, le même phénomène survient lorsqu'on considère une version favorisant la profondeur (ne générant presque pas de pénalité de recherche) et une version favorisant la borne inférieure.

Les valeurs d'utilisation sont toutes semblables à celles des facteurs d'équilibrage, ce qui signifie qu'au moins un travailleur demeure actif pendant toute la durée de la résolution parallèle. Il n'y a que dans le cas du problème $P6$ que l'utilisation est assez distante du facteur d'équilibrage.

Prob	D	B	DB	BD
$P2$	0.890	0.952	0.903	0.960
$P3$	0.922	0.929	0.921	0.929
$P4$	0.805	0.750	0.781	0.726
$P5$	0.744	0.829	0.784	0.830
$P6$	0.741	0.929	0.797	0.917
$P7$	0.643	0.698	0.631	0.687
$P8$	0.952	0.972	0.987	0.972
$P9$	0.994	0.989	0.976	0.989

TABLEAU 29: Approche hybride: utilisation minimum (p=8)

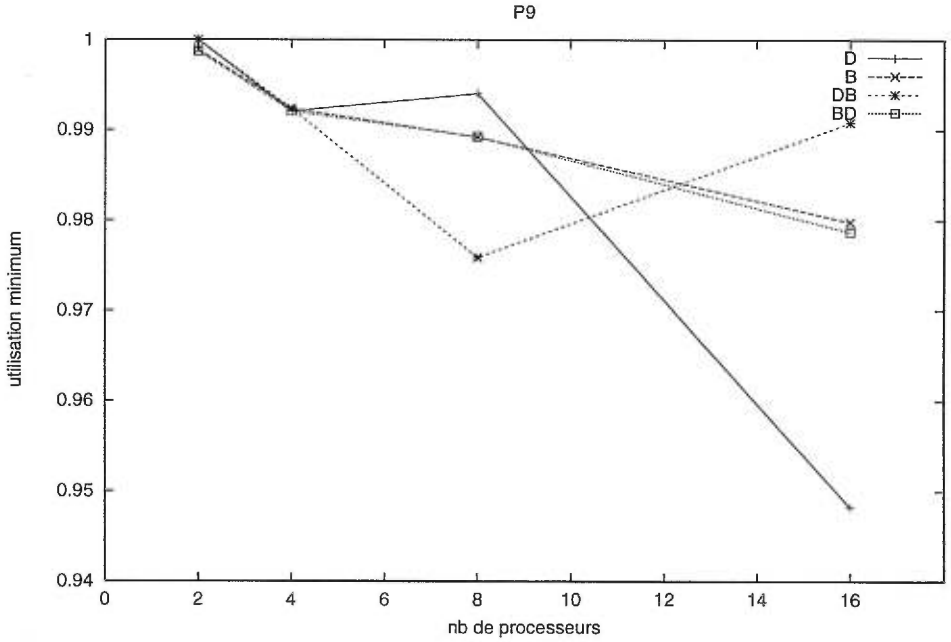

FIGURE 26: Approche hybride: utilisation minimum

5.6.4 Accélération

On retrouve aux tableaux 30 et 31 les valeurs d'accélération obtenues avec l'approche hybride en utilisant respectivement 16 et 8 processeurs.

Prob	D	S_A	B	S_A	DB	S_A	BD	S_A
$P2$	3.48	4.47	3.81	4.45	3.48	4.49	3.82	4.45
$P3$	6.53	7.29	6.65	7.33	6.57	7.73	6.76	7.67
$P4$	1.78	2.01	1.73	1.96	1.74	1.97	1.74	1.97
$P5$	2.33	2.96	2.37	3.02	2.24	2.87	2.33	2.96
$P6$	3.43	4.57	3.34	4.49	3.42	4.60	3.25	4.35
$P7$	1.71	1.97	1.72	1.96	1.73	1.99	1.73	1.99
$P8$	12.23	13.54	4.19	13.51	13.55	13.61	4.21	13.54
$P9$	13.06	14.98	9.18	14.98	13.67	14.88	9.22	14.92

TABLEAU 30: Approche hybride: accélérations (p=16)

Globalement, l'accélération est favorisée pour les approches D et DB en raison de leur faible pénalité de recherche. Si on observe les courbes d'accélération (figure 27) on constate clairement la performance des versions qui favorisent la profondeur. À l'exception des problèmes $P8$ et $P9$, qui sont de grande taille, les accélérations obtenues de toutes les versions sont assez similaires.

Le tableau 32 résume les accélérations limites estimées pour les quatre versions testées.

Malgré la grande similarité de structure des approches centralisée et hybride, on constate clairement que les valeurs estimées sont moins prometteuses que celles obtenues avec l'approche centralisée. Une diminution des valeurs est observée pour la première méthode ($1/B$) et la troisième méthode ($1/f_s$) d'estimation, alors qu'une augmentation des valeurs est observée pour un grand nombre de problèmes avec la seconde méthode ($\tilde{p}$).

Prob	D	S_A	B	S_A	DB	S_A	BD	S_A
$P2$	3.08	3.55	2.91	3.57	2.97	3.40	2.81	3.44
$P3$	4.71	5.27	4.71	5.27	4.78	5.32	4.77	5.31
$P4$	1.72	1.83	1.70	1.84	1.69	1.81	1.67	1.81
$P5$	2.20	2.61	2.23	2.60	2.23	2.60	2.27	2.65
$P6$	2.93	3.57	2.62	3.64	3.09	3.67	2.63	3.68
$P7$	1.58	1.79	1.65	1.83	1.60	1.82	1.64	1.81
$P8$	7.08	7.29	2.19	7.33	7.50	7.37	2.19	7.33
$P9$	7.12	7.75	4.76	7.74	6.87	7.74	4.82	7.75

TABLEAU 31: Approche hybride: accélérations (p=8)

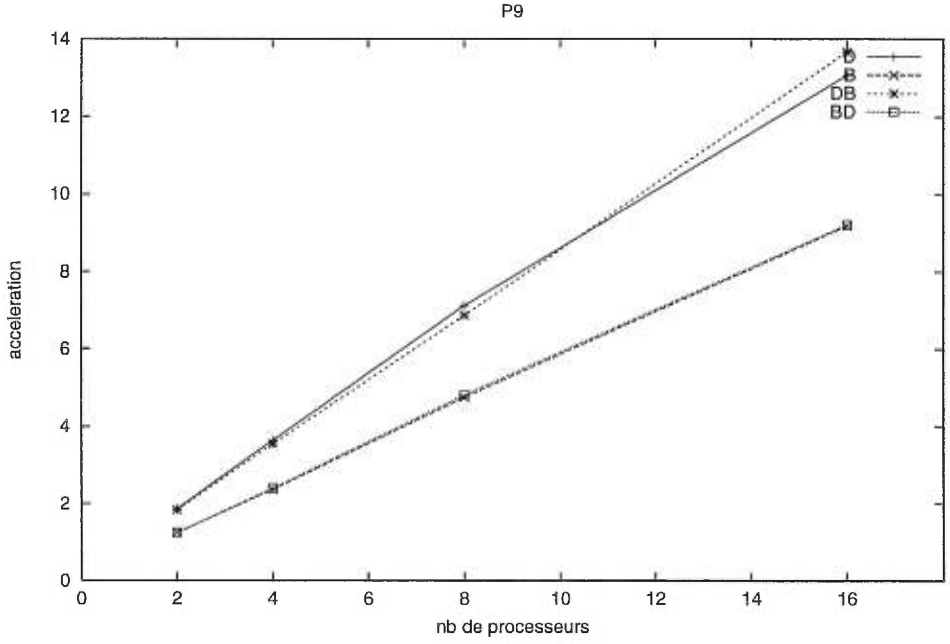

FIGURE 27: Approche hybride: accélération

Prob	D			B			DB			BD		
	$1/B$	$\tilde{p}$	$1/f_s$	$1/B$	$\tilde{p}$	$1/f_s$	$1/B$	$\tilde{p}$	$1/f_s$	$1/B$	$\tilde{p}$	$1/f_s$
$P2$	5	18	6	6	-	6	5	25	6	6	-	6
$P3$	12	49	14	12	110	14	12	38	15	12	102	14
$P4$	3	14	3	2	13	3	2	14	3	2	15	3
$P5$	3	15	4	3	15	4	3	12	4	3	13	4
$P6$	5	26	6	5	-	6	5	16	7	5	-	6
$P7$	2	22	3	2	15	3	2	22	3	2	17	3
$P8$	52	53	83	6	-	82	89	52	86	6	-	83
$P9$	72	-	221	22	-	220	94	-	199	22	-	208

TABLEAU 32: Approche hybride: accélérations limites estimées

5.7 Comparaisons

Dans cette section, nous faisons des comparaisons de performances entre les trois approches de parallélisation précédentes. Considérant les versions testées pour chacune des approches, nous en avons sélectionné une par approche qui semblait fournir des performances satisfaisantes et nous les comparons entre elles.

Afin d'établir ces comparaisons, nous reprenons dans un premier temps, les résultats présentés dans les sections précédentes à la résolution du problème de plus grande taille dont nous disposons ($P9$). Dans un deuxième temps, nous rapportons les résultats obtenus avec chacune des trois versions sélectionnées à la résolution du problème $P1$. Nous rappelons que ce dernier est résolu à l'aide d'une procédure d'évaluation à granularité faible (voir section 5.1).

Les trois versions sélectionnées sont les suivantes:

- approche centralisée (C): DB;
- approche décentralisée (D): $H11$;
- approche hybride (H): DB.

5.7.1 Granularité forte

Les tableaux 33 et 34 rassemblent les résultats déjà obtenus pour la résolution du problème $P9$ à partir de chacune des versions, en utilisant respectivement 16 et 8 processeurs.

Ver	$P(n,p)$	$F^t(n,p)$	$U^t_{min}(n,p)$	$S(n,p)$	$S_A(n,p)$
C	0.997	0.983	0.965	14.14	14.95
D	0.998	0.908	0.907	13.53	14.86
H	0.997	0.991	0.991	13.67	14.88

TABLEAU 33: Granularité forte: versions comparées à la résolution de $P9$ (p=16)

Ver	$P(n,p)$	$F^t(n,p)$	$U^t_{min}(n,p)$	$S(n,p)$	$S_A(n,p)$
C	0.997	0.987	0.981	7.27	7.75
D	0.998	0.963	0.963	6.97	7.73
H	0.997	0.981	0.976	6.87	7.74

TABLEAU 34: Granularité forte: versions comparées à la résolution de $P9$ (p=8)

Globalement, on constate que pour les trois approches de parallélisation, le nombre de sous-problèmes examinés est sensiblement le même que l'algorithme séquentiel (figure 28. Au point de vue de l'équilibrage, les approches utilisant un pool global (C et H) performent mieux (figure 29). On remarque que la version D semble avoir plus de mal à équilibrer à mesure que le nombre de processeurs augmente. L'utilisation minimum est en corrélation avec l'équilibrage (figure 30). Enfin, la meilleure accélération est atteinte par la version C (figure 31).

Considérant les accélérations limites estimées, et en accord avec ce que nous mentionnions plus tôt, il semble que l'approche centralisée (C) conserve une tendance à fournir une meilleure performance. De plus, les valeurs estimées

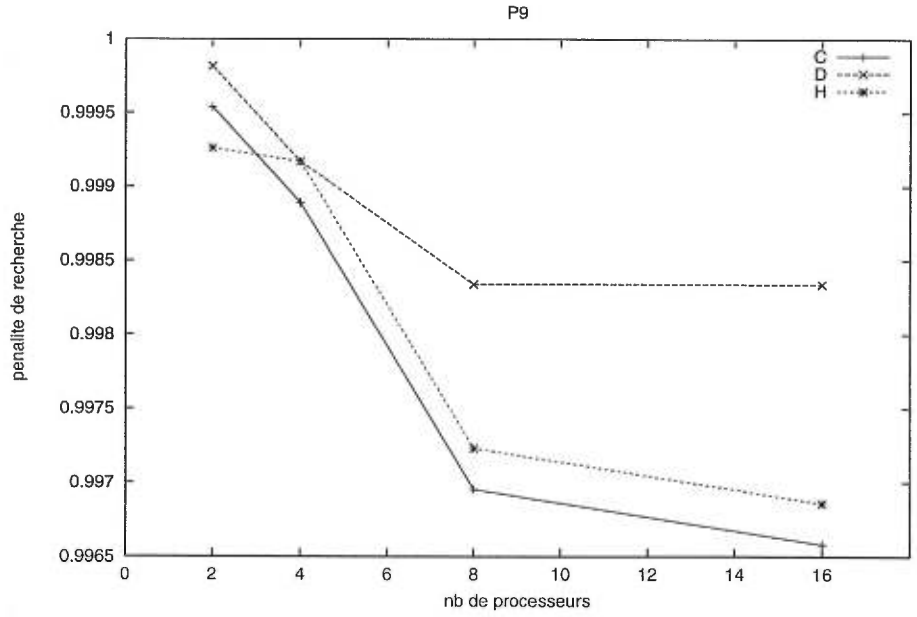

FIGURE 28: Granularité forte: pénalité de recherche

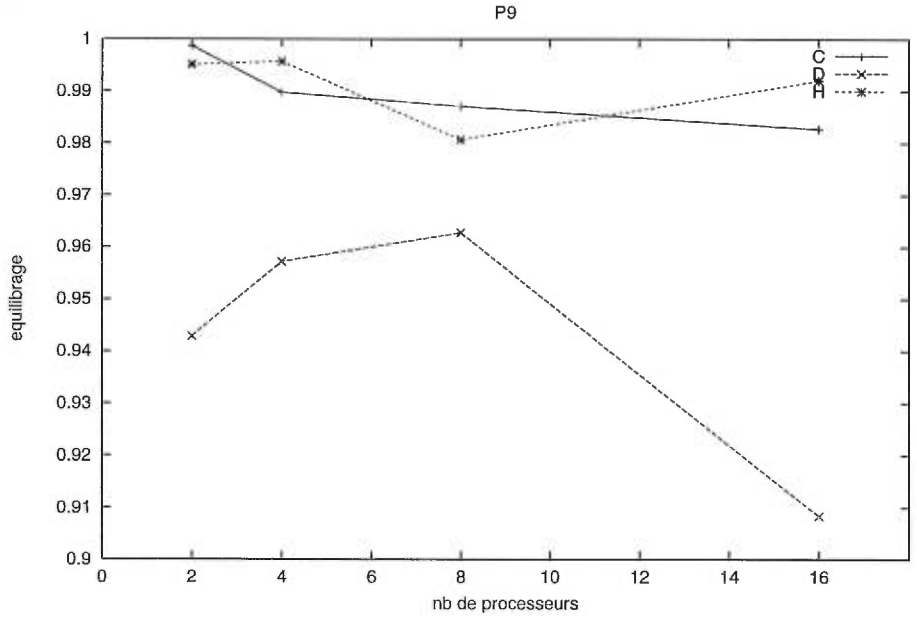

FIGURE 29: Granularité forte: facteur d'équilibrage de travail

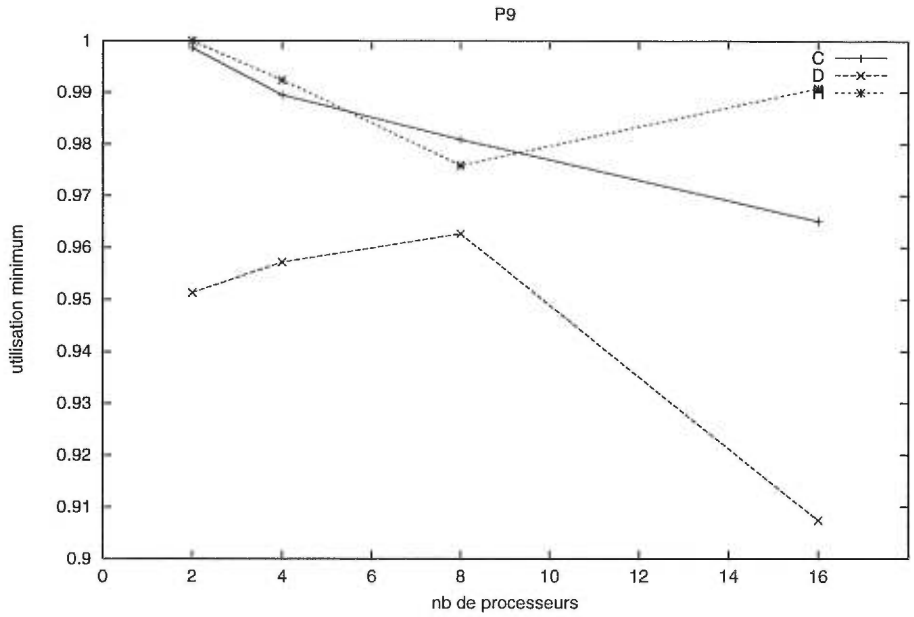

FIGURE 30: Granularité forte: utilisation minimum

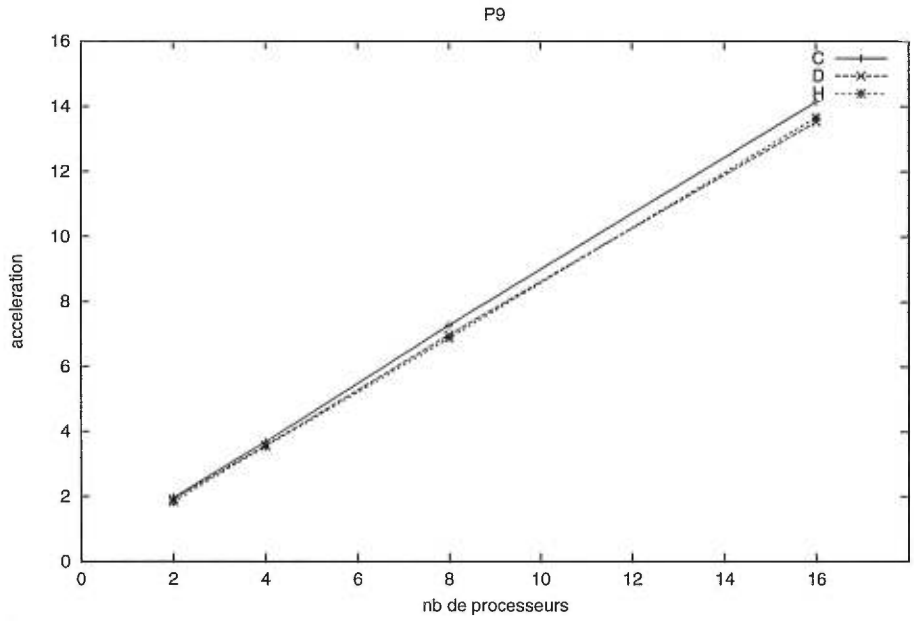

FIGURE 31: Granularité forte: accélération

par la première méthode ($1/B$) représentent en réalité des sous-estimations des accélérations limites étant donné que la courbe de surcoût de parallélisation est en décroissance pour $p = 16$ processeurs.

	C			D			H		
Prob	$1/B$	$\tilde{p}$	$1/f_s$	$1/B$	$\tilde{p}$	$1/f_s$	$1/B$	$\tilde{p}$	$1/f_s$
$P9$	122	-	212	87	-	194	93	-	198

TABLEAU 35: Granularité forte: accélérations limites estimées

5.7.2 Granularité faible

On compare maintenant les trois approches en utilisant la procédure d'évaluation à granularité faible. Les tableaux 33 et 34 rassemblent les résultats obtenus pour la résolution du problème $P1$ à partir de chacune des versions, en utilisant respectivement 16 et 8 processeurs.

Ver	$P(n,p)$	$F^t(n,p)$	$U^t_{min}(n,p)$	$S(n,p)$	$S_A(n,p)$
C	1.000	0.873	0.838	13.55	15.99
D	0.999	0.998	0.998	15.92	15.99
H	0.998	0.974	0.974	15.74	15.99

TABLEAU 36: Granularité faible: versions comparées à la résolution de $P1$ (p=16)

Du point de vue de la pénalité de recherche (figure 32), toutes les approches évaluent à peu près le même nombre de sous-problèmes que l'algorithme séquentiel. On remarque que l'approche décentralisée est celle qui présente la moins grande stabilité bien qu'elle génère une pénalité favorable très légèrement supérieure aux deux autres.

Si on considère les courbes d'équilibrage de travail (figure 33) on constate

Ver	$P(n,p)$	$F^t(n,p)$	$U^t_{min}(n,p)$	$S(n,p)$	$S_A(n,p)$
C	1.000	0.947	0.946	7.64	8.00
D	0.999	0.999	0.999	7.96	8.00
H	0.999	0.979	0.979	7.90	8.00

TABLEAU 37: Granularité faible: versions comparées à la résolution de $P1$ (p=8)

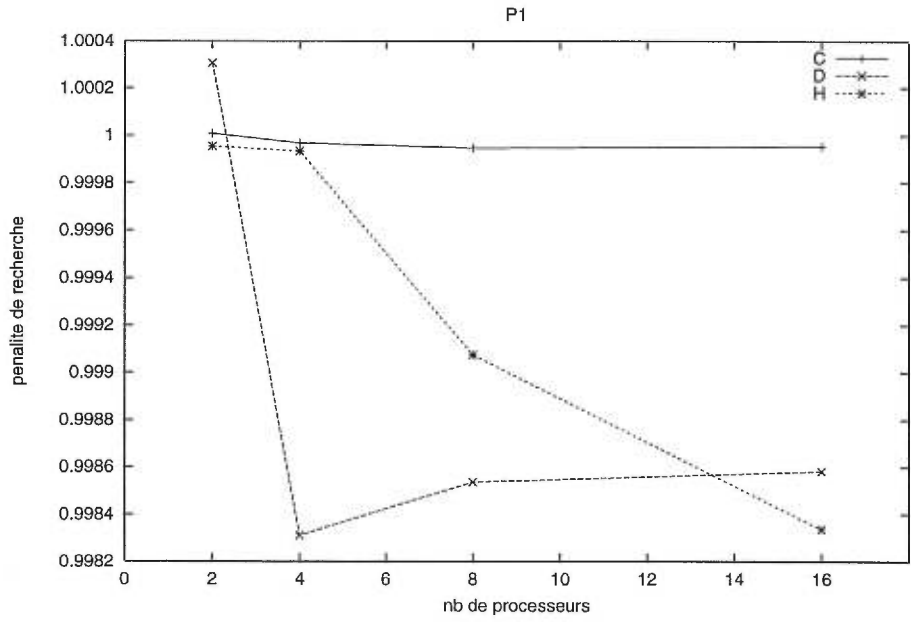

FIGURE 32: Granularité faible: pénalité de recherche

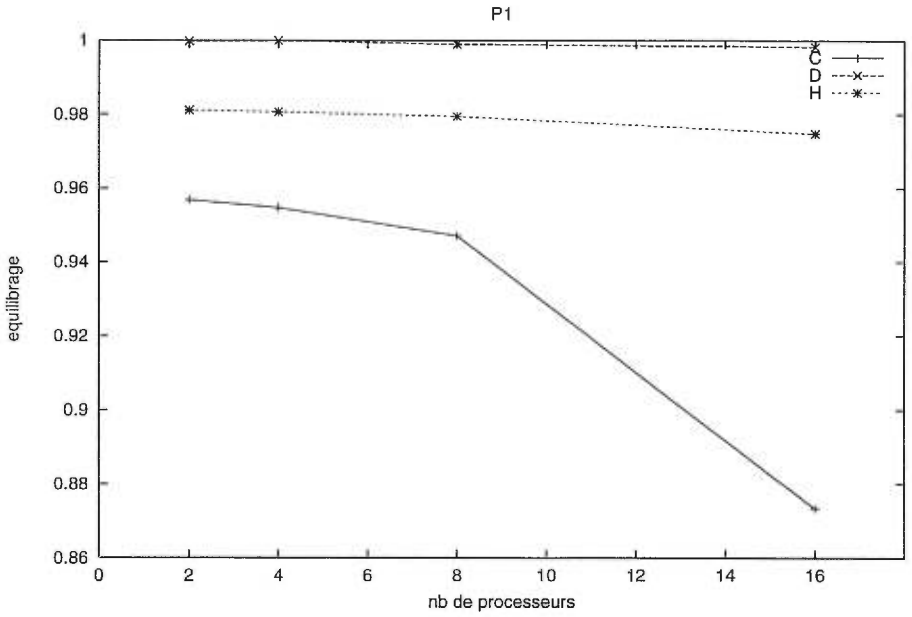

FIGURE 33: Granularité faible: facteur d'équilibrage de travail

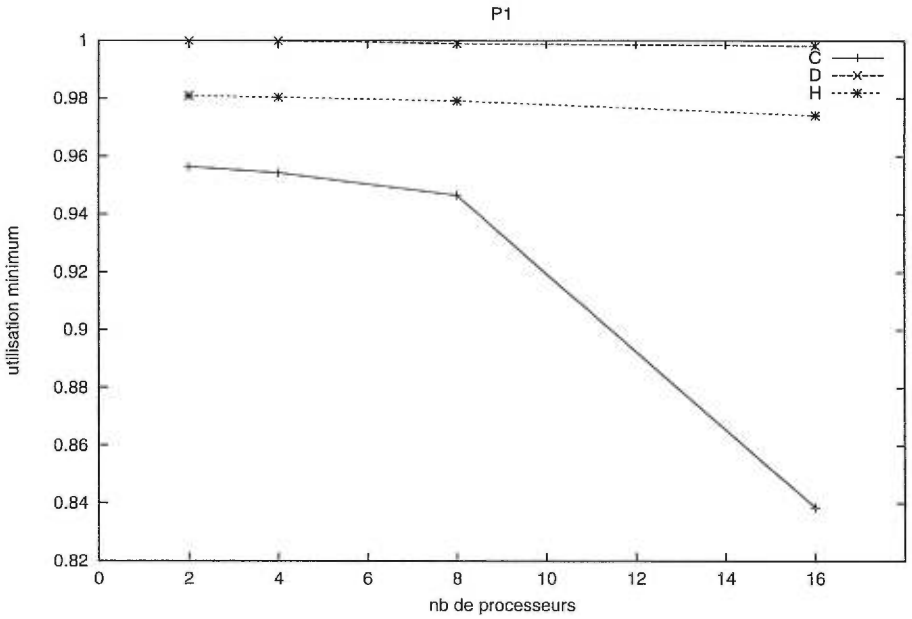

FIGURE 34: Granularité faible: utilisation minimum

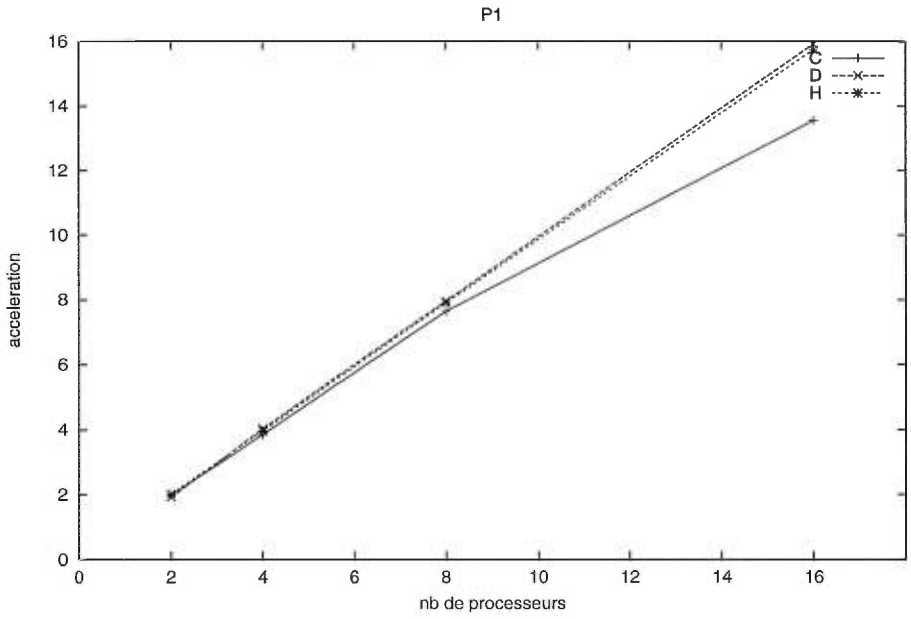

FIGURE 35: Granularité faible: accélération

que la performance de l'approche totalement décentralisée domine les autres approches. L'approche hybride présente un équilibrage de moins grande qualité quoiqu'elle soit très stable. C'est l'approche centralisée qui présente les performances les moins bonnes, avec une forte décroissance de la qualité de l'équilibrage à mesure que le nombre de travailleurs augmente. Cette contre-performance dépend directement du surcoût de communication. En effet, dans l'approche centralisée, un goulot d'étranglement est provoqué par la très grande quantité de messages qui sont échangés entre le coordonnateur et les travailleurs. Dans les deux autres versions, le nombre de messages échangés est moins important étant donné les explorations de sous-arborescences qui sont réalisées par les travailleurs. Les accélérations limites, présentées au tableau 38 confirment ces observations: l'accélération limite de l'approche centralisée, estimée par la deuxième méthode, est seulement de 38.

Si on considère les courbes d'accélération (figure 35) on constate que globalement, avec ce genre d'application, l'approche totalement décentralisée offre de

meilleures performances que l'approche totalement centralisée avec un comportement quasi-linéaire. L'approche hybride, quant à elle, se situe entre les deux.

	C			D			H		
Prob	$1/B$	$\tilde{p}$	$1/f_s$	$1/B$	$\tilde{p}$	$1/f_s$	$1/B$	$\tilde{p}$	$1/f_s$
$P1$	88	38	17110	3073	-	19678	961	-	24290

TABLEAU 38: Granularité faible: accélérations limites estimées

5.8 Conclusions

Nous avons testé quatre stratégies d'affectation pour l'approche centralisée, qui se distinguent par la politique de sélection. Cette politique consiste à choisir un sous-problème dans le pool global pour examen par un travailleur inactif. Les stratégies testées combinent les règles profondeur d'abord et meilleur d'abord. Nous avons observé qu'une règle choisissant en priorité selon la profondeur et comme second critère selon la borne inférieure du parent, que nous avons appelé la version DB, fournit les meilleurs résultats. En effet, nous avons observé que, sur plusieurs exemplaires du problème de localisation multi-produits avec équilibrage, favoriser d'abord la borne inférieure du parent comme critère de sélection (versions B et BD) peut générer de grandes pénalités de recherche puisqu'elle amène à parcourir l'arborescence davantage en largeur. Pour les quatre versions testées, nous avons observé que la stratégie d'affectation par cyclage est grandement influencée par la variation de la granularité. Plus précisément, nous avons identifié trois facteurs qui favorisent un équilibrage efficace: un petit nombre de processeurs; une arborescence de grande taille; une faible variation de la granularité.

Nous avons testé huit versions de la stratégie d'affectation pour l'approche décentralisée. Ces versions diffèrent notamment par la politique de localisation, qui consiste dans ce cas à choisir un travailleur capable de donner une partie de

ses sous-problèmes à un travailleur requéreur. Deux politiques ont été testées. La première consiste à choisir le travailleur le plus chargé (politique H), tandis que la deuxième effectue un choix par cyclage (politique R). Une fois cette politique fixée, deux paramètres sont ajustés afin de caractériser les différentes versions de la stratégie d'affectation. Le premier est le seuil d'avertissement (L_a), qui représente le nombre de sous-problèmes en deça duquel un travailleur fait une requête, alors que le second (N) fixe le nombre de sous-problèmes transférés lors d'un échange. Nous avons constaté que la variation de ces paramètres a un impact minime sur les performances, sauf lorsqu'ils perturbent l'ordre d'exploration, ce qui engendre parfois une pénalité de recherche importante. Pour les problèmes testés, nous avons observé que la politique H est légèrement supérieure à la politique R lorsque $N = 1$.

Avec l'approche hybride, nous avons testé quatre versions de la stratégie d'affectation, similaires à celles utilisées par l'approche centralisée. Les résultats sont semblables à ceux observés pour l'approche centralisée: une politique de sélection qui choisit en priorité selon la profondeur, puis selon la borne inférieure du parent, obtient les meilleurs résultats, en raison des faibles pénalités de recherche induites par un tel parcours.

Nous avons sélectionné la version la plus prometteuse des stratégies d'affectation afin de comparer les performances relatives des trois approches. Notre analyse s'est concentrée sur plusieurs jeux de données ($P2$ à $P9$) pour lesquels l'application de la procédure d'évaluation, dite à granularité forte, génère des arborescences de taille relativement petite. Parmi ces problèmes, nous avons sélectionné le plus difficile ($P9$) à des fins d'illustration. Nous avons aussi considéré un jeu de données ($P1$) pour lequel l'application d'une autre procédure d'évaluation, dite à granularité faible, génère une arborescence de grande taille.

Pour faciliter l'analyse des résultats obtenus, remarquons que la décomposition par tâches induite par chacune des approches est fort différente. Ainsi, dans chacune d'elles, une tâche, qui correspond au traitement effectué par un travailleur entre deux réceptions consécutives de sous-problèmes, est associée à:

- l'examen d'un sous-problème (approche centralisée);
- l'exploration d'une sous-arborescence partielle (approche décentralisée);
- l'exploration d'une branche jusqu'à l'obtention d'une feuille (approche hybride).

Notons d'abord que peu importe l'approche utilisée, avec les meilleures versions des stratégies d'affectation, on n'observe aucune pénalité de recherche significative pour tous les jeux de données testés. Remarquons cependant qu'avec les politiques de sélection qui favorisent la borne inférieure du parent (B et BD), l'approche hybride génère de plus fortes pénalités de recherche que l'approche centralisée. En effet, dans ce cas, l'approche hybride a tendance à effectuer l'exploration de plusieurs branches à partir de sous-problèmes peu prometteurs.

Lorsque la granularité de calcul est forte, l'approche centralisée est favorisée, pour les problèmes testés, parce qu'elle utilise mieux les processeurs. En effet, pour ces problèmes, les arborescences sont relativement petites et, par conséquent, les approches décentralisée et hybride ne génèrent pas suffisamment de tâches pour utiliser pleinement les processeurs. Par contraste, l'approche centralisée engendre une décomposition plus fine qui favorise l'utilisation et l'équilibrage du travail entre les processeurs. Néanmoins, dans l'approche centralisée, l'accès simultané de plusieurs travailleurs à un pool global provoque un goulot d'étranglement. Pour les problèmes testés, ce phénomène est révélé par des différences significatives entre les valeurs des mesures d'équilibrage et d'utilisation. Ainsi, les valeurs de la mesure d'utilisation sont nettement inférieures à celles de la mesure d'équilibrage, ce qui indique des périodes d'inactivité importantes.

Ce phénomène de goulot d'étranglement serait amplifié si le rapport du nombre de processeurs sur le temps par tâche était plus grand. Dans ce cas, la situation serait différente et il n'est pas exclu que les approches décentralisée et hybride fournissent alors de meilleures performances. Ce serait le cas si les arborescences générées étaient plus grandes.

Ces observations sont confirmées par l'étude de la procédure d'évaluation à granularité faible. En effet, dans ce cas, la granularité faible augmente le rapport du nombre de processeurs sur le temps par tâche, ce qui défavorise l'approche centralisée. Pour le problème testé, les approches décentralisée et hybride sont plus performantes car l'arborescence générée est de grande taille, créant ainsi un nombre de tâches suffisant pour favoriser l'utilisation des processeurs.

CONCLUSION

L'objectif de ce mémoire était de réaliser une étude comparative de trois approches de parallélisation de méthodes de SÉP basées sur l'organisation de la mémoire. Ces approches ont été développées dans l'optique de résoudre efficacement des problèmes dérivés de modèles de conception de réseaux. Nous avons sélectionné un problème représentatif de cette classe, le problème de localisation multi-produits avec équilibrage, pour tester les performances des trois approches.

Nos approches de parallélisation sont adaptées à un environnement parallèle asynchrone à échange de messages, où les processeurs sont puissants et capables de stocker une grande quantité de données. Elles sont basées sur l'exploration d'une arborescence par un examen concurrent des sous-problèmes. Les approches proposées, qui diffèrent selon la répartition des pools, sont les suivantes:

Une approche centralisée : Tous les sous-problèmes générés sont conservés dans un seul pool. Toutes les sélections et les insertions de sous-problèmes se font par ce pool global.

Une approche décentralisée : Tous les sous-problèmes générés sont répartis sur plusieurs pools (un par processeur). Les sélections et les insertions de sous-problèmes sont réalisées dans chacun de ces pools.

Une approche hybride : Certains sous-problèmes générés sont répartis dans les pools locaux à chacun des processeurs, les autres étant stockés dans un pool global accessible par tous les processeurs.

Chaque approche est composée d'un ensemble de processus communicants: un coordonnateur et un nombre fixe de travailleurs. Le coordonnateur assure un contrôle sur l'exploration de l'arborescence, facilite l'équilibrage de travail et coordonne les échanges d'information entre les travailleurs. Les travailleurs sont

responsables de l'évaluation et de la séparation des sous-problèmes.

Nous avons réalisé des expérimentations sur un réseau de 16 stations de travail en utilisant plusieurs jeux de données du problème de localisation multi-produits avec équilibrage. Ces expérimentations visaient d'abord à sélectionner, parmi plusieurs versions, une stratégie d'affectation plus prometteuse pour chacune des trois approches. Puis, nous avons comparé les performances relatives des trois approches. De ces expérimentations, nous pouvons tirer les conclusions suivantes:

- Pour chacune des trois approches, les tâches issues de la décomposition correspondent à:
  - l'examen d'un sous-problème (approche centralisée);
  - l'exploration d'une sous-arborescence partielle (approche décentralisée);
  - l'exploration d'une branche jusqu'à l'obtention d'une feuille (approche hybride).
- Peu importe l'approche utilisée, on n'observe aucune pénalité de recherche significative.
- Les performances relatives des trois approches dépendent d'un compromis entre le nombre de tâches générées et le rapport du nombre de processeurs sur le temps par tâche.
- Lorsque la granularité de calcul est forte et les arborescences relativement petites, l'approche centralisée est favorisée, car les deux autres ne génèrent pas suffisamment de tâches, et ainsi n'utilisent pas pleinement les processeurs.
- Lorsque la granularité de calcul est faible et les arborescences relativement grandes, l'approche décentralisée est favorisée car elle possède le plus petit rapport du nombre de processeurs sur le temps par tâche. En particulier, dans l'approche centralisée, nous observons un goulot d'étranglement en raison de l'accès simultané des travailleurs au pool global.

À ce jour, nous ne connaissons pas d'autre étude comparative de ces trois types d'approches. Toutefois, une recherche semblable a été menée par Eckstein (1997) dans laquelle on compare une approche centralisée et une approche décentralisée appliquées à la résolution de problèmes basés sur un modèle général de problèmes de programmation linéaire mixte. Ces travaux ont été réalisés sur une architecture massivement parallèle.

Les avenues de recherches qui s'ouvrent à la suite de ce travail sont nombreuses:

- Utiliser de nouvelles stratégies d'affectation initiale en vue d'améliorer davantage les accélérations et de mieux utiliser les processeurs. Notons qu'un certain nombre de ces stratégies ont été étudiées par Gendron et Crainic (1997).
- Étudier plus à fond l'impact de la variation des paramètres de l'approche décentralisée.
- Développer des approches de parallélisation adaptatives qui tiennent compte, notamment, de l'évolution dynamique des changements de charge des pools et de la vitesse de découverte des meilleures solutions.
- Diminuer les surcoûts de communication:
  - minimiser la quantité d'information véhiculée, soit en diminuant la quantité de messages envoyés, soit en utilisant des techniques de reconstruction d'information, soit en utilisant des techniques de compression d'information.
  - répartir autrement les rôles attribués aux processus: par exemple, dans l'approche centralisée, confier l'application des règles de branchement au coordonnateur, ce qui permettrait de réduire la quantité de messages véhiculés entre le coordonnateur et les travailleurs.
- Construire une banque d'outils logiciels générale de développement d'algorithmes de SÉP parallèles. Celle-ci faciliterait l'implantation d'autres

applications possédant des caractéristiques variées et ainsi, de mieux comprendre les mécanismes permettant d'obtenir des parallélisations efficaces. Plusieurs efforts ont été menés en ce sens. Mentionnons notamment la bibliothèque *BOB* de l'équipe de parallélisme non numérique de l'université de Versailles et la bibliothèque *PPBB-Lib* du groupe de calcul parallèle de l'université de Paderborn.

- Développer un simulateur permettant d'analyser le comportement de parallélisations de méthodes de SÉP. En vue de réaliser ce simulateur, il faudrait établir un ensemble de paramètres caractérisant les environnements parallèles (nombre de processeurs, organisation de la mémoire, délais de communication, etc.), ainsi que les méthodes de SÉP (taille des arborescences, granularité de calcul des sous-problèmes, etc.). Ces simulations permettraient d'avoir une idée de la performance de parallélisations avant même de les réaliser concrètement en environnement parallèle.

BIBLIOGRAPHIE

[1] Abdelrahman T.S. (1988), *Parallel Best-First Branch and Bound Algorithms on Distributed Memory Multiprocessors,* thèse de doctorat, University of Michigan, Ann Arbor.

[2] Abdelrahman T.S. et Mudge T.N. (1988), *Parallel Branch-and-Bound Algorithms on Hypercube Multiprocessors*, Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications, vol. II: Applications, pp. 1492-1499.

[3] Amdahl G.M. (1967), *Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities*, AFIPS Conference Proceedings, 483-485.

[4] Balakrishnan A., Magnanti T.L., Shulman A. et Wong R.T. (1991), *Models for Planning Capacity Expansion in Local Access Telecommunication Networks*, Annals of Operations Research, vol. 33, pp. 239-284.

[5] Barr R.S. et Hickman B.L. (1993), *Reporting Computational Experiments with Parallel Algorithms: Issues, Measures and Experts' Opinions*, ORSA Journal on Computing, vol. 5, no. 1, pp. 2-18.

[6] Bertsekas D.P. et Tsitsiklis J.N. (1989), *Parallel and Distributed Computation, Numerical Methods*, Prentice Hall.

[7] Brassard G. et Bratley P. (1987), *Algorithmique, conception et analyse*, Les Presses de l'Université de Montréal, Masson.

[8] Chabini I. (1994), *Nouvelles méthodes séquentielles et parallèles pour l'optimisation de réseaux à coûts convexes et linéaires*, thèse de doctorat, Département d'informatique et de recherche opérationnelle, Université de

Montréal, Publication CRT-986, Centre de recherche sur les transports, Université de Montréal.

[9] Chabini I. et Gendron B. (1995), *Parallel Performance Measures Revisited*, publication CRT-95-15, Centre de Recherche sur les Transports, Montréal, Canada, actes de la conférence High Performance Computing Symposium, Montréal, Canada, 10-12 juillet, pp.381-392.

[10] Cornuejols G., Nemhauser G.L. et Wolsey L.A. (1990), *The Uncapacitated Facility Location Problem*, in *Discrete Location Theory*, Mirchandani P.B. et Francis R.L., éditeurs, Wiley-Interscience, pp. 119-171.

[11] Crainic T.G., Dejax P.J. et Delorme L. (1989), *Models for Multimode Multicommodity Location Problems with Interdepot Balancing Requirements*, Annals of Operations Research, vol. 18, pp. 279-302.

[12] Crainic T.G. et Delorme L. (1993), *Dual-Ascent Procedures for Multicommodity Location/Allocation Problems with Balancing Requirements*, Transportation Science, vol. 27, no. 2, pp. 90-101.

[13] Dijkstra E.W. et Sholten C.S. (1980), *Termination Detection for Diffusing Computations*, Information Processing Letters, vol. 11, pp. 1-4.

[14] Dowaji S. (1995), *Contribution à l'Étude des Problèmes d'Équilibrage de Charge dans des Environnements Distribués*, thèse de doctorat, Université Paris VI, Paris, France.

[15] Eckstein J. (1994), *Parallel Branch-and-Bound Algorithms for General Mixed Integer Programming on the CM-5*, SIAM Journal of Optimization, vol. 4, pp. 794-814.

[16] Eckstein J. (1996), *How Much Communication Does Parallel Branch-and-Bound Need*, Parallel Optimization Colloquium, Book of Abstracts, Versailles, France, 25-27 mars 1996.

[17] Eckstein J. (1997), *Distributed versus Centralized Storage and Control for Parallel Branch and Bound: Mixed Integer Programming on the CM-5*, Computational Optimization and Applications, vol. 7, Number 2, March 1997.

[18] El-Rewini H., Lewis T.G. et Ali H.H. (1994), *Task Scheduling in Parallel and Distributed Systems*, Prentice Hall Series in Innovative Technology

[19] Erlenkotter D. (1978), *A Dual-Based Procedure for Uncapacitated Facility Location*, Operations Research, vol. 26, no. 6, pp. 992-1009.

[20] Flynn M.J. (1966), *Very High-Speed Computing Systems,* Proceedings of IEEE, vol. 54, pp. 1901-1909.

[21] Foster I. (1995), *Designing and Building Parallel Programs*, Addison-Wesley.

[22] Fox B.L., Lenstra J.K., Rinnooy Kan A.H.G. et Schrage L.E. (1978), *Branching from the Largest Upper Bound: Folklore and Facts*, European Journal of Operational Research, vol. 2, pp. 191-194.

[23] Gavish B. (1991), *Topological Design of Telecommunication Networks - Local Access Design Methods*, Annals of Operations Research, vol. 33, pp. 17-71.

[24] Geist A., Beguelin A., Dongarra J., Jiang W., Manchek R., Sunderam V. (1993), *PVM 3 User's Guide and Reference Manual*, publication ORNL/TM-12187, Oak Ridge National Laboratory, Engineering Physics and Mathematics Division, Tennessee, USA.

[25] Gendron B. (1991), *Implantations parallèles d'un algorithme de séparation et évaluation progressive pour résoudre le problème de localisation avec équilibrage*, mémoire de maîtrise, Publication no. 761, Centre de recherche sur les transports, Université de Montréal, Canada.

[26] Gendron B. et Crainic T.G. (1993), *Parallel Implementations of a Branch-and-Bound Algorithm for Multicommodity Location with Balancing Requirements*, INFOR, vol. 31, no. 3, pp. 151-165.

[27] Gendron B. (1994), *Nouvelles méthodes de résolution de problèmes de conception de réseaux et leur implantation en environnement parallèle*, thèse de doctorat, Département d'informatique et de recherche opérationnelle, Université de Montréal, publication CRT-94-50, Centre de recherche sur les transports, Université de Montréal.

[28] Gendron B. et Crainic T.G. (1994), *Parallel Branch-and-Bound Algorithms: Survey and Synthesis*, Operations Research, vol. 42, no. 6, pp. 1042-1066.

[29] Gendron B. et Crainic T.G. (1995), *A Branch-and-Bound Algorithm for Depot Location and Container Fleet Management*, Location Science, vol. 3, no. 1, pp. 39-53.

[30] Gendron B. et Crainic T.G (1997), *A Parallel Branch-and-Bound Algorithm for Multicommodity Location with Balancing Requirements*, Computers Operations Research, vol. 24, no.9, pp. 829-847, 1997.

[31] Gendron B., Crainic T.G. et Frangioni A. (1997), *Multicommodity Capacitated Network Design*, à paraître dans Telecommunications Network Design Planning, Kluwer Academic Publishers.

[32] Gengler M, Ubéda S., Desprez F. (1996), *Initiation au Parallélisme*, Masson.

[33] Ibaraki, T. (1987), *Enumerative Approaches to Combinatorial Optimization,* Annals of Operations Research, vol. 10-11.

[34] Jansen J.M. et Sijstermans F.W. (1989), *Parallel Branch-and-Bound Algorithms*, Future Generation Computer Systems, vol. 4, pp. 271-279.

[35] Krarup J. et Pruzan P.M. (1983), *The Simple Plant Location Problem: Survey and Synthesis*, European Journal of Operational Research, vol. 12, pp. 36-81.

[36] Kröger B. et Vornberger O. (1990), *A Parallel Branch-and-Bound Approach for Solving a Two-Dimensional Cutting Stock Problem,* Rapport technique, Department of Mathematics and Computer Science, University of Osnabrück, Germany.

[37] Kumar A., Kumar C.B., Ullah N. et Szygenda S.A. (1995), *An Evaluation of Load Balancing Algorithms through Simulation*, Proceedings of Parallel and Distributed Computing Systems, Florida.

[38] Le Cun B. (1996), *Des structures de données parallèles*, Thèse de doctorat, Université Paris VI, Paris, France.

[39] Li G.-J. et Wah B.W. (1984), *How to Cope with Anomalies in Parallel Approximate Branch-and-Bound Algorithms*, Proceedings of the National Conference on Artificial Intelligence, pp. 212-215.

[40] Magnanti T.L. et Wong R.T. (1984), *Network Design and Transportation Planning: Models and Algorithms*, Transportation Science, vol. 18, no. 1, pp. 1-55.

[41] Minoux M. (1989), *Network Synthesis and Optimum Network Design Problems: Models, Solution Methods and Applications*, Networks, vol. 19, pp. 313-360.

[42] Nemhauser G.L. et Wolsey L.A. (1988), *Integer and Combinatorial Optimization*, Wiley-Intersience.

[43] Schwan K., Gawkowski J. et Blake B. (1988), *Process and Workload Migration for a Parallel Branch-and-Bound Algorithms on a Hypercube Multicomputer*, Proceedings of the Third Conference on Hypercube Multiprocessors, vol. II: Applications, pp. 1520-1530.

[44] Schwan K., Blake B., Bo W. et Gawkowski J.(1989a), *Global Data and Control in Multicomputers: Operating System Primitives and Experimentation with a Parallel Branch-and-Bound Algorithm*, Concurrency: Practice and Experience, vol. 1, no. 2, pp 191-218.

[45] Schwan K., Bo W., Blake B. et Gawkowski J.(1989b), *OS Primitives for the Implementation of Distributed Objects in Multicomputers: Experimentation with a Parallel Branch-and-Bound Algorithm*, Proceedings of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications, vol. II, pp. 785-791.

[46] Standish T.A. (1980), *Data Structures Techniques*, Addison Wesley.

[47] Treleaven P.C., Brownbridge D.R. et Hopkins R.P. (1982), *Data-Driven and Demand-Driven Computer Architectures,* ACM Computing Surveys, vol. 14, pp. 93-143.