RAIRO. RECHERCHE OPÉRATIONNELLE

S. NORRE

Ordonnancement de tâches sur un système multiprocesseur - modèles déterministes et modèles stochastiques

RAIRO. Recherche opérationnelle, tome 28, n° 3 (1994), p. 221-253

<http://www.numdam.org/item?id=RO_1994__28_3_221_0>

Recherche opérationnelle/Operations Research
(vol. 28, n° 3, 1994, p. 221 à 253)

ORDONNANCEMENT DE TÂCHES SUR UN SYSTÈME MULTIPROCESSEUR– MODÈLES DÉTERMINISTES ET MODÈLES STOCHASTIQUES (*)

par S. NORRE ([1])

Communiqué par Philippe CHRÉTIENNE

Résumé. – *Cet article traite du problème d'affectation statique d'un ensemble de tâches, soumises à des contraintes de précédence, sur une architecture multiprocesseur à mémoire partagée. Deux types d'ordonnancement sont considérés : les ordonnancements déterministes (la durée d'exécution de chaque tâche est connue) et les ordonnancements stochastiques (la durée d'exécution de chaque tâche est modélisée par une loi de probabilité). Pour chacun de ces deux problèmes d'ordonnancement, nous proposons différentes méthodes de résolution et nous élaborons différents modèles afin d'évaluer l'efficacité des ordonnancements générés. La détermination d'ordonnancements déterministes est réalisée à l'aide de méthodes qui reposent sur les algorithmes d'ordonnancement par liste et sur l'algorithme du recuit simulé et ses variantes ; les modèles sont des modèles de simulation déterministe finie. Pour les ordonnancements stochastiques, nous proposons des méthodes qui sont des adaptations du recuit simulé. Plusieurs modèles sont développés : un modèle d'analyse markovienne et des modèles de simulation stochastiques. Bien que ce problème soit NP-complet, ces méthodes fournissent des solutions satisfaisantes en des temps de calcul raisonnables.*

Mots clés : Ordonnancement déterministe, ordonnancement stochastique, recuit simulé, analyse markovienne, simulation à événements discrets.

Abstract. – *This paper deals with the problem of task allocation, subjected to precedence constraints, on a multiprocessor architecture with shared memory. Two kinds of scheduling are considered: the deterministic scheduling (the duration of each task is known and constant) and the stochastic scheduling (the duration of each task is modelled by a probability law). For each of these two scheduling problems, we propose several scheduling methods and we build several models in order to estimate the efficiency of the obtained scheduling. The determination of deterministic scheduling is obtained by methods based on priority list algorithms or on the simulated annealing algorithm; models that we build use deterministic simulation. For the stochastic scheduling, we propose several methods which adjust the simulated annealing algorithm to stochastic optimization problems. Several models are built: a markovian model and a stochastic simulation model. Although this scheduling problem is NP-complete, these methods compute satisfactory solutions in reasonable computing times.*

Keywords: Deterministic scheduling, stochastic scheduling, simulated annealing, markovian analysis, discrete event simulation.

(*) Reçu en novembre 1992.

([1]) Université Blaise-Pascal, Clermont-Ferrand-II, Laboratoire d'Informatique, 63177 Aubière Cedex, France. Tel : (33) 73 40 74 43, Fax : (33) 73 40 74 44, E-mail : glm@libd1.univ-bpclermont.fr

Recherche opérationnelle/Operations Research, 0399-0559/94/03/$ 4.00

1. INTRODUCTION

Les besoins en calcul de diverses disciplines (météorologie, physique nucléaire, intelligence artificielle...), ont conduit à un essor très important des ordinateurs à architecture parallèle. L'élaboration de programmes parallèles peut être décomposée en trois étapes successives [29] :

– Spécification du programme et analyse des dépendances entre instructions : les problèmes spécifiques à cette étape concernent, d'une part, le choix d'un langage ou d'un formalisme bien adapté à la spécification de l'algorithme et, d'autre part, la définition de transformations syntaxiques ou sémantiques permettant d'exhiber le parallélisme de l'application considérée.

– Identification des tâches parallèles et ordonnancement de ces tâches sur les différents processeurs de telle sorte que les contraintes de précédence entre tâches soient satisfaites et que la durée d'exécution soit la plus petite possible. Les contraintes liées à l'architecture (nombre de processeurs, type de la mémoire...) doivent donc être prises en compte à ce niveau.

– Génération de code parallèle (automatique ou non).

Dans cet article, nous nous intéressons à la résolution du problème d'affectation statique de tâches sur une architecture multiprocesseur à mémoire partagée. Nous supposons que les tâches parallèles sont identifiées et que les contraintes de précédence entre tâches sont spécifiées à l'aide d'un graphe de précédence.

Cet article comporte trois parties.

Dans la première partie, nous présentons le contexte de l'étude et les hypothèses faites. Nous distinguons deux types de modèles : les modèles déterministes et les modèles stochastiques. Les modèles déterministes supposent que toutes les données du problème (durée d'exécution des tâches...) sont connues et constantes. L'inconvénient de ces modèles est qu'ils comportent un ensemble d'hypothèses souvent trop simplificatrices et qu'ils sont donc rarement proches de la réalité. Les modèles stochastiques intègrent cette complexité de fonctionnement en supposant que les données du problème sont modélisées par des lois de probabilité (généralement des lois exponentielles).

La seconde et la troisième partie sont consacrées à la présentation de différentes méthodes de résolution de chacun de ces deux problèmes d'ordonnancement. Les méthodes, que nous proposons, reposent sur le couplage de l'algorithme du recuit simulé avec différents modèles (markovien ou de simulation) dont le rôle est d'évaluer, d'une part, l'efficacité des

ordonnancements obtenus et, d'autre part, les performances de l'architecture multiprocesseur considérée (taux d'occupation des processeurs...). Nous étudions à la fois l'efficacité moyenne (au sens de la qualité des solutions) et la complexité moyenne en temps de ces méthodes. Des résultats expérimentaux mettent en évidence l'efficacité des méthodes proposées comparées aux algorithmes d'ordonnancement par liste.

2. CONTEXTE DE L'ÉTUDE

2.1. Hypothèses

Face à la complexité des systèmes étudiés et de leur charge, des hypothèses simplificatrices sont introduites. Ces hypothèses sont les suivantes :

– Le nombre n de tâches et le nombre m de processeurs sont finis.

– Les processeurs ont un comportement identique. Une tâche peut être exécutée sur n'importe quel processeur. Ce placement n'est pas connu *a priori.*

– Les architectures considérées sont des architectures multiprocesseur à mémoire partagée. Les conflits potentiels d'accès à la mémoire ne sont pas pris en compte.

– L'allocation des processeurs est statique.

– À un instant donné, un processeur réalise au plus une tâche.

– Une tâche est exécutée par un seul processeur.

– Les coûts de communication inter-processeurs sont nuls.

– Les tâches sont non morcelables.

– Une tâche ne peut être réalisée que lorsque tous ses prédécesseurs sont achevés.

La figure 1 présente différentes méthodes de résolution du problème d'ordonnancement. La figure 2 décrit différents moyens d'évaluer un ordonnancement selon que le modèle est déterministe ou stochastique. Les zones hachurées sur ces deux figures mentionnent les problèmes abordés dans cette étude.

2.2. Modèle déterministe

La durée d'exécution de chaque tâche est supposée constante. Dans ce contexte, le problème est de trouver un ordonnancement qui minimise la durée d'exécution totale de toutes les tâches (quantité nommée makespan) tout en respectant les contraintes de précédence entre tâches. Cependant, un tel problème est NP-difficile [14] et peut seulement être réduit à un problème

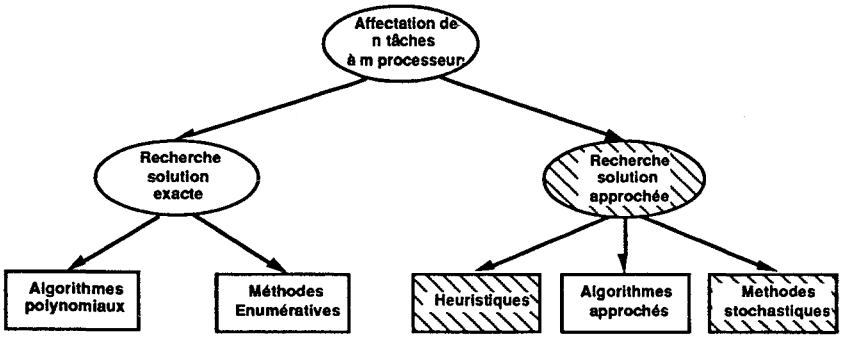

Figure 1. – Différentes méthodes de résolution du problème d'ordonnancement.

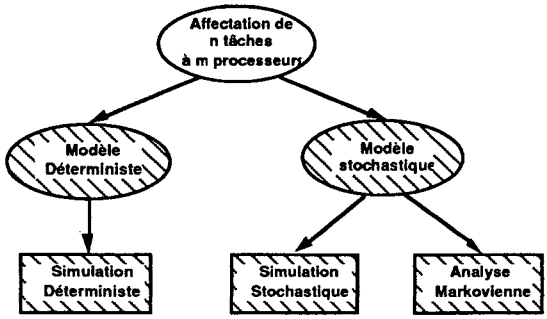

Figure 2. – Différents moyens d'évaluer un ordonnancement.

polynomial si des contraintes très restrictives sont imposées. Ainsi, lorsque les tâches sont toutes de même durée, il existe des algorithmes polynomiaux pour les différents cas particuliers suivants :

– Le nombre de processeurs est limité à deux [6].

– Le graphe de précédence est une anti-arborescence [18], une forescence [2] ou une anti-forescence [4].

– Le graphe de précédence admet comme complément un graphe triangulé [26].

La recherche d'une solution optimale peut être effectuée par un algorithme énumératif de type « Branch and Bound ». Ces algorithmes sont en moyenne plus efficaces que les énumérations explicites mais restent de même complexité dans le pire des cas.

La détermination d'une solution optimale est donc délaissée au profit de la recherche d'une solution approchée, sensée être proche de l'optimum. Nous distinguons trois types de méthodes : les heuristiques, les algorithmes

approchés et les méthodes stochastiques. Contrairement aux heuristiques, les algorithmes approchés construisent des solutions avec garantie de performance. Les méthodes stochastiques (telles que le recuit simulé, la méthode Tabou, les algorithmes génétiques) sont, quant à elles, des « méthodes qui, face à l'impossibilité d'explorer la totalité des configurations du système, permettent d'évoluer vers une solution en envisageant un nombre limité de configurations, dont le choix est guidé, en partie, par le hasard » [30]. Les algorithmes, que nous considérons dans cet article, sont les algorithmes d'ordonnancement par liste et la méthode du recuit simulé et ses variantes. Pour évaluer le makespan d'un ordonnancement, nous exploitons un modèle de simulation déterministe.

2.3. Modèle stochastique

La durée des tâches est appréhendée par une loi de probabilité. Cette hypothèse est relative à deux cas de figures différents :

– la durée d'une tâche est fonction des paramètres en entrée du programme. Par exemple, une boucle peut être réalisée une fois, dix fois ou cent fois selon l'exécution considérée,

– le système d'exploitation induit, sur la durée d'exécution d'une tâche, des variations non négligeables.

Dans ce contexte, le problème est de déterminer un ordonnancement satisfaisant en moyenne, c'est-à-dire de durée moyenne minimale. Quelques algorithmes polynomiaux permettent la résolution de ce problème lorsque certaines hypothèses simplificatrices sont ajoutées [27], [13], [8], [21].

Les méthodes de résolution que nous proposons sont des méthodes approchées dont le principe consiste à adapter le recuit simulé au traitement de problèmes stochastiques. L'évaluation des ordonnancements stochastiques obtenus est réalisée avec deux types de modèles : un modèle markovien, un modèle de simulation.

3. ORDONNANCEMENTS DÉTERMINISTES

Ce problème peut être formalisé de la façon suivante :

Soit $G = (T, X)$ un graphe acyclique orienté tel que :

– $T = \{1, 2, \ldots, n\}$ désigne l'ensemble des tâches,

– X représente l'ensemble des contraintes de précédence entre tâches [un arc est situé entre la tâche i et la tâche j $(i \neq j)$ si i doit être achevée avant que j ne débute (on note $i < j$)].

Un ordonnancement O est un ensemble de triplets $\{(j, Pr_j, t_j), j = 1, n\}$ (la tâche j débute à la date t_j sur le processeur Pr_j) qui vérifie les contraintes de précédence entre tâches. La date d'achèvement de la tâche j $(1 \leqslant j \leqslant n)$ est égale à $C_j = t_j + p_j$ où p_j représente la durée de la tâche j (les durées sont supposées être de type entier; toutefois, une généralisation au type réel peut être réalisée sans aucune difficulté). Le *makespan* $C(O)$ d'un ordonnancement O correspond à la date d'achèvement de la dernière tâche:

$$C(O) = \max_{j=1,n} C_j$$

L'objectif est donc de déterminer un ordonnancement S qui minimise le makespan c'est-à-dire tel que:

$$C(S) = \min_{O \in \mathcal{O}} C(O)$$

où $\mathcal{O}$ désigne l'ensemble des ordonnancements.

Déterminer explicitement le processeur alloué à chaque tâche peut paraître inutile dans la mesure où tous les processeurs sont identiques. Cette stratégie a seulement été adoptée afin de permettre une extension aisée des méthodes proposées au traitement de problèmes d'ordonnancement de tâches sur des architectures comportant des processeurs différents ou uniformes.

Nous résolvons ce problème d'ordonnancement à l'aide de deux types d'algorithmes: les algorithmes d'ordonnancement par liste et l'algorithme du recuit simulé.

3.1. Principes des méthodes de résolution

3.1.1. Les algorithmes d'ordonnancement par liste

Les algorithmes d'ordonnancement par liste appartiennent à la classe des algorithmes gloutons. Leur principe repose sur l'affectation d'une priorité à chaque tâche et la construction d'une liste dans laquelle les tâches réalisables sont classées par priorité décroissante (les tâches étant soumises à des contraintes de précédence, une tâche est dite réalisable à un instant donné si elle n'admet pas de prédécesseur ou si tous ses prédécesseurs sont achevés). Dès qu'un processeur est libre, la tâche de priorité la plus élevée (si elle existe) lui est affectée.

[7] a montré que, quelle que soit la manière de définir les priorités, le rapport entre le makespan obtenu par un algorithme d'ordonnancement par liste et le makespan optimum est toujours majoré par la quantité $2 - 1/m$. Le classement des tâches dans la liste a toutefois un impact important sur le makespan obtenu.

[1] présente différentes heuristiques de classement et remarque que l'heuristique qui fournit les meilleures résultats est l'heuristique HLFET (*Highest Levels First with Estimated Times*), également nommée CP (*Critical Path*). Cette heuristique classe les tâches par niveau décroissant, le niveau d'une tâche étant défini comme la longueur du plus long chemin dans le graphe de précédence, qui est compris entre cette tâche et une tâche fictive de fin d'exécution (la longueur du chemin critique désigne la somme des durées d'exécution des tâches associées aux sommets composant ce chemin).

Cette heuristique comporte donc deux étapes successives. La première étape concerne la détermination de la liste de priorités ce qui se résume au calcul de tous les plus longs chemins à partir d'une source unique d'où une complexité en $O(n^2)$ (n^2 = nombre maximum d'arêtes). La seconde étape consiste à réaliser l'ordonnancement proprement dit, ce qui se traduit par une complexité en $O(n \log_2 n)$ [10]. La complexité de cette heuristique est donc en $O(n^2)$.

L'efficacité de l'heuristique HLFET (les résultats expérimentaux obtenus par [1] montrent que l'heuristique HLFET retourne des solutions qui sont à moins de 5% de la solution optimale dans 90% des cas), sa faible complexité en $O(n^2)$ et sa robustesse justifient sa sélection parmi l'ensemble des algorithmes d'ordonnancement par liste. Cette heuristique a d'ailleurs été reprise pour développer de nombreux algorithmes [19], [9].

Comme tous les algorithmes d'ordonnancement par liste, cette heuristique présente toutefois certains inconvénients :

– Le choix de la tâche de niveau le plus élevé n'est pas toujours pertinent.

– La solution optimale n'appartient pas toujours au domaine des solutions produites par ce type d'heuristiques. En effet, dès qu'un processeur est libre et qu'il existe une tâche réalisable, cette tâche est obligatoirement affectée à ce processeur. En d'autres termes, la possibilité de laisser un processeur oisif lorsqu'il existe des tâches réalisables n'est pas considérée. Une telle démarche pourrait cependant conduire à la solution optimale dans la mesure où il est parfois plus judicieux de laisser un processeur inactif pendant un certain laps de temps afin de lui permettre ensuite de prendre en compte une tâche non réalisable au préalable.

3.1.2. L'algorithme du recuit simulé et ses variantes [30], [32]

a. Principe

Le principe de l'algorithme du recuit simulé repose sur la construction d'un ordonnancement voisin OV à partir d'un ordonnancement courant

OC. Si OV a un makespan inférieur à OC, OV est accepté et devient ordonnancement courant. Sinon, OV est accepté avec une certaine probabilité π dépendant de la différence de makespan entre les deux ordonnancements OV et OC et d'un paramètre de contrôle appelé température. Lorsque la probabilité π d'accepter un mauvais ordonnancement est nulle, on parle de *descente stochastique* ou d'*amélioration itérative.*

D'autres variantes de cet algorithme ont été proposées :

– la *méthode du kangourou* [12] consiste à réaliser une descente stochastique et à accepter une mauvaise solution lorsqu'aucune amélioration n'a été apportée pendant un nombre d'itérations donné.

– la *méthode des descentes stochastiques successives* consiste à réaliser successivement plusieurs descentes stochastiques indépendantes admettant des solutions initiales différentes et à retenir, comme solution, celle de makespan le plus faible.

L'inconvénient majeur des algorithmes itératifs (convergence vers des minima locaux) est contourné par le recuit simulé qui autorise l'acceptation de mauvaises solutions. Contrairement à la méthode Tabou [33], les risques de bouclage dans l'ensemble des solutions produites ne sont toutefois pas exclus.

La résolution de problèmes d'ordonnancement avec la méthode du recuit simulé a déjà été mise en œuvre par de nombreux auteurs [11], [9], [31]. Ces auteurs concluent que cet algorithme est très bien adapté à la résolution de ce type de problème, et, constatent que la rapidité de la convergence et la qualité du résultat dépendent essentiellement de la stratégie de construction des voisins, de la température initiale et de la fonction de décroissance de la température. Ces travaux nous ont également guidés dans la mise en œuvre de l'algorithme du recuit simulé pour la résolution de notre problème d'ordonnancement. Nous avons ainsi décidé :

– de ne pas séparer les étapes « affectation des tâches aux processeurs » et « ordonnancement des tâches » [9],

– d'utiliser un algorithme rapide pour la construction d'une solution initiale satisfaisante [31].

L'efficacité de l'heuristique HLFET et de l'algorithme du recuit simulé nous ont conduit à envisager le couplage de ces deux méthodes. L'heuristique HLFET construit la solution initiale puis l'algorithme du recuit simulé tente de modifier cette solution afin de l'améliorer.

b. Couplage de l'heuristique HLFET et de l'algorithme du recuit simulé

Comme la solution initiale est construite par l'heuristique HLFET, la stratégie de construction des voisins, que nous proposons, a pour principal objectif de pallier les inconvénients de cette heuristique. Ainsi, elle offre la possibilité de laisser un processeur oisif même s'il existe des tâches réalisables et elle autorise la sélection d'une tâche réalisable qui n'est pas obligatoirement celle de niveau le plus élevé. De façon plus détaillée, cette stratégie peut être décrite de la façon suivante :

Considérons un ordonnancement courant OC. Cet ordonnancement est composé de n triplets (j, Pr_j, t_j) $(1 \leqslant j \leqslant n)$ qui sont numérotés par ordre d'apparition croissant.

La construction d'un ordonnancement voisin OV est effectuée comme suit :

– Un triplet $E = (e, Pr_e, t_e)$ $(1 \leqslant e \leqslant n)$ est choisi aléatoirement.

– Tous les triplets de l'ordonnancement OC, qui sont antérieurs au triplet E, sont maintenus dans l'ordonnancement OV.

– L'ordonnancement OV est ensuite complété selon l'une des deux stratégies suivantes choisie aléatoirement :

Stratégie 1 :

• la tâche e est remplacée par une autre tâche j $(1 \leqslant j \leqslant n$ et $j \neq e)$ choisie aléatoirement parmi l'ensemble des tâches réalisables à la date t_e et non encore affectées à un processeur.

• l'ordonnancement des tâches restantes (y compris e) est déterminé par l'heuristique HLFET.

Stratégie 2 :

• le processeur Pr_e est laissé inactif pendant un certain délai à partir de la date t_e. Différents délais sont applicables : ils correspondent à la différence entre la fin d'exécution des tâches en cours sur les processeurs autres que Pr_e et la date t_e. Un de ces délais est choisi aléatoirement (ces délais permettent d'attendre qu'une tâche en cours d'exécution sur les autres processeurs se termine).

• l'ordonnancement des tâches restantes (y compris e) est déterminé par l'heuristique HLFET.

Si la stratégie choisie n'est pas applicable (pour la stratégie 1, il n'y a pas de tâches réalisables différentes de e à la date t_e ; pour la stratégie 2, aucun délai n'est applicable), son alternative est activée. Si aucune des deux stratégies n'est applicable, le processus est réinitialisé en choisissant un nouveau triplet E.

L'hypothèse de réversibilité (si une solution A peut être atteinte à partir d'une solution B en une étape, le changement inverse est également vrai) exigée par le théorème de [17], qui garantit la convergence théorique du recuit, n'est pas vérifiée. La condition d'accessibilité (toute solution peut être atteinte à partir de n'importe quelle autre en un nombre fini d'étapes) est par contre satisfaite, ce qui est suffisant pour démontrer la convergence théorique de l'algorithme du kangourou [12].

Une borne inférieure B du makespan est définie. Elle correspond au maximum entre les deux critères suivants :

– la longueur du chemin critique du graphe de précédence,

– le rapport de la somme des durées des tâches sur le nombre de processeurs.

Cette borne n'est pas obligatoirement atteinte. Aussi, l'algorithme du recuit simulé est itéré jusqu'à ce que cette borne soit atteinte ou jusqu'à ce qu'un nombre maximum d'itérations soit réalisé. Cette borne sur le nombre d'itérations pourrait dépendre de la taille du problème, c'est-à-dire du nombre de tâches. Cependant, par soucis d'homogénéité, nous avons préféré fixer, pour une méthode donnée, la même borne pour tous les jeux d'essais. Cette borne nous servira par la suite de référence pour apprécier la qualité des résultats retournés par les différentes méthodes d'ordonnancement proposées.

3.2. Résultats

a. Création des jeux d'essais

Les méthodes retenues ont été testées sur différents graphes de précédence construits aléatoirement. Le générateur de graphes que nous avons élaboré comporte trois étapes successives :

– *Saisie des données* telles que le nombre total de tâches, le nombre minimal et le nombre maximal de tâches sans prédécesseur (ou sans successeur), le nombre minimal et le nombre maximal de prédécesseurs (ou successeurs) que peut avoir une tâche quelconque, les durées minimale et maximale des tâches.

– *Construction du graphe* : choix aléatoire du nombre de tâches sans prédécesseur (ou sans successeur), choix aléatoire pour chaque tâche de ses prédécesseurs et de ses successeurs.

– *Simplification du graphe* : suppression de tous les arcs de transitivité du graphe.

Ainsi, les graphes construits diffèrent par le nombre de tâches qu'ils comportent (10, 20, 50 ou 100 tâches), par le degré de parallélisme mis

en œuvre (nombre de tâches parallèles plus ou moins important) et par le domaine de variation des durées d'exécution des tâches (entre 5 et 10, entre 1 et 20, entre 1 et 40).

b. Méthodes de résolution mises en œuvre

Différentes méthodes de résolution ont été testées sur les jeux d'essai retenus :

– *l'heuristique HLFET.*

– *la descente stochastique (Hdesc)* : la solution initiale est construite par l'heuristique HLFET. Le nombre maximal d'itérations est fixé à 5 000, un nombre d'itérations plus élevé n'entraînant pratiquement aucune amélioration.

– *le recuit simulé non homogène (Hrecuit)* : la température initiale est égale à la différence de makespan entre la solution initiale, obtenue par l'heuristique HLFET, et la borne inférieure B. La fonction de décroissance de la température est en $O(1/k)$ (k désigne le nombre d'itérations). Le nombre maximal d'itérations est fixé à 20 000.

La descente stochastique convergeant très rapidement vers une solution (dans une très grande majorité des cas en moins de 2 000 itérations), nous avons ensuite testé :

– *la méthode du kangourou (Hkang)* : le nombre maximal d'itérations est fixé à 20 000 et une mauvaise solution est acceptée lorsqu'aucune amélioration n'est apportée durant 2 000 itérations. La solution initiale est construite par l'heuristique HLFET.

– des *méthodes des descentes successives*

• *Hdescsuc* : la solution initiale de la première descente est construite par l'heuristique HLFET ; les solutions initiales des descentes suivantes sont obtenues grâce à l'heuristique RANDOM, qui consiste à construire une liste de priorité arbitraire [1]. Le mécanisme de construction des voisins reste basé sur l'heuristique HLFET pour toutes les descentes.

• *HRdescsuc* : pour la première descente, le mécanisme de construction des voisins est basé sur l'heuristique HLFET. Pour les autres descentes, il utilise à la fois l'heuristique HLFET et l'heuristique RANDOM. Le voisin d'un ordonnancemnet courant est alors obtenu en appliquant l'une des deux stratégies 1 et 2 décrites dans le paragraphe 3.1.2.b ; la seule différence concerne l'ordonnancement des tâches restantes qui est déterminé soit par l'heuristique HLFET (probabilité 1/2), soit par l'heuristique RANDOM (probabilité 1/2).

Pour chacune de ces deux méthodes, deux nombres maxima d'itérations sont envisagés: 20000 et 50000. La construction d'une nouvelle solution est alors réalisée lorsqu'aucune amélioration n'est apportée durant 2000, respectivement 5000, itérations.

c. Présentation des résultats

Les résultats sont synthétisés dans les tableaux 1 et 2.

Pour comparer une solution S par rapport à une solution SH, nous calculons la quantité:

$$100^* (C(SH) - C(S))/C(SH)$$

appelée gain de S par rapport à SH.

La comparaison de deux méthodes est alors réalisée en calculant la moyenne des gains des solutions obtenues par ces deux méthodes. Une telle quantité est dans la suite nommée gain moyen.

Dans chacun des tableaux 1 et 2 est indiqué, pour chaque méthode de résolution, le gain moyen par rapport à l'heuristique HLFET. Pour un nombre suffisamment élevé de processeurs, l'heuristique HLFET donne la solution optimale et ce quelque soit le graphe de précédence. La solution optimale est par exemple obligatoirement atteinte si le nombre de processeurs est supérieur ou égal au nombre maximal de tâches qui peuvent être exécutées en parallèle. Il n'est toutefois pas exclu qu'un nombre moins élevé de processeurs conduise à la solution optimale. Ainsi, dans le tableau 1, les résultats sont donnés pour un nombre de processeurs variant seulement de

TABLEAU 1

Gain moyen par rapport à l'heuristique HLFET (étude en fonction du nombre de processeurs)

Nombre de processeurs Nombre de graphes étudiés	2 320	3 298	4 233	5 143	6 70
Hdesc	0,41	0,74	1,28	1,50	2,12
Hrecuit	0,48	0,77	1,34	1,53	**2,59**
Hkang	0,42	0,77	1,38	1,63	**2,59**
Hdescsuc 20000 it.	0,42	0,86	1,34	1,68	**2,59**
Hdescsuc 50000 it.	0,42	0,76	1,42	**1,71**	**2,59**
HRdescsuc 20000 it.	**0,50**	0,89	1,45	1,60	**2,59**
HRdescsuc 50000 it.	0,44	**1,13**	**1,61**	1,63	**2,59**

TABLEAU 2
Gain moyen par rapport à l'heuristique HLFET (étude en fonction du nombre de tâches)

Nombre de tâches Nombre de graphes étudiés	10 80	20 80	50 80	100 80
Hdesc	3,23	1,60	1,14	0,78
Hrecuit	3,44	1,80	1,26	0,78
Hkang	3,23	1,70	1,26	0,80
Hdescsuc 20 000 it.	3,44	1,70	1,29	0,83
Hdescsuc 50 000 it.	3,44	1,80	1,45	0,83
HRdescsuc 20 000 it.	4,30	**3,88**	1,49	**0,85**
HRdescsuc 50 000 it.	**4,62**	3,16	**1,73**	0,84

2 à 6. Les résultats obtenus pour un nombre plus élevé de processeurs ne sont pas indiqués car le nombre de graphes pour lesquels l'heuristique HLFET ne donne pas la solution optimale n'est alors pas jugé suffisant. Dans le tableau 2, les résultats sont calculés en fonction du nombre de tâches (10, 20, 50 ou 100). Dans chaque tableau, nous indiquons également le nombre de graphes étudiés par colonnes.

d. Interprétation des résultats

La moyenne des écarts relatifs de l'heuristique HLFET par rapport à la borne inférieure B (indépendamment du nombre de tâches et du nombre de processeurs) est de l'ordre de 6 %. Toutes les variantes du recuit simulé sont toutefois plus performantes que l'heuristique HLFET. La méthode donnant les meilleurs résultats est la méthode HRdescsuc (50 000 itérations). La moyenne des écarts relatifs de cette méthode par rapport à la borne inférieure B est de l'ordre de 4 %, soit un gain par rapport à l'heuristique HLFET de l'ordre de 2 %. Cette amélioration peut paraître peu importante mais ce résultat doit être nuancé par le fait que la borne inférieure B n'est pas forcément atteinte.

Le deuxième critère, intervenant dans la définition de la borne B, ne peut être atteint que si tous les processeurs sont saturés. Cependant, les contraintes de précédence entre tâches peuvent interdire une telle situation, par exemple s'il existe une ou plusieurs tâches qui ne peuvent pas être réalisées en parallèle avec d'autres tâches. Il serait donc plus judicieux de considérer comme borne inférieure le maximum entre :

– la longueur du chemin critique du graphe de précédence,

– la somme des deux quantités suivantes :

• la somme des durées des tâches qui ne peuvent pas être réalisées en parallèle avec d'autres tâches,

• le rapport des durées d'exécution des tâches restantes sur le nombre de processeurs.

Une telle borne inférieure n'a cependant pas été retenue car elle nécessite une analyse détaillée du graphe de précédence afin de déterminer quelles sont les tâches qui ne peuvent pas être réalisées en parallèle avec d'autres tâches.

Même si le gain des différentes variantes du recuit simulé par rapport à l'heuristique HLFET est faible, il est toutefois important de remarquer que, dans 90 % des cas, une amélioration est apportée à la solution construite par l'heuristique HLFET. Nous constatons d'ailleurs que cette amélioration conduit à une solution optimale (*i.e.* de makespan égal à la borne B) dans 20 % des cas.

Pour l'ensemble des méthodes proposées, le gain moyen par rapport à l'heuristique HLFET augmente avec le nombre de processeurs tant que celui-ci reste faible (cf. Tableau 1). Par contre, plus le nombre de tâches est important et plus le gain moyen par rapport à l'heuristique HLFET diminue (cf. Tableau 2). Notons cependant que la différence moyenne de makespan entre les solutions trouvées et celles construites par l'heuristique HLFET croît lorsque le nombre de tâches augmente. Cette croissance est toutefois plus faible que celle des durées d'exécution (induite par l'augmentation du nombre de tâches) ce qui explique la diminution du gain. La figure 3 présente les variations du gain moyen et de la différence moyenne de makespan en fonction du nombre de tâches pour la méthode HRdescsuc (50 000 itérations).

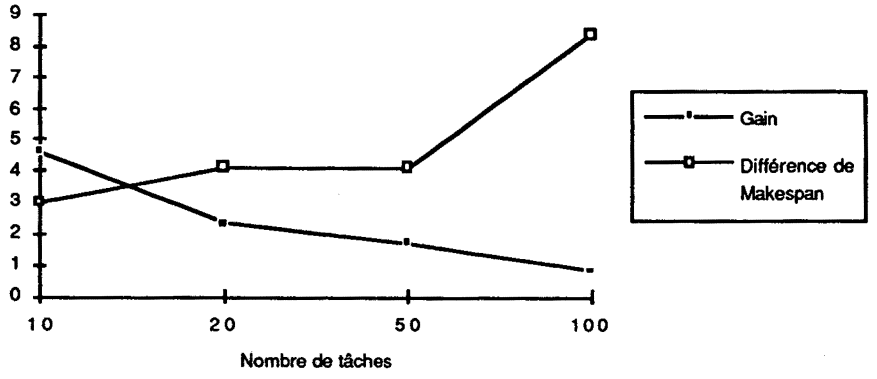

Figure 3. – Variations du gain moyen et de la différence moyenne de makespan en fonction du nombre de tâches pour la méthode HRdescsuc (50 000 itérations).

La substitution de l'heuristique RANDOM à l'heuristique HLFET dans les méthodes Hdesc, Hrecuit et Hkang a été envisagée. Après modification, ces méthodes ne donnaient pas de bons résultats et étaient même en moyenne légèrement moins performantes que l'heuristique HLFET. Dans certains cas, elles donnaient cependant des solutions proches de la borne inférieure aussi n'avons nous pas exclu la mise en œuvre de l'heuristique RANDOM dans les méthodes de descentes successives (cf. méthode HRdescsuc).

Toutes ces méthodes de résolution permettent d'étudier les variations du makespan en fonction du nombre de processeurs. Il est en particulier possible de déterminer le nombre minimum de processeurs au delà duquel aucune amélioration du makespan n'est possible. Ces méthodes permettent également de déterminer quel est le nombre minimal de processeurs qu'il est nécessaire d'allouer à un programme afin d'obtenir un gain de temps donné (30 % ou 50 %...). Cette approche est très intéressante si l'on se place dans un contexte de multiprogrammation, dans lequel plusieurs programmes peuvent être exécutés simultanément, car elle permet d'avoir une idée du nombre de processeurs qu'il est pertinent d'allouer à chaque programme.

Deux modèles ont été élaborés pour chacune des méthodes de résolution : l'un dans le langage de simulation proposé par le logiciel QNAP2 [28], l'autre dans le langage de programmation Fortran. L'intérêt de ces deux modèles est qu'ils se valident mutuellement. Le générateur de nombres aléatoires étant le même [20], les ordonnancements retournés sont identiques bien que les implantations des deux modèles soient différentes. Le tableau 3 présente l'ordre de grandeur des temps de calcul des différents modèles Fortran (station de travail Unix équipée d'un processeur 68040 cadencé à 25 MHz). Le modèle QNAP2 [15] induit des temps de calcul qui sont en moyenne 90 fois supérieurs à ceux du modèle Fortran. Notons cependant

TABLEAU 3
Ordre de grandeur des temps de calcul induits par le modèle Fortran

Méthode	10 tâches	20 tâches	50 tâches	100 tâches
Hdesc 5 000 it.	2 s	4 s	10 s	40 s
Hrecuit, Hkang 20 000 it.	8 s	16 s	40 s	3 mn
Hdescsuc 20 000 it.	8 s	15 s	50 s	3 mn
Hdescsuc 50 000 it.	20 s	40 s	2 mn	7 mn
HRdescsuc 20 000 it.	10 s	20 s	1 mn	4 mn
HRdescsuc 50 000 it.	30 s	50 s	3 mn	9 mn

que le modèle QNAP2 présente l'avantage de valider le modèle Fortran et d'être très facilement adaptable au cas stochastique.

4. ORDONNANCEMENTS STOCHASTIQUES

Les hypothèses de l'étude sont identiques à celles du cas déterministe sauf celle concernant les durées d'exécution des tâches, qui sont à présent modélisées par des lois exponentielles indépendantes de paramètre différent pour chaque tâche. La résolution du problème d'ordonnancement n'a donc plus pour objectif la détermination d'un ordonnancement optimal ou proche de l'optimum mais celle d'un ordonnancement de durée moyenne minimale. En effet, bien que la notion d'algorithme optimal ou de politique d'ordonnancement optimale ait toujours un sens en stochastique, la notion d'ordonnancement optimal n'est, quant à elle, plus adaptée.

Déterminer la durée moyenne d'un ordonnancement revient à supposer que l'on dispose de jobs identiques qui sont exécutés les uns après les autres. Le problème est alors de calculer le temps moyen de réponse d'un job.

Dans ce contexte, un ordonnancement est caractérisé par un ensemble de triplets $\{(j, Pr_j, I_j),\ 1 \leqslant j \leqslant n\}$ qui vérifie les contraintes de précédence entre tâches (la tâche j et la I_j-ième tâche à être affectée sur le processeur Pr_j. En d'autres termes, I_j désigne l'ordre d'introduction de la tâche j sur le processeur Pr_j). Le *makespan* d'un tel ordonnancement désigne la durée moyenne d'exécution de l'ensemble de toutes les tâches.

Dans une première partie, nous présentons un bref état de l'art des acquis théoriques sur le comportement des algorithmes d'ordonnancement par liste en stochastique. Dans une seconde partie, nous proposons deux modèles permettant d'évaluer le makespan d'un ordonnancement stochastique: un modèle markovien et un modèle de simulation. Ayant remarqué que pour un graphe de tâches donné, le meilleur ordonnancement trouvé pour le modèle déterministe n'induisait pas sur le modèle stochastique une durée moyenne d'exécution minimale, nous avons ensuite décidé d'adapter l'algorithme du recuit simulé au modèle stochastique. La méthode que nous proposons, nommée méthode hybride, est décrite dans la troisième partie. Une approche duale, proposée par [12], est présentée dans la quatrième partie. Cette approche diffère quelque peu de la précédente dans la mesure où l'objectif fixé ne concerne plus la minimisation de la durée moyenne mais la maximisation de la probabilité que l'ordonnancement trouvé soit meilleur que

les autres ordonnancements possibles. Une cinquième partie est consacrée à la comparaison de ces deux approches.

4.1. Quelques résultats théoriques

De nombreux auteurs se sont intéressés à l'étude du comportement des algorithmes d'ordonnancement par liste en stochastique. Nous allons décrire brièvement quelques uns de ces résultats.

L'ordonnancement sur 2 processeurs de n tâches soumises à des contraintes de précédence du type « anti-arborescence » a été abondamment étudié. Sous ces hypothèses, l'heuristique HLFET est optimale lorsque les durées des tâches sont des variables aléatoires indépendantes et identiquement distribuées (i.i.d.) selon une loi exponentielle, que ce soit dans le cas préemptif ou dans le cas non préemptif [5]. Ce résultat reste valable, dans le cas non préemptif, lorsque les durées des tâches sont i.i.d. selon une loi quelconque [2]. Les travaux de [27] et [13] généralisent ce résultat à des problèmes où les durées des tâches ne sont pas toutes i.i.d. [27] prouve que l'heuristique HLFET est optimale lorsque toutes les tâches situées à un même niveau dans l'anti-arborescence ont des durées i.i.d. selon une loi exponentielle (deux tâches situées à des niveaux différents peuvent avoir des durées qui suivent des lois exponentielles de paramètres différents). Ce résultat est vérifié dans le cas préemptif comme dans le cas non préemptif. Sous les mêmes hypothèses, [13] propose un algorithme optimal, basé sur l'heuristique HLFET, pour la prise en compte de lois autres que la loi exponentielle.

Les travaux de [9] concernent l'ordonnancement stochastique de forescences sur un ensemble de processeurs identiques. Les durées des tâches sont i.i.d. selon une loi exponentielle et la préemption de tâches est autorisée. Un algorithme glouton, analogue à l'heuristique HLFET, est prouvé optimal pour une classe particulière de forescences lorsque le nombre de processeurs est égal à 2. Ce résultat est étendu à un nombre arbitraire de processeurs lorsque des hypothèses restrictives supplémentaires sont imposées sur les forescences.

L'ordonnancement stochastique de tâches sur des processeurs uniformes dont la courbe de disponibilité est variable et non obligatoirement connue *a priori* a également été étudié [21]. Là encore, les durées des tâches sont i.i.d. selon une loi exponentielle et la préemption de tâches est autorisée. Les auteurs montrent que différents algorithmes d'ordonnancement par liste sont optimaux lorsque le graphe de précédence est une forescence, une

anti-forescence ou un graphe d'intervalles. Ces résultats généralisent ceux de [5], [2] et [8].

L'étude de ces différents travaux nous montre que les algorithmes d'ordonnancement par liste sont optimaux pour de nombreux cas particuliers. Nous avons donc décidé de les retenir comme algorithmes de référence pour l'étude de notre problème d'ordonnancement.

4.2. Évaluation d'ordonnancements stochastiques

Le principe de l'heuristique HLFET repose sur la construction d'une liste de tâches dans laquelle les tâches réalisables sont classées par niveaux décroissants. Dans le modèle stochastique, le calcul du niveau de chaque tâche ne peut être réalisé qu'à partir des valeurs moyennes des durées d'exécution de ces tâches, les valeurs instantanées n'étant pas connues *a priori*. Bien que le niveau des tâches soient identiques quelle que soit l'exécution, les ordonnancements obtenus lors de deux exécutions successives peuvent toutefois être différents dans la mesure où les variations liées aux durées d'exécution peuvent entraîner des modifications au niveau de l'ordre d'exécution des tâches et de leur affectation aux processeurs. Ce type de démarche correspond donc à l'évaluation de l'efficacité de l'heuristique HLFET dans un contexte d'allocation dynamique. Afin de nous ramener aux problèmes d'allocation statique, nous avons dû envisager un autre type d'approche. Nous avons ainsi décidé d'évaluer l'efficacité de différents ordonnancements construits en déterministe et imposés en stochastique.

Imposer en stochastique un ordonnancement construit « hors ligne » implique que, quelle que soit l'exécution, chaque tâche est toujours affectée au même processeur et l'ordre d'exécution des tâches est toujours le même. Le processus retenu est le suivant :

– À l'aide d'une méthode de résolution quelconque, le modèle déterministe construit un ordonnancement. Il détermine donc, pour chaque tâche, le processeur qui lui est alloué et son ordre d'introduction sur ce processeur.

– À partir de ces résultats, le modèle stochastique estime la durée moyenne de l'ordonnancement. Deux modèles sont proposés : un modèle de simulation stochastique et un modèle markovien.

4.2.1. Modèle de simulation stochastique

L'élaboration d'un modèle de simulation stochastique est une première façon d'évaluer le makespan des ordonnancements obtenus. Plusieurs simulations statistiquement indépendantes sont successivement réalisées. La moyenne des différentes durées d'exécution des ordonnancements produits

constitue alors un estimateur du makespan. Chaque simulation reproduit un comportement différent du modèle (les dates de début et de fin d'exécution d'une tâche diffèrent d'une simulation à l'autre) car un nouveau flot de nombres aléatoires est généré à chaque simulation. Le nombre de simulations effectuées doit donc être suffisamment grand pour assurer un bon échantillonnage statistique du comportement stochastique du modèle.

4.2.2. Analyse markovienne

L'élaboration d'un modèle markovien permet également d'évaluer la durée moyenne d'un ordonnancement stochastique (résultats exacts). Le principe de ce modèle est le suivant: étant donnés un graphe de précédence et un ordonnancement (on connaît le processeur alloué à chaque tâche et l'ordre de passage des tâches sur les processeurs), l'exploitation du modèle markovien permet:

– la construction de la chaîne de Markov associée,

– l'évaluation des performances, à l'état stationnaire, de l'ordonnancement stochastique.

Le *vecteur d'état* E de la chaîne de Markov de ce modèle comporte trois ensembles disjoints deux à deux: $E = A \cup B \cup C$, $\text{Card}(A) + \text{Card}(B) + \text{Card}(C) = n$, $1 \leqslant \text{Card}(A) \leqslant m$, $1 \leqslant \text{Card}(B) \leqslant n$, $1 \leqslant \text{Card}(C) \leqslant n$.

– A est *l'ensemble des tâches en cours d'exécution:*

$A = \{k_i,\ i = 1, 2, \ldots, \text{Card}(A)/k_i \in T\}$.

– B est *l'ensemble des tâches non encore réalisées:*

$B = \bigcup_{i=1,m} B_i$ avec $B_i = \{l_j,\ j = 1, 2, \ldots, \text{Card}(B_i)/l_j \in T,\ Pr_{l_j} = i,$
$\forall 1 \leqslant j \leqslant \text{Card}(B_i) - 1,\ I_{l_j} < I_{l_{j+1}}\}$.

B_i est l'ensemble ordonné des tâches non encore réalisées qui doivent être affectées au processeur i (la notation $I_{l_j} < I_{l_{j+1}}$ signifie que la tâche l_j doit être réalisée avant la tâche l_{j+1} sur le processeur i).

– C est *l'ensemble des tâches achevées:*

$C = \{m_k,\ k = 1, 2, \ldots, \text{Card}(C)/m_k \in T\}$.

Les *changements d'états* de la chaîne de Markov sont dus à l'achèvement de l'exécution d'une des tâches de A. Si le processeur Pr_{k_t} est libéré par la tâche k_t $(1 \leqslant t \leqslant \text{Card}(A))$,

– La tâche k_t est déplacée de l'ensemble A vers l'ensemble C c'est-à-dire :

$$A = A\backslash\{k_t\}; \qquad C = C \cup \{k_t\}.$$

– Pour tous les processeurs z non actifs $(z \notin \{Pr_{k_i},\ k_i \in A\backslash\{k_t\}\})$, on vérifie si la tâche $l_1 \in B_z$ (la prochaine susceptible d'être exécutée sur le processeur z) a tous ses prédécesseurs qui sont achevés. Si tel est le cas, on place la tâche l_1 dans l'ensemble A. Sinon, on ne fait rien (une tâche l_j $(j > 1)$ peut éventuellement être réalisable sur z mais comme l'ordre d'exécution des tâches impose qu'elle ne soit affectée au processeur z qu'après l'exécution de la tâche l_1, son affectation sur le processeur n'est pas envisagée. Ceci revient donc à laisser le processeur libre même si certaines tâches sont réalisables).

Soit $D(k_t)$ l'ensemble (éventuellement vide) des tâches qui sont affectées à un processeur suite à l'achèvement de la tâche k_t.

$$D(k_t) = \bigcup_{z \notin \{Pr_{k_i},\, k_i \in A\backslash\{k_t\}\}} (l_1/Pr_{l_1} = z,\ \mathrm{Pred}(l_1) \subset C) \text{ avec } \mathrm{Pred}(l_1)$$

ensemble des prédécesseurs de la tâche l_1

$$A = A \cup D(k_t); \qquad B = B\backslash D(k_t).$$

L'achèvement de la tâche k_t fait donc passer le système de l'état $E = A \cup B \cup C$ à l'état E_t tel que

$$E_t = A_t \cup B_t \cup C_t = (A \cup D(k_t)\backslash\{k_t\}) \cup (B\backslash D(k_t)) \cup (C \cup \{k_t\}).$$

Le taux de transition de l'état E à l'état E_t est égal à l'inverse de la durée moyenne de la tâche k_t $(1/p_{k_t})$.

Puisqu'il y a $K = \mathrm{Card}(A)$ processeurs actifs, il y a K états accessibles à partir de l'état $E = A \cup B \cup C$.

L'ensemble de ces états est donné sur la figure 4.

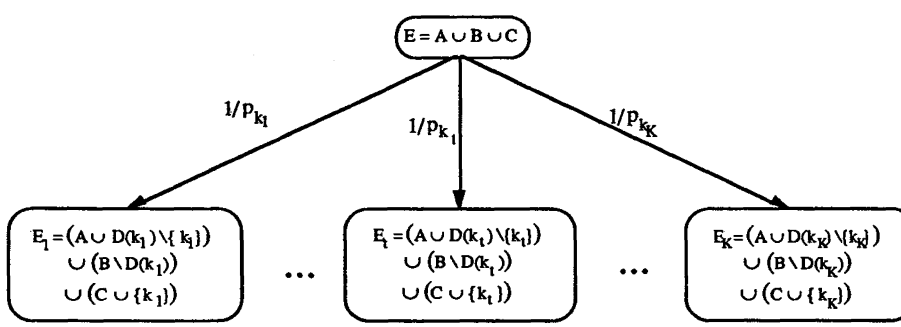

Figure 4. – Ensemble des états accessibles à partir de l'état $E = A \cup B \cup C$.

L'état initial (choisi pour des raisons de commodité) est tel que :

– Card $(C) = 0$ (aucune tâche n'est achevée),

– Card $(A) = \min(k, m)$ et Card $(B) = n - \min(k, m)$ s'il existe k tâches réalisables initialement.

Ce modèle est implanté avec le logiciel QNAP2 qui comporte une méthode d'analyse markovienne. Quel que soit le graphe de tâches, la chaîne de Markov est construite et les performances de l'ordonnancement à l'état stationnaire sont calculées. La durée moyenne de l'ordonnancement est déduite mais une évaluation du système multiprocesseur est également fournie grâce au calcul du taux d'occupation des processeurs. Notons que ce processus de modélisation est proche de celui retenu par [23] et [22], qui consiste à spécifier le problème à l'aide d'un réseau de Petri stochastique et à construire la chaîne de Markov homogène associée au graphe des marquages accessibles de ce réseau de Petri afin de déduire les critères de performance du problème.

4.2.3. Résultats

Ces deux modèles ont été testés sur différents exemples considérés en déterministe. Le tableau 4 reporte les résultats obtenus pour trois de ces exemples comportant respectivement 10, 20 et 50 tâches.

TABLEAU 4

Comparaison du modèle markovien et du modèle de simulation stochastique.

Ex	Méthode	Nb. Proc.	Déterministe	Modèle markovien		Modèle de simulation	
				Durée moyenne	Nb états	Durée moyenne	Intervalle de confiance
1	HLFET	2	117	142,40	18	142,50 ± 0,21	
(10	Hdesc	2	113	137,30	19	137,50 ± 0,21	
tâches)	HRdescsuc	2	113	137,30	19	137,50 ± 0,21	
2	HLFET	2	75	89,53	69	89,51 ± 0,10	
(20		3	66	76,91	148	76,94 ± 0,09	
tâches)	Hdesc	2	73	95,74	45	95,75 ± 0,11	
		3	66	81,26	104	81,27 ± 0,10	
	HRdescsuc	2	70	90,37	65	90,37 ± 0,10	
		3	66	78,59	134	78,62 ± 0,10	
3	HLFET	2	277	331,60	197	331,40 ± 0,27	
(50		3	194	258,00	676	257,80 ± 0,24	
tâches)	Hdesc	2	274	325,40	231	325,20 ± 0,28	
		3	191	260,60	657	260,50 ± 0,24	
	HRdescsuc	2	267	328,80	201	328,70 ± 0,30	
		3	190	264,80	643	264,60 ± 0,26	

Pour chaque exemple, trois ordonnancements sont évalués : ceux construits par l'heuristique HLFET, la méthode de descente stochastique Hdesc et la méthode des descentes successives HRdescsuc (50 000 itérations). La durée moyenne de chaque ordonnancement est calculée d'une part par le modèle markovien, d'autre part par le modèle de simulation. La durée d'exécution de chaque ordonnancement en déterministe est également mentionnée.

Pour le modèle markovien, le nombre d'états de la chaîne Markov est indiqué.

Pour le modèle de simulation, un intervalle de confiance de la durée moyenne de chaque ordonnancement est calculé grâce à la méthode des points de régénération [28]. Ce calcul est effectué pour 400 000 simulations successives, un nombre inférieur de simulations entraînant une surestimation de la durée moyenne des ordonnancements.

Les résultats retournés par le modèle markovien sont toujours situés dans les intervalles de confiance calculés par la simulation. Nous pouvons donc considérer que ces modèles se valident mutuellement. La méthode d'analyse markovienne est intéressante car, d'une part, elle donne des résultats exacts à l'état stationnaire et, d'autre part, elle est moins coûteuse en temps que la simulation pour les applications induisant un nombre d'états peu élevé. Il est malheureusement difficile, dans notre cas, d'estimer *a priori* le nombre d'états de la chaîne de Markov pour un graphe donné. Ce nombre d'états dépend, d'une part, du nombre de tâches et du nombre de processeurs et, d'autre part, du degré de parallélisme du graphe (plus le nombre de tâches parallèles est élevé et plus le nombre d'états est grand). Notons que ces deux modèles permettent également d'évaluer le taux d'occupation des processeurs. Les taux d'occupation calculés par le modèle markovien sont également situés dans les intervalles de confiance déterminés par la simulation.

Bien que pratiquement identiques, les résultats fournis par l'analyse markovienne et par simulation diffèrent de ceux obtenus pour le modèle déterministe. Cette différence est d'autant plus grande que le nombre de processeurs est élevé. À titre de vérification, on constate toutefois que, lorsqu'on substitue dans le modèle markovien des lois constantes aux lois exponentielles, les durées d'exécution deviennent proches de celle du modèle déterministe. L'écart restant résulte seulement de l'approximation faite, dans le logiciel QNAP2, de la loi constante par une loi d'Erlang d'ordre cinq. L'augmentation de l'ordre de la loi d'Erlang diminue cet écart mais le nombre d'états de la chaîne de Markov devient alors très vite prohibitif.

Finalement, on constate que pour un graphe donné, l'ordonnancement qui induit la durée d'exécution la plus faible en déterministe n'engendre pas obligatoirement la durée moyenne d'exécution la plus faible en stochastique. Cette constatation nous a donc amenés à adapter le recuit simulé au modèle stochastique dans le but de minimiser la durée moyenne des ordonnancements.

4.3. Méthode hybride

Cette méthode repose sur la combinaison de la descente stochastique, de la simulation et de la méthode d'analyse markovienne. Son principe est de construire différents ordonnancements déterministes selon la stratégie de construction des voisins décrite précédemment et de les évaluer en stochastique à l'aide du modèle markovien présenté dans le paragraphe 4.2.2. Le mécanisme d'acceptation d'un voisin est le suivant: un ordonnancement est accepté si sa durée moyenne d'exécution est inférieure à celle de l'ordonnancement courant.

Cet algorithme est itératif. Chaque itération peut être décrite de la façon suivante:

– Étant donné un ordonnancement courant OC de durée moyenne $C(OC)$.

– Un ordonnancement OV voisin de OC est construit (cf. mécanisme de voisinage du modèle déterministe). Chaque tâche se voit donc allouer un processeur ainsi qu'un ordre de passage sur ce processeur.

– Ces données sont transmises au modèle markovien, présenté précédemment, qui évalue alors la durée moyenne $C(OV)$ de l'ordonnancement OV.

– Si $C(OV) \leq C(OC)$ alors OV devient ordonnancement courant.

Ce processus est illustré sur la figure 5.

[12] propose un autre algorithme permettant de résoudre le problème d'ordonnancement stochastique. Cet algorithme, nommé méthode des descentes stochastiques itérées, a pour objectif la maximisation de la probabilité que l'ordonnancement trouvé soit meilleur que les autres ordonnancements possibles.

4.4. Méthode des descentes stochastiques itérées

Cette méthode est une adaptation de la descente stochastique au problème d'ordonnancement stochastique. Son principe est de prendre en compte différents ordonnancements déterministes et de les comparer en stochastique. Le mécanisme d'acceptation d'un voisin est le suivant: un ordonnancement voisin est retenu si, à l'issue de N comparaisons avec l'ordonnancement

courant, il est meilleur dans la majorité des cas. Pour chacune de ces N comparaisons, les durées des tâches de l'ordonnancement voisin et de l'ordonnancement courant sont identiques et constantes (elles sont issues de tirages pseudo-aléatoires basés sur les distributions exponentielles associées à chaque tâche). Pour chaque comparaison, l'évaluation des makespans est donc effectuée à l'aide d'un modèle de simulation déterministe.

Cette méthode est itérative. Chaque itération peut être décrite de la façon suivante :

– Étant donné un ordonnancement courant OC, on génère un ordonnancement voisin OV (cf. mécanisme de voisinage du cas déterministe).

– Une durée constante, issue d'un tirage aléatoire selon une loi exponentielle, est affectée à chaque tâche. Les makespans des ordonnancements OV et OC sont alors déterminés par le modèle de simulation déterministe finie.

Cette étape est réalisée pour N jeux de valeurs différents. Le nombre de fois p_N où le makespan de OV est strictement inférieur à celui de OC est comptabilisé.

– Si $2^* p_N \geqslant N$ alors OV devient ordonnancement courant.

Ce processus est illustré sur la figure 6.

4.5. Comparaison des deux méthodes

Ces deux méthodes sont appliquées sur une trentaine de graphes de précédence considérés en déterministe qui comportent 10, 20 et 50 tâches. Le nombre de processeurs est limité à deux. Un nombre plus élevé de processeurs pourrait être pris en compte mais n'a toutefois pas été considéré car les méthodes proposées n'entraînent que peu d'amélioration par rapport à l'heuristique HLFET pour la plupart des graphes traités. En effet, le nombre de tâches étant faible, l'heuristique HLFET conduit très fréquemment à la solution optimale dans le cas de 3 processeurs ou plus (en particulier pour les graphes comportant 10 tâches).

Pour chacune des méthodes, la solution initiale est obtenue par application de l'heuristique HLFET mais l'utilisation d'une autre heuristique, telle que RANDOM, n'est pas exclue.

Les ordonnancements stochastiques sont caractérisés par le processeur affecté à chaque tâche et l'ordre d'exécution des tâches sur les processeurs.

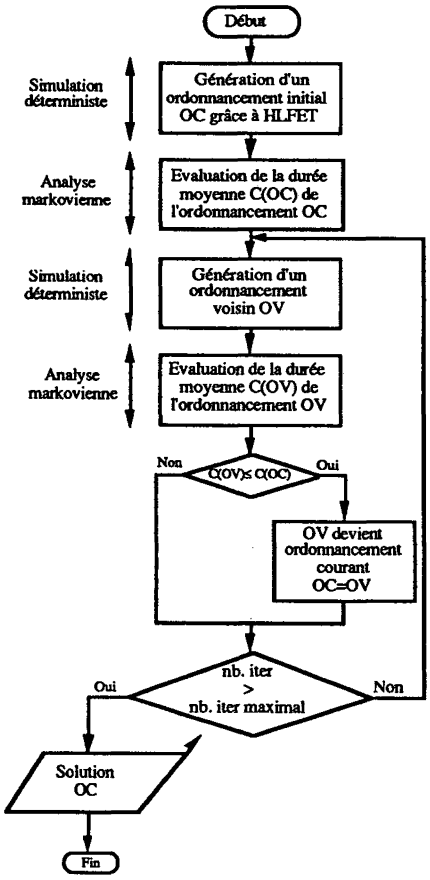

Figure 5. – Principe de la méthode hybride

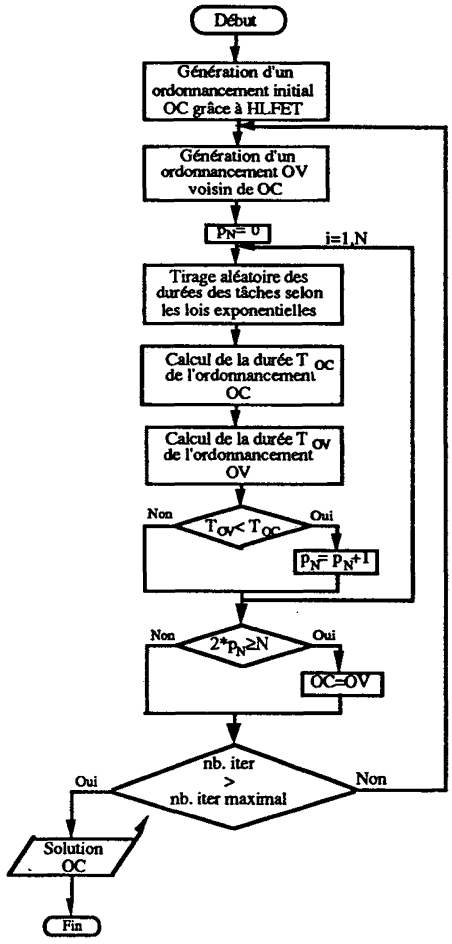

Figure 6. – Principe de la méthode des descentes stochastiques itérées.

Afin de comparer les ordonnancements produits par ces deux méthodes, deux critères d'évaluation sont retenus :

– Le premier consiste à évaluer le makespan de ces ordonnancements à l'aide du modèle markovien présenté au paragraphe 4.2.2.

– Le second permet d'estimer grossièrement la probabilité qu'un ordonnancement soit meilleur qu'un autre. Pour cela, les makespans des ordonnancements obtenus par la méthode hybride et par la méthode des descentes stochastiques itérées sont comparés sur 1 000 jeux de valeurs (Pour chaque jeu de valeurs, des durées constantes, issues de tirages selon des

distributions exponentielles, sont affectées aux tâches). Sur ces 1 000 jeux de valeurs sont comptabilisées:

- le nombre de fois où les deux makespans sont égaux,
- le nombre de fois où le makespan de l'ordonnancement produit par la méthode des descentes itérées est inférieur à celui fourni par la méthode hybride (et vice versa).

Chacun de ces ordonnancements est également comparé de façon analogue avec l'ordonnancement produit par l'heuristique HLFET.

Dans la méthode hybride, le nombre d'itérations est fixé à 1 000, aucune amélioration ne se produisant généralement au delà. Dans la méthode des descentes stochastiques itérées, différentes valeurs sont affectées à N: 10, 25 et 50. Les nombres d'itérations correspondant sont fixés à 5 000, 2 500 et 1 000, de telles valeurs permettant d'avoir des durées d'exécution équivalentes.

Le tableau 5 (resp. 6,7) présente les durées moyennes d'exécution des ordonnancements obtenus par l'heuristique HLFET, par la méthode hybride et par la méthode des descentes stochastiques itérées (pour $N = 10$, 25 et 50) pour un nombre de tâches égal à 10 (resp. 20, 50). Pour la méthode des descentes stochastiques itérées, les trois dernières colonnes reportent les résultats obtenus suite à une modification apportée au mécanisme d'acceptation des voisins (cette modification sera décrite ultérieurement). Les nombres apparaissant en gras dans ces trois tableaux mentionnent les améliorations obtenues par rapport à l'heuristique HLFET.

La *méthode hybride* induit une amélioration moyenne de 2,18 % par rapport à l'heuristique HLFET. Cette amélioration est d'autant plus grande que le nombre de tâches est élevé (1,15 % pour 10 tâches, 2,21 % pour 20 tâches et 3,17 % pour 50 tâches). Dans tous les cas, l'ordonnancement obtenu est de makespan inférieur ou égal à celui construit par l'heuristique HLFET. Cette méthode présente l'avantage de travailler sur des valeurs exactes à l'état stationnaire. Elle est malheureusement d'application assez restreinte car elle entraîne des temps de calcul importants. Pour 1 000 itérations (1 000 activations du modèle markovien), ces durées sont de l'ordre de 15 minutes pour 10 tâches et de l'ordre de 1 heure 50 minutes pour 20 tâches sur station de travail UNIX équipée d'un processeur 68040 cadencé à 25 MHz. L'application de cette méthode sur un graphe comportant 100 tâches n'est donc pas envisageable actuellement.

La descente stochastique ayant comme particularité de converger vers des minima locaux, il peut sembler intéressant de remplacer cette méthode

TABLEAU 5
Comparaison de la méthode hybride et de la méthode des descentes stochastiques itérées (10 tâches).

Ex	HLFET	Méthode hybride	Méthode des descentes stochastiques itérées					
			Avant modification			Après modification		
			$N=10$	$N=25$	$N=50$	$N=10$	$N=15$	$N=25$
1	55,04	**52,99**	**52,99**	**53,88**	**52,99**	**52,99**	**52,99**	**52,99**
2	72,12	**70,13**	72,40	72,12	**71,76**	72,12	**71,64**	**71,76**
3	159,40	159,40	159,40	159,40	159,40	159,40	159,40	159,40
4	61,25	61,25	61,25	61,25	61,25	61,25	61,25	61,25
5	75,64	75,64	75,64	75,64	75,64	75,64	75,64	75,64
6	142,40	**137,20**	142,40	**137,20**	**137,20**	**138,10**	**141,00**	142,40
7	53,24	**52,19**	**52,19**	53,24	53,24	54,12	**52,19**	**52,19**
8	68,38	**66,93**	**67,73**	**66,93**	**66,93**	**66,93**	**66,93**	68,38
9	134,90	**134,70**	**134,70**	**134,70**	134,90	134,90	**134,70**	**134,70**
10	51,31	51,31	52,88	52,48	51,31	51,31	51,31	51,31
11	56,00	56,00	56,70	56,00	56,00	56,53	56,00	56,00
12	161,90	161,90	168,80	161,90	161,90	161,90	162,60	162,60
13	65,08	**64,44**	**64,02**	65,08	65,08	**64,44**	65,08	**64,44**
14	104,30	**102,30**	**102,30**	**102,40**	**102,40**	**102,40**	**102,30**	**102,30**
15	147,60	147,60	147,60	147,60	147,60	147,60	147,60	147,60
Gain moyen /HLFET		**1,15 %**	**0,09 %**	**0,50 %**	**0,78 %**	**0,60 %**	**0,74 %**	**0,58 %**

par d'autres variantes du recuit simulé telle que la méthode des descentes successives. Bien que cette substitution soit très attrayante et facile à mettre en œuvre, elle n'a toutefois pas été envisagée car de telles méthodes requièrent un nombre d'itérations plus élevé que la descente stochastique et par conséquent des temps de calcul plus importants.

Les modèles markoviens induisant des temps de calcul élevés, il serait intéressant de les remplacer par des modèles de simulation. Dans les modèles de simulation, il est malheureusement difficile de connaître la date à partir de laquelle le système atteint son état stationnaire. La méthode hybride étant basée sur cet état stationnaire, il serait donc indispensable de prendre en compte les intervalles de confiance pour décider de l'acceptation ou du rejet d'un ordonnancement voisin, comme le signale [3].

Dans la *méthode des descentes stochastiques itérées,* nous avions initialement défini p_N comme étant le nombre de fois où le makespan de

TABLEAU 6
Comparaison de la méthode hybride et de la méthode des descentes stochastiques itérées (20 tâches).

Ex	HLFET	Méthode hybride	Méthode des descentes stochastiques itérées					
			Avant modification			Après modification		
			$N=10$	$N=25$	$N=50$	$N=10$	$N=15$	$N=25$
1	93,55	**90,85**	**92,78**	93,55	**92,73**	**90,35**	**90,88**	**90,88**
2	121,40	**116,40**	**116,10**	126,10	121,40	**120,90**	121,60	121,40
3	212,30	**210,90**	**211,00**	219,80	**211,00**	213,10	212,30	212,30
4	91,80	**88,62**	**90,73**	92,53	**89,06**	**89,38**	**88,73**	**91,39**
5	109,60	**107,80**	**108,00**	109,60	109,60	109,90	**107,50**	**107,80**
6	345,80	**339,80**	**343,80**	**342,10**	**344,30**	**342,50**	**341,70**	**340,30**
7	89,53	**89,41**	89,53	90,23	89,79	89,95	89,53	**89,52**
8	144,70	**144,60**	149,20	144,80	145,00	144,90	144,70	144,70
9	298,60	**293,50**	303,20	**297,00**	**296,00**	299,70	298,60	298,60
10	93,15	**89,90**	95,33	**92,59**	**91,34**	**91,21**	**91,13**	**91,89**
11	161,30	**159,90**	161,50	161,30	161,30	161,30	**160,80**	161,30
12	268,70	**254,20**	268,70	**259,40**	270,10	**255,00**	**266,90**	**268,20**
13	114,60	**112,00**	114,60	**113,00**	**112,40**	**112,40**	**112,40**	**113,00**
14	125,70	**122,00**	125,70	**122,20**	**122,20**	**121,50**	**121,50**	**123,70**
15	370,30	**363,20**	**367,20**	**367,20**	**366,30**	**365,20**	370,30	370,30
Gain moyen /HLFET		**2,21 %**	**0,18 %**	**0,11 %**	**0,83 %**	**1,31 %**	**1,17 %**	**0,73 %**

l'ordonnancement voisin est inférieur ou égal à celui de l'ordonnancement courant. Cependant, la méthode fournissait des résultats moins satisfaisants que l'heuristique HLFET quelle que soit la valeur de N. Ce résultat est intuitivement assez naturel car cette méthode autorisait l'acceptation de nombreuses solutions défavorables. En effet, sur N comparaisons de l'ordonnancement courant et de l'ordonnancement voisin (N supposé pair), l'ordonnancement voisin pouvait être de makespan égal à celui de l'ordonnancement courant $N/2$ fois et de makespan supérieur $N/2$ fois. Un tel voisin est globalement moins satisfaisant que l'ordonnancement courant or la méthode des descentes stochastiques itérées l'acceptait car il était de makespan (inférieur ou) égal au makespan de l'ordonnancement courant dans la majorité des cas.

TABLEAU 7
Comparaison de la méthode hybride et de la méthode des descentes stochastiques itérées (50 tâches).

Ex	HLFET	Méthode hybride	Méthode des descentes stochastiques itérées					
			Avant modification			Après modification		
			$N=10$	$N=25$	$N=50$	$N=10$	$N=15$	$N=25$
1	221,3	**212,6**	**219,4**	**218,6**	**220,7**	221,3	221,3	**220,8**
2	329,3	**319,1**	335,4	330,0	**327,1**	**326,4**	329,3	**327,5**
3	659,0	**642,1**	664,8	661,7	**650,0**	**650,5**	**652,1**	**652,9**
4	234,6	**225,0**	**231,3**	**233,3**	**234,4**	**232,7**	**232,9**	**232,9**
5	331,6	**319,5**	**328,2**	338,1	333,1	**323,6**	**325,7**	**328,7**
6	658,2	**644,9**	677,0	708,3	674,5	**647,0**	**654,8**	**653,7**
7	226,3	**221,3**	233,9	229,1	228,7	**221,7**	**225,2**	226,3
8	365,3	**351,7**	**365,0**	369,5	366,9	**357,5**	**359,6**	**361,2**
9	659,6	**649,1**	665,4	668,0	665,5	**659,1**	**652,1**	**651,7**
10	217,6	**211,1**	220,6	221,4	217,8	**214,0**	**216,8**	**216,4**
11	287,9	**277,7**	292,0	294,2	288,4	**278,4**	**281,8**	**284,6**
12	653,3	**624,6**	659,8	**648,1**	**630,2**	**638,3**	**646,1**	**644,9**
13	208,7	**205,6**	214,2	214,3	216,3	207,5	208,7	208,7
14	350,3	**337,3**	**347,9**	**349,7**	**350,7**	**345,2**	**345,1**	**346,2**
15	612,5	**584,3**	**604,0**	629,7	**609,0**	**598,7**	**606,2**	**595,5**
Gain moyen /HLFET		**3,17 %**	**-0,72 %**	**-1,37 %**	**-0,19 %**	**1,52 %**	**0,89 %**	**0,88 %**

Nous avons ensuite défini p_N comme étant le nombre de fois où l'ordonnancement voisin est strictement inférieur (et non plus inférieur ou égal) à l'ordonnancement courant. Pour des valeurs élevées de N, les ordonnancements construits ont en moyenne un makespan légèrement plus faible que les ordonnancements produits par l'heuristique HLFET: pour $N=50$, le gain moyen par rapport à l'heuristique HLFET est de 0,50 % (cf. méthodes des descentes stochastiques itérées avant modification). Par contre, pour N petit, la méthode des descentes stochastiques itérées entraîne une dégradation des résultats car la méthode autorise la sélection d'un trop grand nombre de solutions défavorables: pour $N=10$ (resp. $N=25$), on note une dégradation de 0,15 % (resp. 0,25 %) par rapport à l'heuristique HLFET.

Comme la méthode des descentes stochastiques itérées demeurait moins performante que la méthode hybride, nous avons dans un deuxième temps modifé l'algorithme de la façon suivante: un ordonnancement voisin n'est accepté que si son makespan est inférieur ou égal au makespan de l'ordonnancement courant pour les N tirages (c'est-à-dire dans la totalité des cas). Cette contrainte étant très forte, les valeurs affectées à N doivent être faibles. Pour $N=10$ (resp. $N=15$, $N=25$), nous avons noté une amélioration de 1,14 % (resp. 0,93 %, 0,73 %) par rapport à l'heuristique HLFET (cf. méthode des descentes stochastiques itérées après modification).

L'intérêt de cette dernière modification est que, d'une part, elle améliore les gains par rapport à l'heuristique HLFET et, d'autre part, elle diminue le temps de calcul de l'algorithme. En effet, dès que le makespan de l'ordonnancement voisin est supérieur au makespan de l'ordonnancement courant lors d'un tirage, un nouveau voisin est considéré; les N tirages ne sont plus systématiquement réalisés comme précédemment. Avant cette modification, les temps de calcul de la méthode des descentes stochastiques itérées était de l'ordre de 50 minutes pour 10 tâches, de l'ordre de 2 heures pour 20 tâches et de l'ordre de 9 heures pour 50 tâches. Après la modification, ces temps de calcul sont de l'ordre de 15 minutes pour 10 tâches, de l'ordre de 35 minutes pour 20 tâches et de l'ordre de 3 heures pour 50 tâches.

En conclusion, nous signalons que la méthode hybride est plus efficace que la méthode des descentes stochastiques itérées et que le gain moyen par rapport à l'heuristique HLFET augmente en fonction du nombre de tâches. Cette dernière méthode ne doit cependant pas être abandonnée car, d'une part, elle fournit des améliorations par rapport à l'heuristique HLFET et, d'autre part, son application n'est pas restreinte par le nombre de tâches du jeu d'essai considéré.

5. CONCLUSION

Nous avons présenté dans cet article différentes méthodes de résolution du problème d'ordonnancement de tâches sur une architecture multiprocesseur à mémoire partagée. Nous avons distingué deux types de problèmes d'ordonnancement: les ordonnancements déterministes lorsque les durées d'exécution des tâches sont constantes et les ordonnancements stochastiques lorsque ces durées sont approchées par une loi de probabilité (en l'occurrence la loi exponentielle).

Dans le cas déterministe, nous avons proposé différentes méthodes de résolution fondées sur les algorithmes d'ordonnancement par liste et la

méthode du recuit simulé et ses variantes. Ces méthodes ont ensuite été adaptées au traitement du problème d'ordonnancement stochastique.

Cette étude a bien sûr nécessité la construction de différents modèles afin d'évaluer l'efficacité d'un ordonnancement. Ces modèles permettent non seulement d'évaluer la durée d'un ordonnancement mais également d'obtenir des critères de performance relatifs au système multiprocesseur considéré, tel que le taux d'occupation des processeurs. Parmi les modèles élaborés, nous distinguons des modèles de simulation déterministe ou stochastique et un modèle markovien.

L'ensemble de ces méthodes fournit une amélioration d'environ 2 % par rapport à l'heuristique HLFET dans le cas déterministe comme dans le cas stochastique. Cette amélioration peut paraître peu importante mais ce résultat doit toutefois être nuancé par le fait que l'heuristique HLFET produit des ordonnancements qui sont déjà proches de la solution optimale.

La méthode hybride développée dans cet article a par ailleurs été adaptée à des problèmes d'ordonnancement avec coûts de communication inter-tâches [16], [24], [25]. Elle conduit alors à une amélioration de l'ordre de 10 à 20 % par rapport aux algorithmes d'ordonnancement par liste. Nous constatons que les méthodes s'avèrent robustes face aux algorithmes d'ordonnancement par liste lorsque les hypothèses simplificatrices sont moins fortes. Nous envisageons donc d'étendre notre étude à des problèmes d'ordonnancement plus complexes et plus proches de la réalité, que ce soit au niveau des architectures distribuées (réseaux de transputers...) ou au niveau des systèmes de production (problèmes de transport...).

L'application de la méthode hybride est toutefois limitée par la taille du modèle: elle n'est pas applicable si le nombre de tâches est trop élevé car le modèle markovien induit un trop grand nombre d'états. Récemment, différentes méthodes analytiques permettant de considérer les phénomènes de synchronisation ont été développées. Le remplacement de la méthode d'analyse markovienne par ces méthodes analytiques sera peut-être un moyen efficace pour pallier les problèmes de place mémoire et de temps d'exécution posés par la méthode hybride.

REMERCIEMENTS

L'auteur remercie tous les membres de l'équipe « Modélisation des Systèmes de Production et Aide à la Décision » (Laboratoire d'Informatique, Université Clermont-II) qui ont contribué à ce travail, et plus particulièrement Jeanne-Marie Gourgand, Michel Gourgand et Patrick Kellert.

RÉFÉRENCES

1. T. L. ADAM, K. M. CHANDY, J. R. DICKSON, A comparison of list schedules for parallel processing systems, *Communications of the ACM*, 1974, *Vol. 17*, p. 685-690.
2. J. BRUNO, Deterministic and stochastic scheduling with treelike precedence constraints, in *Deterministic and stochastic scheduling*, Dempster et al. (eds.), Dordrecht (Hollande), 1982, p. 367-374.
3. A. A. BULGAK, J. L. SANDERS, *Integrating a modified simulated annealing algorithm with the simulation of a manufacturing system to optimize buffer sizes in automatic assembly systems,* Proceeding of the 1988 Winter Simulation Conference, p. 684-690.
4. J. CARLIER, P. CHRÉTIENNE, *Problèmes d'ordonnancement*, Masson, Paris, 1988.
5. K. M. CHANDY, P. F. REYNOLDS, *Scheduling partially ordered tasks with probabilistic execution times*, in Proceedings of the Fifth Symposium on Operating Systems Principles, 1975, p. 164-177.
6. E. G. COFFMAN, R. L. GRAHAM, Optimal scheduling for two processor systems, *Acta Informatica*, 1972, *Vol. 1*, p. 200-213.
7. E. G. COFFMAN, *Computer and jobshop scheduling theory*, John Wiley and sons, 1976.
8. E. G. COFFMAN, Z. LIU, On the optimal stochastic scheduling of out-forests, *Operations Research*, Jan-Feb. 1992, *Vol. 40*, Supp. n° 1, p. 567-575.
9. C. COROYER, Z. LIU, *Effectiveness of heuristics and simulated annealing for the scheduling of concurrent tasks – an empirical comparison*, Rapport INRIA, n° 1379, Janvier 1991.
10. E. D'HOLLANDER, Computer aided dataflow analysis for the conversion of sequential programs into parallel form, *Special topics in supercomputing*, *Vol. 3*: algorithms and applications on vector and parallel computers), North-Holland, 1987.
11. A. FERRARA, R. MINCIARDI, *Resource constrained scheduling via simulated annealing: a discrete event approach*, Proceedings of the European Simulation Symposium, Ghent (Belgium), 1990, p. 177-181.
12. G. FLEURY, *Résolution de problèmes NP-complets: méthodes déterministes et stochastiques*, Thèse d'université, Université Blaise Pascal, Clermont-Ferrand II, 1993.
13. E. FROSTIG, A stochastic scheduling problem with intree precedence constraints, *Operations Research*, 1988, *Vol. 36*, p. 937-942.
14. M. R. GAREY, D. S. JOHNSON, *Computers and intractability: a guide to the theory of NP-completeness*, Freeman, New York, 1983.
15. J. M. GOURGAND, S. NORRE, Design and realization of a Petri Net simulator for program parallelization, Proceedings of the tenth IASTED International Conference, Innsbruck (Austria), p. 83-86, February 1992.
16. J. M. GOURGAND, S. NORRE, *Static allocation of tasks on multibus multiprocessor architectures*, OPOPAC (International Workshop On Principles Of PArallel Computing), 22-26 Novembre 1993, Lacanau (France), Ed. Hermès.
17. B. HAJECK, Cooling schedules for optimal annealing, *Mathematics of Operations Research*, 1988, p. 311-329.
18. T. C. HU, Parallel sequencing and assembly line problem, *Operational Research*, 1961, *Vol. 9*, p. 841-843.
19. H. KASAHARA, S. NARITA, Practical multiprocessor scheduling algorithms for efficient parallel processing, *IEEE Transaction on Computers*, 1984, *Vol. C33*, n° 11, p. 1023-1029.
20. T. G. LEWIS, W. H. PAYNE, Generalized feedback shift register pseudo random number algorithm, *J. ACM*, 1973, *Vol. 20*, n° 3, p. 456-468.

21. Z. Liu, E. Sanlaville, *Stochastically minimizing the makespan of structured jobs*, École d'été sur la théorie de l'ordonnancement et ses applications, 28 Sept-2 Oct 1992, Chateau de Bonas (Gers), France, p. 166-184.
22. M. A. Marsan, G. Balbo, G. Conte, *Performance models of multiprocessor systems*, The MIT Press, USA, 1986.
23. M. K. Molloy, Performance analysis using stochastic Petri nets, *IEEE Transactions on Computers*, 1984, *Vol. C31*, p. 913-917.
24. S. Norre, Static allocation of tasks on multiprocessor architectures with interprocessor communication delays, *Lecture Notes in Computer Science 694*, Arndt Bode-Mike Reeve-Gottfried Wolf (eds), Springer-Verlag, p. 488-499.
25. S. Norre, *Affectation de tâches sur une architecture multiprocesseur – Méthodes stochastiques et évaluation des performances,* Thèse de Doctorat, Université de Clermont-Ferrand II, 1993.
26. C. H. Papadimitriou, M. Yannakakis, Scheduling interval-ordered tasks, *Siam. J. Comput.*, 1979, Vol. 8, n° 3, p. 405-409.
27. M. Pinedo, G. Weiss, Scheduling jobs with exponentially distributed processing times and intree preceding constraints on two parallel machines, *Operations Research*, 1985, *Vol. 33*, p. 1381-1388.
28. QNAP2 version 8, manuel de référence, Société Simulog, 1991.
29. P. Quinton, *From specifications to machine code: parallelization methods*, Parallel and Distributed Algorithms, M. Cosnard et al. (Editors), North-Holland, 1989, p. 253-256.
30. P. Siarry, G. Dreyfus, *La méthode de recuit simulé : théorie et applications*, ISDET, Paris, 1988.
31. N. Tawbi, *Parallélisation automatique: estimation des durées d'exécution et allocation statique des processeurs*, Thèse de doctorat, Université Paris VI, MASI 91.47, 1991.
32. P. J. M. Van Laarhoven, *Simulated annealing: theory and applications,* Kluwer Academic Publishors, The Netherlands, 1989.
33. M. Widmer, *Modèles mathématiques pour une gestion efficace des ateliers flexibles*, Thèse d'état, École Polytechnique de Lausanne, 1990.