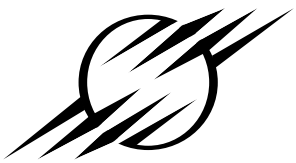
CENTRE DE RECHERCHE EN INFORMATIQUE

Modèle de Compilation d'HPF pour la Machine MIMD à Bancs Mémoire et Réseau Distribué Programmable Phénix

Pierre Fiorini
François Irigoin
Ronan Keryell

22 mai 1996



Document EMP–CRI A/297-CRI

8ème Rencontres Francophones du Parallélisme (RenPar8)
20-24 mai 1996, Bordeaux, FRANCE.

RenPar'8,
Bordeaux, France, 20–24 mai 1996.

Modèle de Compilation d'HPF pour la Machine MIMD à Bancs Mémoire et Réseau Distribué Programmable Phénix

Pierre FIORINI[a] François IRIGOIN[b] Ronan KERYELL[b]

[a]DSIT/DAPSA/PTO, Cité de l'Air, 2bis, rue Lucien Boussoutrot, 00460 ARMÉE.

[b]CRI-ENSMP, 35, rue Saint-Honoré, 77305 FONTAINEBLEAU CEDEX,
{irigoin,keryell}@cri.ensmp.fr, http://www.cri.ensmp.fr/pips.

Résumé

Nous décrivons une approche basée sur l'algèbre linéaire de compilation du langage parallèle HPF pour le projet PHÉNIX de co-conception matérielle et logicielle. Cette machine MIMD a la particularité d'avoir une mémoire distribuée avec plusieurs bancs mémoire programmables par processeur et un accès réseau par banc mémoire afin d'augmenter le débit données et adresses de la mémoire et du réseau. Notre approche, plutôt ciblée sur les applications de type gros traitement du signal, consiste à définir les ensembles de données distantes, locales et dans le cache sous forme de systèmes linéaires et à résoudre ces systèmes pour en déduire les codes de communication, de transferts mémoire, d'accès à la mémoire cache programmable et de calcul tout en pipelinant accès mémoire et calculs.

1. Introduction

Les améliorations classiques du modèle de style Von NEUMANN consistent à essayer d'augmenter le débit de calcul, le débit mémoire (la « quantité ») et enfin le débit d'adresse mémoire (la « qualité »), point souvent oublié. La solution actuelle classique au problème de débit d'adresse est basée sur l'usage de mémoire cache. Malheureusement, de nombreuses applications sont handicapées par la nécessité pratique d'accéder à la mémoire avec une certaine cohérence spatio-temporelle. D'autres, telles que les méthodes itératives de type échiquier, gaspillent de la bande passante mémoire en ne lisant qu'un élément sur deux et en n'écrivant aussi qu'un élément sur deux.

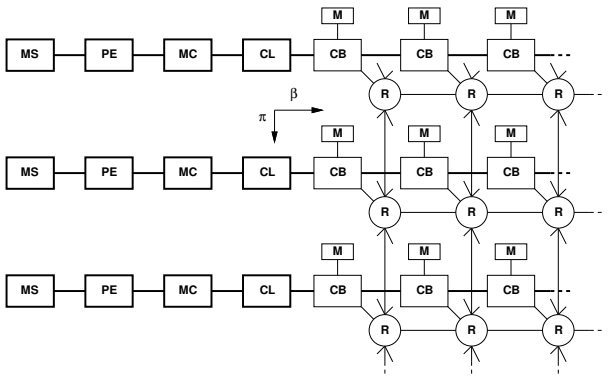


FIG. 1 – *Synoptique de la machine* PHÉNIX.

Dans le cas de la machine PHÉNIX en cours d'étude à l'ETCA[1] représentée sur la figure 1, le débit de calcul est atteint par du parallélisme MIMD à mémoire distribuée à partir de processeurs du commerce (PE avec de la mémoire scalaire MS). Afin d'améliorer le débit d'adresse au niveau de la mémoire, la mémoire de calcul de chaque processeur élémentaire est banquée (les bancs M) et afin de palier au faible débit d'adressage du processeur, un processeur d'adressage (contrôleur

1. Établissement Technique Central de l'Armement.

de banc CB) est rajouté au niveau de chaque banc mémoire M et permet d'exécuter du code localement. Un contrôleur de ligne CL permet de faire des échanges entre les bancs mémoire et la mémoire cache programmable MC. Dans la suite, nous considérons deux types de CL, un simple avec des possibilités d'adressage minimales, un plus complexe capable d'écrire ou de lire n'importe où dans la MC. Enfin, pour compenser le caractère très distribué de la mémoire, les bancs sont reliés entre eux par un réseau de type grilles par des contrôleurs de réseau R et la machine complète contient plusieurs plans de ce type interconnectés par des réseaux de type Omega en parallèle (non représentés ici) [4].

Plutôt que de définir un nouveau langage, nous préférons partir du langage parallèle classique HPF qui permet bien d'exprimer des applications de type traitement du signal et, en nous basant sur l'expérience acquise [1, 2], de compiler ce langage pour PHÉNIX. Par rapport aux machines classiques, une dimension de parallélisme est ajoutée : celle des bancs mémoire et du réseau. En adaptant le modèle d'exécution d'HPF qui a la caractéristique de n'être presque pas spécifié [5, pages 2,3, 10–19] on peut choisir qu'une dimension de processeurs HPF est en fait une dimension de processeurs virtuels dont la mémoire est représentée par chaque banc mémoire.

```
!HPF$INDEPENDENT(i)
     forall(Li ≤ b0(n))
           X(SX i + aX0(n)) = f(Y(SY i + aY0(n)),
                                Z(SZ i + aZ0(n)),
                                ...)
```

FIG. 2 – *Boucle générique.*

2. Modèle d'exécution

Nous nous intéressons pour l'instant aux nids de boucles parfaitement imbriquées avec des bornes de boucles et des fonctions d'accès aux éléments de tableaux de type `independent` (c'est-à-dire sans dépendances entre itérations) car

- ils permettent une compilation plus simple ;
- ils représentent souvent une grande partie du parallélisme disponibles dans les programmes qui nous intéressent ;
- on peut souvent se ramener à ce cas par des transformations de programme.

D'autre part, on utilise la « *owner computes rule* » pour distribuer les calculs, sachant qu'on peut simplement choisir une autre règle (voir l'introduction de [2]).

Le concept de bancs mémoire n'existe pas en HPF mais il est intéressant de contrôler leur utilisation de la même manière que les processeurs d'HPF puisque c'est un problème clé pour éviter les conflits dans les mémoires banquées. Plutôt que d'introduire une nouvelle directive de distribution, on décide qu'une dimension de processeur correspond en fait à la dimension des B bancs mémoire. Cette dimension peut

```
      real A(0:42), B(0:42)
!HPF$ template T(0:127)
!HPF$ processors P(0:3)
!HPF$ align A(i) with T(3*i)
!HPF$ align B(i) with A(i)
!HPF$ distribute T(cyclic(4)) onto P
      A(0:U:3) = A(0:U:3) + B(1:U+1:3)
```

FIG. 3 – *Exemple inspiré de* CHATTERJEE et al. *[3].*

```
array X'((c,ℓ) ∈ Own_X(p)),
      Y'((c,ℓ) ∈ View_Y(p)),
      Z'((c,ℓ) ∈ View_Z(p))
array X''(α ∈ Flush_X(p)),
      Y''(α ∈ Fill_Y(p)),
      Z''(α ∈ Fill_Z(p))
forall(U ∈ {Y',Z',...})                                          1°
      forall((p,p'),p ≠ p',Send_U(p,p') ≠ ∅)
            forall((ℓ,c) ∈ Send_U(p,p'))
                  send(p', U(ℓ,c))
forall(U ∈ {Y',Z',...})                                          2°
      forall((p,p'),p ≠ p',Receive_U(p,p') ≠ ∅)
            forall((ℓ,c) ∈ Receive_U(p,p'))
                  U(ℓ,c) = receive(p')
forall(U ∈ {Y',Z',...})                                          3°
      forall((ℓ,c) ∈ View_U(p))
            send_to_cache(U(ℓ,c))
forall(U ∈ {Y'',Z'',...})                                        4°
      forall(α ∈ Fill_U(π))
            receive_from_memory(U(α))
forall(α ∈ Compute(π))                                           5°
      X''(S_X''(α)) = f(Y''(S_Y''(α)),
                        Z''(S_Z''(α)),...)
forall(α ∈ Flush_X(π))                                           6°
      send_to_memory(X''(α))
forall((ℓ,c) ∈ Write_X(p))                                       7°
      receive_from_cache(X'(ℓ,c))
```

FIG. 4 –. Modèle d'exécution de la boucle générique figure 2.

aussi être vue microscopiquement comme une dimension de processeurs virtuels.

Les fonctionnalités offertes par cette équivalence permettent d'exprimer des distributions de données sous forme cyclique ou bloc via la directive `distribute`, des alignements de variables ou encore des répartitions de tableaux décalées pour éviter des conflits sur le bus de données via la directive `align`. Notons qu'on peut répliquer un tableau sur tous les bancs, ce qui peut être utile lorsqu'on fait certaines lectures avec des motifs irréguliers.

Considérons le nid de boucle générique de la figure 2 à exécuter sur PHÉNIX. Le modèle d'exécution choisi suit de près l'architecture matérielle et étend le modèle de [2] en introduisant la notion de banc mémoire β et de processeur élémentaire de ligne π et les projections (canoniques pour simplifier) $\beta = \boldsymbol{\beta}(p)$ et $\pi = \boldsymbol{\pi}(p)$.

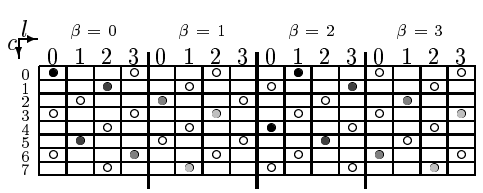


FIG. 5 – *Exemple de distribution et d'écriture correspondant à la figure 3 — bancs mémoire.*

Les directives HPF sont modélisées par le système d'équations 1 [2] et permettent par exemple de représenter figure 2 le *template* `T` et le tableau `A` de l'exemple figure 3 (les ronds représentent les éléments de tableau et les ronds noirs ou grisés les éléments accédés).

$$\left.\begin{array}{rc} \text{Alignement} & Rt = Aa + s_0 \\ \text{Distribution} & \Pi t = CPc + Cp + \ell \\ \text{Référence affine} & a = Si + a_0(n) \end{array}\right\} \quad (1)$$

Tout tableau `U` distribué au sens HPF se retrouve sous la forme d'un tableau local `U'` sur tous les bancs mémoire mais aussi sous forme d'un tableau `U''` dans la mémoire cache de chaque processeur élémentaire de ligne. Le modèle d'exécution consiste pour chaque itération i sur un processeur virtuel (c'est-à-dire en fait un banc mémoire) p à exécuter l'algorithme de la figure 4.

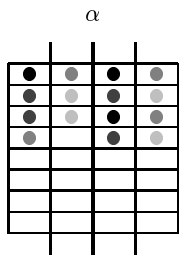


FIG. 6 – *Exemple de distribution et d'écriture correspondant à la figure 3 — cache vectoriel.*

3. Conclusion

Notre modèle de compilation permet d'étendre les distributions du langage à parallélisme de données standard HPF à des machines à mémoire banquée sans aucune modification du langage en unifiant la notion de processeur HPF et de banc mémoire suivant certaines dimensions. La génération de code pour les différents sous-systèmes d'une machine telle que PHÉNIX est automatisée en utilisant un formalisme utilisant l'algèbre linéaire pour compiler des nids de boucles à bornes affines. Enfin, en fonction de la complexité de la machine, différentes optimisations sont possibles et une solution permettant de pipeliner calculs et chargement mémoire est proposée.

Une version étendue de cet article avec le modèle de compilation peut être consultée dans `http://www.cri.ensmp.fr/rapports.html`.

4. Remerciements

Nous tenons à remercier Sylvain CARTIER et Philippe MICHALLON de l'équipe PHÉNIX, Corinne ANCOURT et Fabien COELHO du projet PIPS pour leur discussions sur le sujet et les arbitres du comité de programme pour leurs commentaires.

Bibliographie

1. Corinne ANCOURT, Fabien COELHO, François IRIGOIN, et Ronan KERYELL. « A Linear Algebra Framework for Static HPF Code Distribution ». Dans *CPC'93: Fourth Workshop on Compilers for Parallel Computers*, Delft, Hollande, décembre 1993.
2. Corinne ANCOURT, Fabien COELHO, François IRIGOIN, et Ronan KERYELL. « A Linear Algebra Framework for Static HPF Code Distribution ». Rapport Technique A-278-CRI, Centre de Recherche en Informatique, École Nationale Supérieure des Mines de Paris, France, novembre 1995. À paraître dans « *Scientific Programming* ». `http://www.cri.ensmp.fr/doc/A-278.ps.gz`.
3. Siddhartha CHATTERJEE, John R. GILBERT, Fred J. E. LONG, Robert SCHREIBER, et Shang-Hua TENG. « Generating Local Addresses and Communication Sets for Data-Parallel Programs ». *Journal of Parallel and Distributed Computing*, 26(1):72–84, avril 1995.
4. Pierre FIORINI. « *Réseaux d'Interconnexion pour les Architectures Parallèles. Les Delta-Pyramides et le Projet Phénix* ». Thèse, Université de Paris Sud, décembre 1994.
5. High Performance Fortran FORUM. « High Performance Fortran Language Specification ». Version 1.1, Center for Research on Parallel Computation, Rice University, Houston, USA, novembre 1994. `http://www.erc.msstate.edu/hpff/hpf-report-ps/hpf-v11.ps`.