Analyse de la mise en œuvre parallèle d'applications de traitement du signal

Laurent KWIATKOWSKI, Fernand BOÉRI et Jean-Paul STROMBONI

Laboratoire UNSA d'Informatique Signaux Systèmes (I3S) - URA 1376 du C.N.R.S.
41, Boulevard Napoléon III - 06041 Nice Cédex - Tél : 93.21.79.61 - Fax : 93.21.20.54

RÉSUMÉ

Le développement et l'utilisation d'environnements d'aide à l'implantation d'applications de traitement du signal sur des architectures multiprocesseurs s'avère nécessaire pour la recherche de performances optimales. Ces outils ne tiennent pas compte de deux facteurs influant sur les performances : la granularité des tâches composant l'application et le rapport des durées communication/calculs de l'architecture cible (CCR). Les heuristiques utilisées ne peuvent fournir une solution optimale à un problème dont la formulation inhibe les parallélismes potentiellement exploitables. Nous présentons une chaîne d'analyse de la parallélisation des applications, en soulevant les divers problèmes rencontrés et en y apportant des éléments de solutions. Les résultats sont argumentés par la mise en œuvre d'une application de reconnaissance de formes : l'apprentissage par rétropropagation du gradient de l'erreur.

ABSTRACT

Design and use of environment shells for implementation of signal processing applications on multiprocessors is necessary when the best performance is required. These usual tools don't take into account the two parameters : the grain used for the application and the relative cost of communication and computing (CCR) on the target multiprocessor.

Available heuristics cannot reach the best solution if the formulation inhibits some of the application potential parallelisms. Here, an analysis process is presented for the parallelization of applications and the solutions for several related problems are discussed. The obtained results are illustrated through an example drawn from pattern recognition : the backpropagation learning algorithm.

1. INTRODUCTION

Le nombre important des calculs identiques effectués par certaines applications de traitement du signal suggère l'utilisation de machines multiprocesseurs pour accélérer les traitements. Cependant, l'obtention de performances significatives n'est pas triviale car d'une part la formulation des applications doit faire apparaître les parallélismes et d'autre part, ces parallélismes doivent être convenablement exploités sur l'architecture cible. La mise en œuvre peut donc être complexe car elle doit prendre en compte à la fois les parallélismes potentiels de l'application, les parallélismes effectifs de la machine et la stratégie d'allocation. C'est pourquoi plusieurs environnements d'aide à l'implantation ont été développés tels que SynDEx à l'INRIA Rocquencourt [1], Ptolemey à Berkeley [2] et TOPModèle [3] à l'Université de Nice - Sophia Antipolis. Ils ont pour but entre autres, de partitionner l'application, chacune des parties étant exécutée par un processeur différent de l'architecture cible. On espère ainsi exploiter les parallélismes potentiels inscrits dans la nature de l'application et diminuer le temps d'exécution par rapport à un traitement séquentiel. Une modélisation de l'application et de la machine sous forme de graphes est généralement entreprise et des méthodes d'allocation automatiques sont implémentées.

Dans cet article, nous présentons les différentes étapes de la chaîne d'analyse et de la mise en œuvre d'une application sur multiprocesseur en soulevant les problèmes liés au choix du grain de modélisation de l'application et de la méthode d'allocation. La première partie traite du problème de la modélisation, plus particulièrement du choix de la granularité des tâches composant l'application et de l'impact sur les performances parallèles. Une analyse des parallélismes potentiels est nécessaire afin de déterminer des bornes de performances et de suggérer des caractéristiques architecturales. La seconde expose différentes méthodes d'allocation pour diminuer le temps d'exécution de l'application et suggère diverses stratégies en fonction du CCR (Communication Computation Ratio). Enfin, la dernière partie met en œuvre une application de reconnaissance de formes par apprentissage avec la rétropropagation du gradient de l'erreur.

2. MODELISATION ET ANALYSE

2.1. Modélisation de l'application

La première étape de la chaîne concerne la modélisation de l'application. On adopte un graphe logiciel où les sommets représentent les tâches et les arcs, les dépendances entre tâches. Cette représentation dite dataflow est intéressante car elle fait apparaître les parallélismes potentiels des calculs et des échanges. Cependant, différents grains de décomposition sont possibles pour une même application; la granularité de décomposition pouvant être définie comme le rapport entre la durée moyenne d'une tâche et la durée totale de l'application. Par exemple dans [4], les tâches sont divisées en sous-tâches de durée élémentaire afin d'obtenir un ensemble de tâches de durée identique. La granularité la plus fine permet en effet de faire apparaître l'ensemble des parallélismes potentiels, mais elle pose un problème d'explosion combinatoire lorsque la taille des applications est importante.

Les environnements d'aide cités considèrent une décomposition a priori, où la granularité dépend de la taille de l'application modélisée. L'explosion combinatoire est ainsi évitée, mais les parallélismes exploitables sont réduits. Considérons l'exemple de la figure n°1 du calcul d'un produit C d'une matrice carrée $n \times n$ W par un vecteur E de taille n.

L'exécution de l'application séquentielle nécessite n^2 multiplications et additions (la tâche correspond alors au calcul de toutes les composantes du vecteur C, ce qui peut être apparenté au grain le plus gros). Si chacune des tâches est le calcul d'une composante de ce vecteur, la granularité de décomposition est moyenne. Enfin, la granularité est fine lorsque les tâches représentent les opérations élémentaires de multiplication et d'addition.

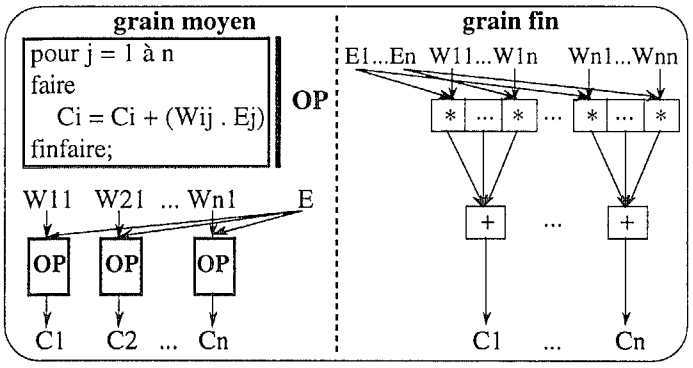

Figure n°1 : Modélisation d'une application selon la granularité

Il serait intéressant de savoir choisir la granularité. Afin de faire apparaître l'ensemble des parallélismes potentiels exploitables et donc d'espérer obtenir des performances maximales, une modélisation grain fin s'impose. Il est parfois possible que l'implémentation d'une application modélisée par un graphe logiciel grain fin conduise à regrouper les tâches de telle manière que l'on aurait pu utiliser un grain moyen. Dans ce cas particulier, le décomposition grain fin n'est pas inutile, car elle permet de prouver que l'emploi de la décomposition grain moyen ne grève pas les performances et que les parallélismes potentiels sont effectivement bien exploités. Afin de repousser le problème de l'explosion combinatoire, une méthode adaptée aux applications régulières et fondée sur les variations de la granularité ainsi que sur la taille de l'application est développée dans [5]. Elle a été mise en œuvre dans la dernière partie.

2.2. Analyse des parallélismes potentiels

La modélisation de l'application sous la forme d'un graphe grain fin permet d'analyser de façon détaillée, les parallélismes afin de déterminer une borne d'accélération et de dimensionner l'architecture cible si celle-ci n'est pas préalablement définie. La borne d'accélération est obtenue en considérant une durée de communication interprocesseur nulle. Il est ainsi possible de vérifier que les parallélismes potentiels sont effectivement exploités, notamment lorsque des heuristiques d'allocation sont utilisées, pour comparer les résultats obtenus avec une borne, à défaut de l'optimum.

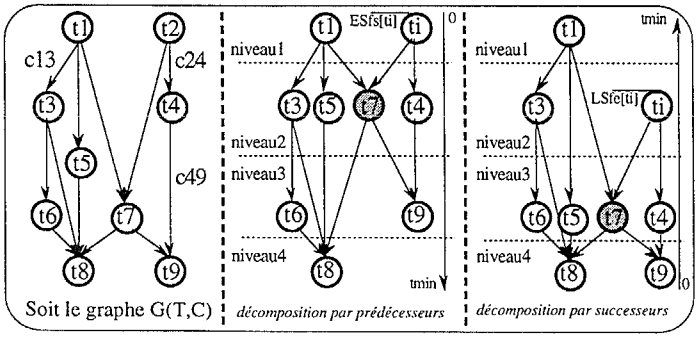

Figure n°2 : Décomposition en niveaux du graphe logiciel

Considérons le graphe logiciel G(T,C) de la figure n°2, où T est l'ensemble des tâches et C l'ensemble des contraintes. G est décomposé en niveaux ou phases de précédence de manière à faire apparaître le chemin critique et le degré de simultanéité des calculs et des transferts. Il existe d'autant plus de décompositions que les tâches possèdent une marge (durée dont peut être retardé le départ de la tâche sans augmenter le temps d'exécution global). Deux décompositions en niveaux importantes sont celle par prédécesseurs (décomposition au plus tôt) et celle par successeurs (décomposition au plus tard). Pour la première, le premier niveau est constitué de toutes les tâches n'ayant aucun prédécesseur. Quelque soit le niveau k≠1, celui-ci est constitué des tâches dont tous les prédécesseurs sont dans des niveaux inférieurs et ayant au moins un prédécesseur dans le niveau (k-1). La seconde est la duale de la précédente dans la mesure où dans un niveau k, toutes les tâches possèdent leurs successeurs dans des niveaux supérieurs dont un au moins dans le niveau immédiatement supérieur. Le temps d'exécution d'un chemin du graphe étant la somme des temps d'exécution des tâches qui le composent, le temps d'exécution t_{min} du chemin critique est le plus grand chemin du graphe.

Dans le cas particulier [4] des tâches de durée identique, le plus long chemin est celui possédant le plus grand nombre de tâches. t_{min} est donc le produit de la durée d'une tâche par le nombre de niveaux. Lorsque les tâches sont de durée quelconque, t_{min} est obtenu par :

$$t_{min} = \mathrm{Max}_{\,ti \in \,graphe}\ (ESfs[ti] + di) \quad \text{ou } t_{min} = \mathrm{Max}_{\,ti \in \,graphe}\ (LSfe[ti]) \qquad (1)$$

avec ti la tâche i du graphe, di la durée de cette tâche, ESfs[ti] la date d'exécution au plus tôt par rapport au début du graphe G et LSfe[ti] la date d'exécution au plus tard par rapport à la date de fin du graphe G.

Chaque niveau du graphe logiciel contient un certain nombre de tâches indépendantes constituant sa largeur. Une étude de celle-ci permet de déterminer le nombre maximum n_{max} de processeurs nécessaires à l'exécution de l'application en t_{min}, quelque soit la décomposition en niveaux. Ce nombre doit être suffisant pour exécuter toutes les tâches qui peuvent l'être en même temps. Pour ce faire, nous définissons une fonction d'activité des tâches, notée f(ti) tenant compte de leur marge :

$$f(ti) = \left| \begin{array}{l} 1 \text{ pour } t \in [ESfs[ti],\ LSfs[ti] + di] \\ 0 \text{ ailleurs} \end{array} \right| \qquad (2)$$

La date ESfs[ti] considère une marge nulle et correspond à la date de fin d'exécution de l'ensemble des prédécesseurs de ti. LSfs[ti] fait intervenir la marge : LSfs[ti] = ESfs[ti] + marge[ti]. Le fait qu'une tâche possède une marge implique que nous la considérions active pendant plus de temps qu'elle ne l'est en réalité. Nous considérons ainsi tous les cas d'appartenance aux différents niveaux et définissons la borne maximale du nombre de processeurs nécessaires pour l'architecture cible. Le nombre de tâches actives au temps t est indiqué par la fonction F(t), sa valeur maximale pour t compris entre 0 et tmin définie n_{max}.

$$F(t) = \sum_{ti \in G} f(ti) \quad \text{et} \quad n_{max} = \mathrm{Max}_{\,t \in [0,\ tmin]}\ [F(t)] \qquad (3)$$

Il faut au plus n_{max} processeurs pour que l'application soit exécutée en un temps minimum. Prendre un nombre de processeurs supérieur engendre alors une baisse de l'efficacité de la machine puisqu'ils n'apportent aucune amélioration. Il existe éventuellement un nombre de processeurs $n_{opt}<n_{max}$ assurant une meilleure efficacité, mais il correspond à une décomposition en niveaux particulière, difficilement identifiable sauf dans des cas simplistes, car elle peut nécessiter une recherche exhaustive de toutes les décompositions. Prendre une taille inférieure à n_{opt} entraîne une augmentation du temps d'exécution. L'application sera donc implantée sur un nombre de processeurs compris entre n_{opt} et n_{max} et son temps d'exécution sera supérieur à t_{min}.

2.3. Modélisation de l'architecture multiprocesseur

La recherche de la décomposition en niveaux de l'application fournissant le temps d'exécution minimal, sur un nombre de processeurs minimum, peut être effectuée par l'intermédiaire de méthodes d'allocation automatiques minimisant ces critères. Il faut alors modéliser l'architecture cible. On utilise un graphe comme pour l'application; graphe matériel où les sommets sont les processeurs et les arcs, les liens physiques de communication entre processeurs. Cette modélisation n'est pas évidente car la diversité des caractéristiques architecturales rend difficile la recherche d'un modèle universel. C'est pourquoi les environnements [1] et [2] se tournent vers la modélisation de machines particulières, constituées de processeurs spécialisés Transputers ou DSPs, limitant les architectures considérées.

Afin d'essayer de faire apparaître un plus grand nombre de possibilités, une modélisation des architectures a été entreprise dans l'environnement d'aide TOPModèle [6]. Un sommet du graphe matériel peut contenir une ou plusieurs ressources telles que l'unité de traitement, de contrôle, de mémorisation et de routage. La machine peut donc être homogène ou hétérogène, la mémoire partagée ou distribuée. Un partitionnement des sites de calcul est introduit afin de représenter les machines partitionnables (MultiSIMD [7]) et les modes parallèles mixtes (SIMD/SPMD [8]). Les unités de calcul sont classées selon un ensemble de registres d'instruction (RI), chaque unité d'une classe est alors soit inactive, soit contrainte à l'exécution de l'opération contenue dans le RI. Afin de tenir compte des modèles de communication associés à chaque topologie, les arcs sont regroupés d'abord en *voies*, ensembles de liens capables de travailler simultanément puis en *canaux*, ensembles de voies actives successivement au cours de la communication. Il est ainsi possible de spécifier qu'un port d'entrée/sortie unique soit partagé dans le temps entre plusieurs liens entrants et sortants d'un processeur. Un délai de transmission est associé à chaque voie pour chaque type de donnée, ainsi qu'un délai de commutation de voie à chaque canal de communication. La figure n°3 décrit un exemple de modélisation de machine existante [9].

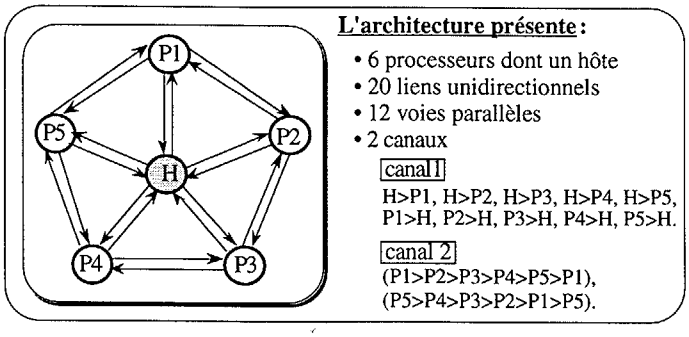

Figure n°3 : Modélisation de l'architecture NNE-CA avec TOPModèle

Dans cet exemple, le canal 1 est constitué de 10 voies pouvant être actives successivement, ce qui signifie que le processeur hôte H possède un unique port d'entrée/sortie alors que les autres processeurs en possèdent trois. Le canal 2 décrit une topologie de type anneau. Les deux canaux peuvent être actifs simultanément : les processeurs reliés en anneau échangent les informations suivant deux sens possibles, l'un d'eux peut en même temps envoyer des informations au processeur hôte ou en recevoir de ce dernier.

Les machines existantes proposent différents modes de calcul parallèle qui diffèrent par les contraintes de synchronisation, la condition de démarrage du calcul et la charge acceptée. TOPModèle simule trois modes, à savoir le SIMD, SPMD et le MIMD. Les données ne sont pas traitées en réalité; l'occupation des sites est représentée à l'aide d'indicateurs d'activité. De la même manière, un indicateur d'activité est associé à chaque ressource de communication.

Après avoir effectué la modélisation, l'étape suivante consiste à rechercher l'allocation des tâches composant l'application sur les processeurs de l'architecture cible, de manière un minimiser un critère qui peut être le temps d'exécution. C'est un problème NP-complet qui fait appel à des méthodes d'allocation automatiques.

3. CHOIX DE LA STRATEGIE D'ALLOCATION

Nous considérons dans cette partie uniquement les méthodes d'allocation statiques, qui nécessitent la connaissance a priori des caractéristiques des tâches composant l'application. Dans le cas d'un graphe quelconque, le seul fait que les tâches soient de durée différente rend le problème NP-complet. C'est pourquoi se sont développées des heuristiques fournissant des solutions sous-optimales en un temps polynomial. En particulier, les méthodes sérielles ou heuristiques de liste [10] sont assez efficaces en ce sens que les expérimentations ont montré qu'elles permettent d'obtenir un résultat proche de celui du recuit simulé [11], tout en étant plus rapide [12].

Le principe de ces heuristiques consiste à choisir une tâche parmi un ensemble de candidates selon un ordre de priorité qui peut être statique (la liste est établie de manière définitive) ou dynamique (la liste de priorité est remise à jour lors de chaque affectation). Il existe de multiples manières de construire ces listes et il y a un certain arbitraire dans le choix de l'ordre des tâches, en particulier pour les tâches d'un même niveau de priorité. Le déroulement de l'heuristique s'effectue suivant une stratégie gloutonne, c'est-à-dire qu'un choix établi n'est jamais remis en question. [1] utilise par exemple un critère basé sur la marge des tâches ou Schedule Flexibility (SF), [6] utilise le ESfs ou le LSfe (latest start from end).

Les environnements d'aide comportent généralement des heuristiques parmi une unique stratégie d'allocation : le placement et l'ordonnancement comme dans [1] et [2]. L'optimisation du temps d'exécution des applications est alors obtenue en jouant sur la fonction critère utilisée et le grain de modélisation. Cependant, il peut être plus judicieux d'utiliser d'autres stratégies, à savoir le regroupement ou la duplication de tâches. Une étude dans [5] sur l'application considérée dans ce papier, montre que le CCR peut être un critère de choix de la stratégie d'allocation. Cette étude, ainsi que la comparaison entre les différentes heuristiques citées précédemment sont développées dans la partie suivante.

4. APPLICATION : APPRENTISSAGE PAR RPGE

4.1. Description de l'application

Cette application est associée à l'apprentissage du réseau neuromimétique multicouche [13]. Les coefficients synaptiques de plusieurs matrices mémorisent les formes apprises. A partir d'un exemple d'entrée, le parcours du réseau consiste à effectuer plusieurs produits matrice-vecteur pour obtenir une sortie correspondante (classification) ou égale à l'entrée. L'apprentissage consiste à comparer la sortie obtenue à celle désirée afin d'en déduire une erreur puis calculer un terme correcteur pour modifier les coefficients. Cette opération est itérée jusqu'à ce que l'erreur devienne inférieure à un certain seuil, signe que l'exemple est mémorisé. L'apprentissage d'autres formes s'effectue en changeant l'exemple d'entrée et la sortie désirée. Le réseau reconnaîtra la forme apprise se rapprochant le plus de l'entrée présentée.

4.2. Modélisation et analyse

Afin de montrer que la granularité de modélisation peut influer sur les performances, l'application est modélisée selon un grain moyen correspondant au produit d'une colonne de matrice par un vecteur d'entrée (traitement d'une cellule en connexionnisme) puis selon un grain fin du niveau opérateur élémentaire.

L'application est de taille réduite pour éviter l'explosion combinatoire du nombre de tâches et de dépendances. L'exemple d'entrée est de taille 4, celui de sortie 2. Deux matrices sont utilisées W et Z de dimension respective (4x2) et (2x2). 14 sommets et 24 arcs sont nécessaires pour le graphe grain moyen alors qu'une modélisation grain fin demande 86 sommets et 140 arcs. La représentation est alors difficile mais fait bien apparaître les parallélismes potentiels comme le montre la figure n°4.

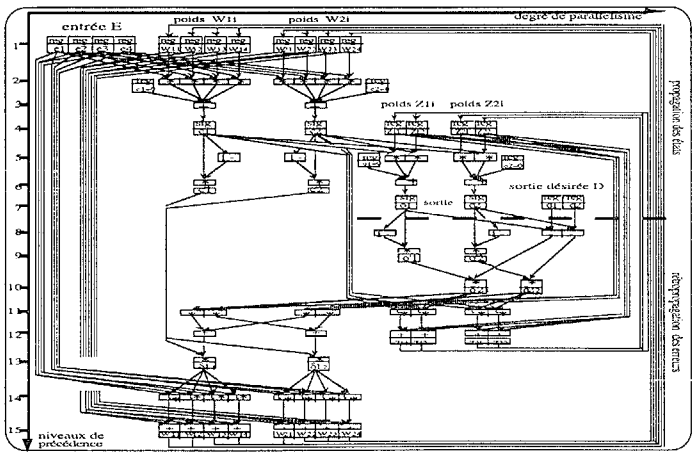

Figure n°4 : Modélisation par un graphe à granularité fine

L'architecture cible est un multiprocesseur MIMD de 4 processeurs (temps d'exécution du DSP96002) totalement interconnectés capable d'effectuer des recouvrements temporels entre calculs et échanges de données. L'analyse du graphe selon les formules (1) et (3) permet de déterminer la borne d'accélération : 4,39 et la taille machine : $n_{max} = 22$. Mais il est facile de voir qu'il existe un $n_{opt} = 12$ assurant 4,39 d'accélération maximale.

4.3. Utilisation des stratégies d'allocation

Dans un premier temps, le nombre de processeurs n'est pas limité et une technique de recuit est utilisé afin de rechercher l'allocation optimale minimisant le temps d'exécution (les durées des communications sont prises en compte). L'accélération prédite est de 2,3 pour une modélisation grain fin contre 1,8 pour le grain moyen. Les performances parallèles varient donc de 25% suivant la granularité choisie lors de la modélisation. Afin de repousser le problème de l'explosion combinatoire, une technique adaptée aux applications régulières est mise en œuvre dans [5]. Elle consiste à réduire la taille de l'application afin de rendre possible une décomposition grain fin et d'effectuer toute l'étude de l'implémentation sur le graphe associé. Ensuite, elle cherche à augmenter le grain sans altérer les performances obtenues, en regroupant les tâches. La nouvelle représentation est extrapolée pour modéliser l'application de taille initiale.

L'utilisation des heuristiques de liste SF ou LSfe permet d'obtenir un comportement qualitatif proche du recuit simulé avec moins de 4% d'erreur. D'autre part, la recherche d'un compromis accélération/efficacité motive le choix de 4 DSP assurant 50% d'utilisation pour une accélération de 1,9.

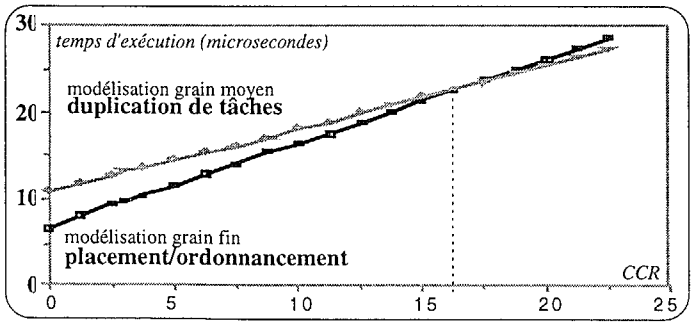

Figure n°5 : Influence du CCR sur la stratégie d'allocation

Les performances des heuristiques de placement et d'ordonnancement utilisées sont comparées avec celles obtenues en employant une stratégie de duplication de tâches grain moyen [14] (Cf figure n°5). En fonction du CCR et en particulier lorsqu'il est supérieur à 16 pour l'application considérée, il est préférable de garder une granularité élevée et d'employer une technique de duplication.

5. CONCLUSION

Les résultats présentés pour une application particulière du traitement du signal montrent que la parallélisation efficace n'est pas triviale. Les performances dépendent de la granularité adoptée pour la décomposition de l'application lors de sa modélisation. Une étude sur le CCR soulève le problème du choix de la stratégie d'allocation et de la granularité. Les environnements de simulation doivent prendre en compte ces difficultés pour aboutir aux performances optimales. La dernière étape de la chaîne d'analyse de la parallélisation consisterait à générer le programme de manière à implanter l'application sur un multiprocesseur réel. C'est l'un des développements nécessaires à cette étude.

REFERENCES

[1] Y. Sorel, C. Lavarenne et al. : *"Implantation d'algorithmes de traitement d'images sur une architecture multiDSP avec l'environnement d'aide SynDEx"*, 14ème colloque GRETSI, Juans-les-Pins, sept 1993, pp 1019-1022.

[2] Ptolemy 0.5 user's manuel, University of California at Berkeley College of Engineering Department of Electrical and Computer Sciences Berkeley California - 94.720.

[3] JP. Stromboni, L. Kwiatkowski : *"MCPS Prototyping using Deterministic Simulation"*, Conference on Massively Parallel Computing Systems : MPCS, Ischia, ITALY, Edition IEEE Computer Society Press, 02 - 06 mai 1994.

[4] M. Cosnard, D. Trystram : *"Algorithmes et architectures parallèles"*, Collection Informatique Intelligence Artificielle, Edition Interéditions, 1993.

[5] L. Kwiatkowski : *"Placement et ordonnancement pour l'exploitation des parallélismes d'un algorithme. Application à un algorithme connexionniste"*, Thèse soutenue à l'UNSA, spécialité SPI, janvier 1995.

[6] JP. Stromboni, L. Kwiatkowski : *"Conception d'un modèle pour l'analyse du parallélisme"*, International workshop on principles of parallel computing, OPOPAC, Lacanau, Edition Hermès, nov 1993, pp 85-99.

[7] MA. Nichols et al. : *"Eliminating memory fragmentation within partitionable SIMD/SPMD machines"*, IEEE Trans. on parallel and distributed systems, Vol 2, N°3, 1991, pp 290-303.

[8] M. Auguin, F. Boéri, JP. Dalban : *"Synthèse et évaluation du projet OPSILA"*, Technique et Sciences Informatiques, Vol 9, N°2, 1990, pp 79-98.

[9] K. Park, K. Cha, J. Choi : *"Neural Network Emulator : the implementation of neural network emulator board"*, Visual Communications and Image Processing, SPIE, Vol 1360, 1990.

[10] B. Braschi : *"Principes de base des heuristiques d'ordonnancement de liste affectation de priorités aux tâches"*, Thèse INPG, Spéc. Informatique, nov 1990.

[11] S. Kirkpatrick, CD. Gelatt, MP. Vecchi : *"Optimization by simulated annealing"*, Science, Vol 228, 3° Edition, 1989.

[12] Z. Liu, C. Coroyer : *"Effectiveness of heuristics and simulated annealing for the schedulingof concurrent tasks - An empirical comparison"*, PARLE'93, Proceedings of the 5th international Parallel Architectures and Langages Europe Conference, Munich, Germany, juin 1993, pp 452-463.

[13] B. Widrow, MA. Lehr : *"30 years of adaptative neural networks : Perceptron, madaline and Backpropagation"*, Proceedings of the IEEE, Vol 78, N°9, sept 1990, pp 1415-1441.

[14] H. Yoon, JH. Nang, SR. Maeng : *"Parallel simulation of multilayered neural networks on distributed-memory multiprocessors"*, Microprocessing and microprogramming, Vol 29, 1990, pp185-195.