MATISSES : une Machine à Transputer Incluant la Synchronisation et l'Émission Scalaire

Ronan KERYELL
Centre de Recherche en Informatique
École des Mines de Paris*

1 Introduction

Le modèle de programmation à parallélisme de données (*dataparallèle*) a reçu une très large attention en ce qui concerne la programmation des machines massivement parallèles, aussi bien SIMD que MIMD, car il offre un compromis intéressant entre la facilité de programmation, l'efficacité et la portabilité à de nombreuses applications scientifiques numériquement intensives.

Dans ce modèle, le programmeur exprime les opérations à effectuer simultanément sur des ensembles de données, telles qu'additionner 2 vecteurs ou 2 matrices. Un tel modèle est plus familier au numéricien habitué à manipuler de telles entités mathématiques que, par exemple, un modèle à parallélisme de contrôle où il faut découper le problème en tâches concurrentes. De plus, une telle formulation des problèmes permet de retrouver à l'exécution le parallélisme de données inhérent à de nombreux modèles physiques sans devoir retrouver tout le parallélisme implicite à la compilation.

Ce modèle a été développé dès les premières machines parallèles SIMD pour refléter leur fonctionnement [9] : une unité de contrôle envoie une instruction par cycle qui est exécutée sur tous les processeurs élémentaires (PEs) sur différentes données du problème. Le modèle a été de même intensivement utilisé sur les machines vectorielles.

À l'opposé, les machines MIMD sont capables d'exécuter différentes instructions sur chaque PE qui opère sur différentes données. En pratique il est difficile d'écrire un programme différent par PE et c'est souvent le même programme répliqué sur tous les PEs qui est exécuté. L'avantage est qu'on est libéré du synchronisme du SIMD, mais au coût de la partie contrôle en plus.

En ce sens les machines MIMD sont intéressantes même avec un modèle de programmation à parallélisme de données car elles sauront mieux tirer parti de la liberté d'exécution dans le cas de contrôle de flot parallèle tout en gardant la facilité de programmation liée à l'existence d'un état global quasi synchrone du système. La compilation d'un tel modèle, tel que celui de HPF [3], sur une machine MIMD nécessite le respect des dépendances entre données, tout en minimisant le synchronisme SIMD. Par contre, des synchronisations entre PEs peuvent être nécessaires, surtout dans le cas de code irrégulier, ce qui peut limiter l'efficacité de la machine. La diffusion de valeurs à tous les PEs est nécessaire si on veut éviter d'avoir à calculer des expressions complexes identiques sur tous les PEs, par exemple dans le cas de code scalaire et d'entrées-sorties. Enfin, comme l'efficacité du parallélisme diminue en général avec le nombre de processeurs, on a intérêt à pouvoir subdiviser la machine en plusieurs sous-machines si on doit résoudre plusieurs problèmes différents simultanément, dans une approche « centre de calcul » par exemple. C'est ce qu'on suppose par la suite.

*77305 FONTAINEBLEAU Cedex, FRANCE, Tél : (+33 1) 64.69.48.44, Fax : (+33 1) 64.69.47.09, E-mail : keryell@cri.ensmp.fr.

Nous présentons des approches plus spécialisées pour le TRANSPUTER T9000 qui est un matériau intéressant dans la mesure où il permet de faire des machines MIMD compactes simplement [6]. La section 2 décrit la méthode de partitionnement utilisée dans la machine cible. Cette méthode est reprise dans la section 3 pour définir un nouveau type de réseau de synchronisation et ensuite dans la section 4 un réseau de diffusion partitionnable spécifique au TRANSPUTER est décrit.

2 Partitionnabilité

Dans l'approche proposée par AMDAHL [1], si un programme contient des proportions s et p scalaire et parallèle respectivement, l'accélération d'un programme sera de l'ordre de $a = \frac{1}{s+p/N}$ dans le cas d'une machine à N PEs. L'efficacité de chaque PE est donc de $\frac{1}{p+Ns}$ et on a donc tout intérêt à avoir un nombre minimal de PEs pour résoudre un problème. En prenant une autre mesure sur des opérations préfixes parallèles, l'efficacité tombe là aussi à $\mathcal{O}(\frac{1}{\log N})$.

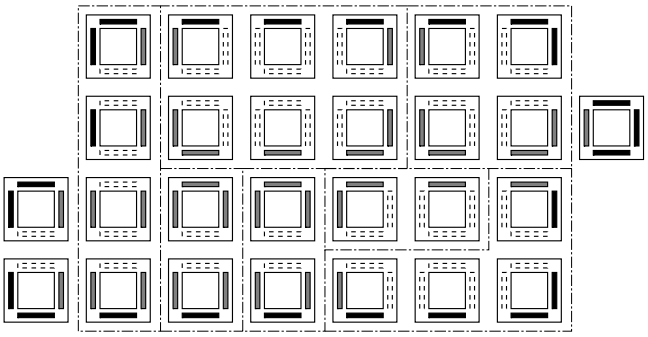

Figure 1: Exemple de partitionnement.

Si on a de nombreuses applications à faire tourner, on aura tout intérêt à utiliser plusieurs machines plus petites qu'une grosse machine en terme d'efficacité[1], d'où l'intérêt d'offrir des machines partitionnables. Le multitâche reste néanmoins nécessaire afin de cacher des entrées-sorties par des changements de tâches, mais seul un petit nombre de tâches est nécessaire.

Comme l'organisation logique des PEs dans le modèle HPF est de type grille de processeurs, il est intéressant de développer un système de partitionnement en grille. Ce système est présenté sur la figure 1 dans le cas d'une machine de type grille 2D.

Chaque limite de PE contient une barrière à trois états possibles, codés sur 2 bits pour indiquer [7] une limite physique infranchissable (telle que le bord physique de la machine, en noir sur le schéma), une limite logique infranchissable par des messages de type « utilisateur » (en grisé sur la figure, qui définit donc les partitions) et bien entendu le cas où il n'y a pas de frontière (on est donc entre 2 PEs d'une même partition).

3 Barrière de synchronisation floue partitionnable

Le principe de base est de bloquer tous les PEs sur une condition globale qu'il faut calculer rapidement, par exemple un *ou* global.

Dans le cas de la machine ARMEN [2] une évaluation séquentielle récursive des conditions globales partielles C_i à partir des conditions locales c_i

$$C_{i+1} = c_{i+1} \vee C_i$$

est utilisée et le résultat est redistribué à tous les PEs. Si on ne considère que les temps de propagation τ entre PEs, le temps de synchronisation est $\hat{t} = \max_{\substack{1\le i\le N\\ 1\le j\le N}} t_{i,j} = \max_{\substack{1\le i\le N\\ 1\le j\le N}} (2N-i-j)\tau = 2(N-1)\tau$ et il faut 2 fils entre chaque PE.

[1] À condition bien entendu que la mémoire des sous-machines le permette.

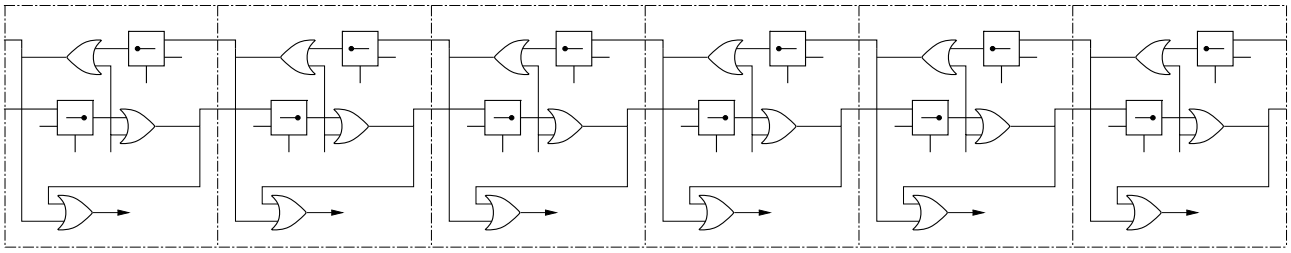

Figure 2: *Ou* global partitionnable optimal en dimension 1.

L'optimisation consiste à doubler le système d'une évaluation dans l'autre sens et de faire gagner l'évaluation la plus rapide pour obtenir [7] :

$$\hat{t'} = \max_{\substack{1 \le i \le N \\ 1 \le j \le N}} (\min(t_{i,j}, t_{N+1-i,N+1-j}))\tau$$

en calculant

$$C_N = (\bigvee_{i=1}^{N} c_i) \wedge (\bigvee_{i=1}^{N} c_{N+1-i})$$

A priori cela impose le doublement du nombre de fils mais en fait on peut économiser les signaux de retour, ce qui revient à calculer sur le PE p

$$C_N = (\bigvee_{i=1}^{p} c_i) \vee (\bigvee_{i=p}^{N} c_i)$$

en un temps

$$\hat{t} = \max_{1 \le i \le j \le n} (j - i)\tau = (n-1)\tau$$

qui est optimal compte tenu du temps de propagation de l'information, du nombre de fils et de la bissection : 2 fils entre chaque PE. Un exemple sur un réseau linéaire est donné sur la figure 2 mais est bien entendu généralisable à d'autres dimensions. Le partitionnement intervient via les bits locaux P_g et P_d. Dans la pratique ce réseau est presque aussi performant qu'un réseau préfixe parallèle symétrique et beaucoup moins cher [7] ce qui nous le fait préférer.

Ce réseau convient à une machine SIMD mais pas à une machine MIMD puisqu'il faut en plus savoir quand commence l'évaluation de la condition globale sous peine de tomber dans une étreinte mortelle. Il faut donc évaluer 2 conditions et par conséquent doubler ce réseau.

En ayant ces 2 réseaux on peut réaliser une barrière floue qui consiste à élargir par percolation des instructions du programme [4] la zone de barrière sans changer la sémantique du programme pour essayer de recouvrir phase de calcul et phase de synchronisation [5]. L'algorithme utilisant 2 réseaux *eb* et *sb* est présenté sur la figure 3 [7].

Initialisation :
  $\forall i \in [1, N],\ eb_i \leftarrow 0,\ sb_i \leftarrow 0$
...
Entrée en synchronisation :
  $eb_p \leftarrow 1$
...
Sortie de synchronisation :
  tant que $(\bigwedge_{i=1}^{N} eb_i \neq 1)$
    `attendre`
  $sb_p \leftarrow 1$
  tant que $(\bigwedge_{i=1}^{N} sb_i \neq 1)$
    `attendre`
  $eb_p \leftarrow 0$
  $sb_p \leftarrow 0$

Figure 3: Algorithme de synchronisation.

On constate que $(eb_i, sb_i) = (1, 1)$ est un élément neutre ; on peut donc par ce biais retirer des PEs de la barrière et faire de la planification statique de barrière [8], pouvant être facilitée par l'ajout de compteurs matériels [7].

Sur une machine à T9000, la synchronisation peut loger dans un PAL connecté à des signaux `EventIn`x et `EventOut`x.

4 Réseau de diffusion

Un T9000 est muni de 6 liens de communications dont seulement 4 sont utilisables pour transmettre des données ; les autres sont des liens de contrôle et d'initialisation [6]. Il est dommage qu'un tiers du réseau soit ainsi gâché.

Nous proposons une perversion de ces 2 liens d'une manière similaire à la barrière de synchronisation en utilisant un système de grilles partitionnables pour faire des diffusions éventuellement partielles de valeurs. Le principe est le suivant : chaque PE configure grâce à 2 sorties `EventOut`x le domaine de diffusion de sorte que le lien émetteur soit relié à tous les liens récepteurs suivant un graphe plaqué sur le réseau grille sous-jacent. Le PE émet une valeur par une instruction réseau `CPoke` [6]. Comme l'émetteur s'attend à recevoir une reconnaissance liée au protocole du réseau, on reboucle le $2^{\text{ème}}$ lien sur le 1^{er} au niveau du processeur émetteur afin qu'il reçoive cette reconnaissance.

Ce système est suffisamment simple là aussi pour loger dans un PAL. Bien entendu, en dehors des diffusions, les liens sont disponibles pour leur fonction originelle.

5 Conclusion

Ultimement, les communications, synchronisations et diffusions font partie de la partie séquentielle du code dans la « formule d'AMDAHL » [1] et sont donc à accélérer si on veut exploiter le maximum de parallélisme. Il faut diminuer ces facteurs dans les ordinateurs parallèles ; c'est ce qui est fait dans cet article.

Enfin, l'intérêt pragmatique des systèmes décrits ci-avant est qu'ils sont simples et que même si on ne parvient pas à les réaliser, la machine finale se comportera aussi bien qu'une autre machine à TRANSPUTER et n'est donc pas absolument dépendante de ces mécanismes.

Il est possible de faire des machines originales à base de TRANSPUTERS : ARMEN en est un exemple, MATISSES en est une autre. Sans une telle valeur ajoutée les machines à TRANSPUTERS sont toutes équivalentes et les perversions de T9000 ci-avant seraient très intéressantes pour faire sortir du rang certaines machines. Ces méthodes sont bien sûr aussi utilisables pour toute machine MIMD avec d'autres processeurs.

Bibliographie

[1] Gene M. AMDAHL. ¡¡ Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities ¿¿. In AFIPS, editor, *Proceedings of the Spring Joint Computer Conference*, pages 483–485, 1967.

[2] Jean-Marie FILLOQUE. ¡¡ *Synchronisation répartie sur une machine parallèle à couche logique reconfigurable* ¿¿. Thèse, Université de Rennes I, novembre 1992.

[3] High Performance Fortran FORUM. ¡¡ High Performance Fortran Language Specification ¿¿. DRAFT Version 1.0 CRPC-TR 92225, Center for Research on Parallel Computation, Rice University, Houston, USA, janvier 1993. Récupérable par `ftp anonymous` sur la machine `titan.cs.rice.edu` dans le fichier `public/HPFF/draft/hpf-v10.ps.Z`.

[4] Caxton C. FOSTER and Edward M. RISEMAN. ¡¡ Percolation of Code to Enhance Parallel Dispatching an Execution ¿¿. *IEEE Transactions on Computers*, C-21(12):1411–1415, décembre 1972.

[5] Rajiv GUPTA and Michael EPSTEIN. ¡¡ High Speed Synchronization of Processors Using Fuzzy Barriers ¿¿. *International Journal of Parallel Programming*, 19(1):53–72, février 1990.

[6] inmos — SGS-THOMSON. ¡¡ *The T9000 Transputer Products Overview Manual* ¿¿, first edition, 1991.

[7] Ronan KERYELL. ¡¡ *POMP : d'un Petit Ordinateur Massivement Parallèle SIMD à Base de Processeurs RISC — Concepts, Etude et Réalisation* ¿¿. Thèse, Laboratoire d'Informatique de l'Ecole Normale Supérieure — Université Paris XI, octobre 1992.

[8] Matthew T. O'KEEFE and Henry G. DIETZ. ¡¡ Hardware Barrier Synchronization: Static Barrier MIMD (SBM) ¿¿. In *Proceedings of the 1990 International Conference on Parallel Processing*, pages I–35–I–42. IEEE, août 1990.

[9] S. H. UNGER. ¡¡ A Computer Oriented Toward Spatial Problems ¿¿. In *Proceedings of the IRE*, pages 1744–1750, octobre 1958.