HPF et l'état de l'art

Fabien COELHO (coelho@cri.ensmp.fr)
Centre de Recherche en Informatique, École des mines de Paris,
5, rue Saint-Honoré, F-77305 Fontainebleau Cedex, FRANCE.

Résumé

HPF (*High Performance Fortran*) est un nouveau langage de programmation standard pour machines à mémoire répartie. Il propose un modèle de programmation à parallélisme de données. Des directives et des constructions, qui précisent le placement des données et le parallélisme présent dans l'application, aident le compilateur. Cet article présente HPF et discute l'adéquation du langage à l'état de l'art en matière de compilation pour machines à mémoire répartie.

Introduction

« Toujours plus vite »: l'histoire de l'informatique scientifique, et plus particulièrement celle des supercalculateurs, semble suivre cette invective depuis les origines. Pour cela, les innovations technologiques et architecturales se sont succédées. L'une de ces innovations consiste à donner à ces machines une architecture parallèle massive, en composant un supercalculateur à partir de processeurs du commerce, souvent RISC, et en les reliant par un réseau de communication performant. La mémoire de ces machines est partagée si chaque processeur élémentaire (nœud) accède à la totalité de la mémoire. Elle est répartie, si un nœud n'accède directement qu'à sa mémoire locale, les accès à distance devant faire appel à d'autres techniques (échange de messages...).

Ce type d'architecture à mémoire répartie est particulièrement attrayant pour les constructeurs, car le principe même en est extensible : le nombre de processeurs sur le réseau de communication peut être étendu pour augmenter la puissance de la machine, sans que le lien mémoire processeur ne devienne *a priori* un goulot d'étranglement. En contrepartie de cette extensibilité, ces machines sont difficiles à programmer. Comme les données ne sont pas partagées par les processeurs, le programmeur doit gérer l'adressage et les communications entre nœuds, tâche difficile qui alourdie le développement. De plus, de par l'absence d'un modèle de programmation portable, la pérénité de ces investissements importants n'est pas assurée.

Deux voies sont explorées actuellement, qui visent à proposer sur ces machines des modèles de programmation similaires à ceux disponibles sur les machines traditionnelles. La première transmet le problème du maintien de la cohérence des données et de la génération des messages au système d'exploitation et au matériel [11], avec un modèle à parallélisme de tâche. La seconde s'appuie uniquement sur le compilateur [3, 7], en s'inspirant du parallélisme de données des machines SIMD. Dans les deux cas, les langages de programmation ont été étendus de manière à assister le processus de traduction, que ce soit Fortran S [2] qui aide le système d'exploitation à gérer la mémoire virtuelle partagée, ou Vienna-Fortran [4, 14] et Fortran D [9, 13] qui assistent le compilateur dans la génération des échanges de messages.

Les prototypes de recherche ont montré que cette dernière approche était réaliste. Des outils de ce type sont proposés par certains constructeurs, tel que TMC avec CM-Fortran. Cependant l'absence de portabilité de ces langages constituaient encore un frein à leur adoption. Pour répondre à ce besoin, un processus de standardisation s'est mis en place à l'initiative de DEC fin 91, pour aboutir à une spécication de langage, le *High Performance Fortran* (HPF) [6].

Dans cet article, nous discutons l'adéquation de HPF à l'état de l'art en matière de compilation pour machines à mémoire répartie, en décrivant les problèmes posés par la compilation de HPF. Les principales caractéristiques de HPF seront d'abord présentées en section 1. Dans la section 2, les différents problèmes posés par sa compilation seront exposés. Enfin quelques éléments de solution seront suggérés.

1 Présentation de HPF

À l'initiative de DEC, le monde du calcul parallèle (constructeurs, universitaires, utilisateurs) s'est réuni en Forum aux États-Unis, en 92-93, pour spécifier un nouveau langage de programmation pour les machines à mémoire répartie. L'objectif était de proposer un langage standard *de fait*, adopté par la plupart des constructeurs. Cette démarche est de nature commerciale, puisque l'existence d'un standard permet de rassurer l'acheteur de supercal-

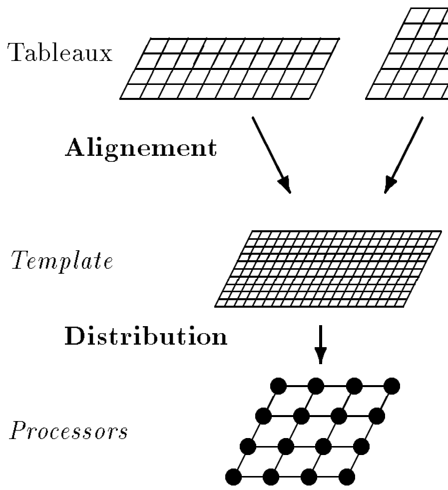

FIG. 1 - *Placement des données en HPF*

```
      real, dimension(100,100):: a,b
chpf$ template t(100,200)
chpf$ align a(i,j) with t(101-j,2*i)
chpf$ align b(*,i) with t(*,i)
chpf$ processors p(4,4)
chpf$ distribute t(block,cyclic(10)) onto p
chpf$ independent(i)
      do 10 i=1,100
 10     a(i,3)=b(10,2*i)
c transposition partielle
      forall(i=1:80,j=1:80) b(i,j)=b(j,i)
```

FIG. 2 - *Exemple de directives et boucles HPF*

culateurs parallèles quant à la durée de vie de ses investissements, et faire disparaître du même coup l'un des freins à l'expansion du marché de ces machines.

L'idée initiale, s'inspirant de Fortran D et Vienna-Fortran, était d'ajouter des directives de compilation à un langage séquentiel de manière à guider le compilateur sur le placement des données dans la mémoire répartie et sur l'exploitation du parallélisme. Le langage spécifié dépasse pour partie les objectifs assignés, et en laisse d'autres de côté, comme la gestion des entrées-sorties. HPF est basé sur le standard Fortran 90. Le choix d'un Fortran dans le domaine du calcul scientifique, langage auquel les utilisateurs sont habitués, s'imposait. HPF y ajoute des directives, de nouvelles structures et de nouvelles fonctions intrinsèques. Le reste de la section présente les directives de compilation de HPF pour le placement des données, puis les directives et structures parallèles.

1.1 Placement des données

En HPF le programmeur décrit le placement des données de son application dans la mémoire répartie de la machine au moyen de directives. Le compilateur exploite ces directives pour allouer la mémoire sur chaque processeur, gérer l'adressage et générer les communications. Ces directives permettent d'équilibrer (statiquement) la charge entre les processeurs (*i.e.* d'exploiter de parallélisme), tout en réduisant les communications qui constituent le goulot d'étranglement pour ces machines.

Le modèle de placement proposé par HPF comporte deux niveaux (figure 1). Les tableaux de données du programme sont placés sur une vue cartésienne des processeurs. Un objet intermédiare, le *template* (gabarit), sert à mettre en correspondance les données des différents tableaux relativement les uns aux autres, avec la directive d'alignement `align`. Il est ensuite découpé régulièrement avec la directive `distribute` pour répartir les données alignées sur chaque morceau de *template* entre les processeurs. Les alignements et distributions sont déclarés statiquement, et peuvent être modifiés dynamiquement avec les directives `realign` et `redistribute`. À l'entrée des procédures le programmeur peut également décrire le placement des paramètres attendus, le compilateur devant assurer les mouvements de données nécessaires.

L'alignement est lié à l'algorithme. Il caractérise la localité des données : deux éléments de différents tableaux placés sur un même élément de *template* seront finalement sur un même processeur. Techniquement, la directive d'alignement spécifie la mise en correspondance des éléments de tableau sur les éléments du *template* dimension par dimension, au moyen de fonctions entières affines d'une variable (figure 2). La directive d'alignement permet, par le jeu de ces fonctions affines, de transposer, retourner, écraser, éclater, dupliquer, décaller régulièrement les éléments des tableaux les uns par rapport aux autres.

La distribution est liée à l'architecture de la machine. Elle précise la granularité du parallélisme souhaitée. Chaque dimension du *template* peut être distribuée `block(n)` ou `cyclic(n)` sur une dimension des *processors*.

Ce modèle de placement, bien que donnant une grand lattitude à l'utilisateur, reste « régulier ». Il ne convient pas à tous les types d'application, mais se prête bien aux problèmes data-parallèles impliquant des calculs simples sur des géométries cartésiennes, qui sont la cible essentielle de ce langage.

1.2 Parallélisme et autres extensions

Le parallélisme est décrit explicitement en HPF au moyen de boucles parallèles et de fonctions de réductions. HPF ajoute deux « sémantiques » de boucle (figure 3) : la boucle indépendante (directive `independent`), où les itérations sont indépendantes les unes des autres, et la boucle à parallélisme de données, avec une notation implicite (sections de

```
chpf$ independent
      do 10 i = 1, 3
        lhsa(i) = rhsa(i)
 10     lhsb(i) = rhsb(i)
```

```
forall(i = 1:3)
  lhsa(i) = rhsa(i)
  lhsb(i) = rhsb(i)
endforall
```

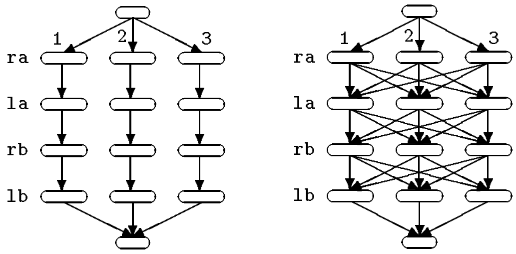

FIG. 3 - *Boucles parallèles de HPF*

tableau : `a(2:n)=a(1:n-1)`) et une forme explicite avec l'instruction `forall`.

Un grand nombre de fonctions intrinsèques de calcul de fonctions réductions et à préfixe parallèle sont également mises à disposition du programmeur, pour calculer les minimum, maximum et leur localisation, les somme, produit, « et » et « ou » logiques des éléments d'un tableau, etc. Certaines de ces fonctions proviennent de Fortran 90.

HPF ajoute également des fonctions intrinsèques nouvelles, des fonctions d'enquête système permettant de connaître à l'exécution les caractéristiques disponibles de la machine (comme le nombre de processeurs) et la possibilité d'importer des fonctions externes à HPF en déclarant une interface... Se rendant compte de la taille et de la difficulté du langage pour les compilateurs, le Forum a spécifié un sous-ensemble « minimum » de HPF. Il comprend principalement Fortran 77, les directives de placement statique et une forme réduite de `forall`.

2 Compilation de HPF

Les compilateurs devront être performant, car les utilisateurs attendent de ces machines de hautes performances. Les principaux problèmes rencontrés portent sur la taille du langage, la sémantique étendue du placement, l'enumération des itérations d'une boucle locale à un processeur, la génération des messages, les appels de procédures avec des paramètres répartis, les entrées-sorties et les fonctions intrinsèques.

Fortran 90 [10] est le dernier standard Fortran. Il ajoute à Fortran 77 les caractéristiques des langages impératifs modernes telles les structures, les pointeurs, l'allocation dynamique, les modules et une syntaxe étendue. Le résultat d'une décennie de standardisation est un très gros langage dont les premiers compilateurs commerciaux - pour machines séquentielles - commencent seulement à apparaître. Le langage est si complexe que le DoD (ministère de la défense américain) l'a rejetté pour l'exécution de ses contrats.

HPF propose une sémantique de placement « régulier » très générale. Ce placement n'est parfois connu qu'à l'exécution (réalignements et redistributions, appels de procédures, paramétrage du placement par des fonctions d'enquête système), les informations sont alors très réduites pour compiler efficacement. De plus, les placements impliquant des alignements non unitaires ($ai+b, a \neq 1$) et des distribution cycliques généralisées (`cyclic(`n`)`, $n \neq 1$) sont pour l'instant hors de l'état de l'art. Enfin la gestion de la duplication des données complique le maintien de la cohérence et les algorithmes de génération de messages. À partir de la description du placement, le compilateur doit allouer la mémoire de manière réduite sur chaque processeur tout en garantissant un adressage efficace, malgré certaines irrégularités permises par le placement.

La compilation performante des boucles parallèles implique d'abord la génération d'un code efficace (de surcoût de contrôle minimal) permettant d'énumérer les itérations locales à chaque nœud. Cette énumération doit être liée à la méthode d'allocation choisie pour les données, pour exploiter la mémoire cache de la machine. Les boucles parallèles impliquent le plus souvent des communications, certaines données n'étant pas disponible localement. Il faut déterminer à la compilation ces ensembles, quelle que soit la répatition des différentes données entre les processeurs. Le compilateur doit également allouer les tableaux temporaires pour stocker, le temps des calculs, ces données distantes et assurer l'adressage de ces temporaires plutôt que des données locales quand nécessaire... Enfin l'utilisateur attend que les caractéristiques particulières de la machine soient exploitées.

Le passage de paramètres répartis au niveau des appels de procédure pose aussi un problème : Fortran 90 permet de passer des sous-tableaux de tableau comme paramètre, ce qui complique la tâche du compilateur lorsque le tableau initial est déjà réparti. Plusieurs compilations sont nécessaires en fonction des différents contextes d'appels. De plus, même si les réalignements et redistributions explicites ne sont pas dans le sous-ensemble de HPF, les redistributions à l'entrée et à la sortie des procédures nécessitent la génération de code comparable.

Lors de la spécification de HPF, le Forum n'a pas trouvé de point d'accord pour le traitement des entrées-sorties. Il n'y a pas de concensus sur un paradigme unique d'entrées sorties parallèles. Il faut exploiter la structure particulière du système d'E/S parallèles de la machine cible pour stocker par morceau les données, et pouvoir récupérer une description du stockage parallèle des données pour les recharger en mémoire ultérieurement. Ce problème déliquat nécessite une aide du programmeur pour le compilateur, et est à l'ordre du jour du second tour de discussion du Forum HPF qui a repris en janvier 94.

Enfin HPF possède de nombreuses fonctions intrinsèques, venant à la fois de Fortran 90 et de la nouvelle spécification. De plus ses fonctions sont souvent très paramétrables. La première difficulté est leur nombre : il faut écrire un code correct et efficace pour chacune de ces fonctions, même pour un ordinateur séquentiel. La seconde difficulté est d'un autre ordre de grandeur : elles doivent être implantées de manière à fonctionner en parallèle, quelle que soit la répatition sur la machine des tableaux passés en argument.

Conclusion

De part ces nombreux problèmes, le développement d'un compilateur complet et optimisant pour HPF est une entreprise qui semble au delà de l'état de l'art. Les prototypes de compilateur pour des langages comparables qui existent actuellement ne gèrent que des alignements qui sont de simples décalages, et des distributions par bloc ou cycliques simples ($n = 1$). L'allocation des temporaires et la génération des communications sont performantes quand les motifs d'echange de messages sont réguliers, dans le cas de recouvrements au voisinage : les données distantes sont stockées en élargissant les déclarations locales des tableaux, ce qui permet en plus d'adresser ces données comme si elles étaient locales. Des librairies optimisées implantent les algorithmes ou communications courants (transposition, inversion de matrice, etc.).

Pour le reste, on doit faire appel à des techniques de résolution dynamique des accès aux variables réparties : chaque accès à un élément de tableau éventuellement distant doit être gardé de manière à vérifier sa localité et générer les communications si nécessaires. Cependant de telles techniques sont incompatibles avec une exécution efficace des codes. Les compilateurs utilisent les routines « `PARTI` » qui permettent de déterminer dynamiquement les communications et réutiliser plusieurs fois les motifs déterminés.

HPF hors de l'état de l'art ? Des techniques ont été proposées qui permettent de gérer des cas de placements complexes en HPF, en s'appuyant sur les automates [5], les sections de tableau [8, 12] ou la théorie des polyèdres [1], qui poussent à l'optimisme. Cependant il est difficile de déterminer si la lourdeur du développement des compilateurs l'emportera sur la petite taille du marché attendant ces derniers. Il est probable que très peu de compilateurs commerciaux seront disponibles même si portés sur de nombreuses machines. La réussite rapide de ces projets, et l'efficacité que montreront les compilateurs dans la pratique, conditionnent l'avenir de ce langage.

Références

[1] C. Ancourt, F. Coelho, F. Irigoin, and R. Keryell. A linear algebra framework for static HPF code distribution. In *Workshop on Compilers for Parallel Computers, Delft*, Dec. 1993. Also available as TR EMP A/250/CRI.

[2] F. Bodin, L. Kervella, and T. Priol. Fortran-S: A Fortran Interface for Shared Virtual Memory Architectures. In *ACM International Conference on Supercomputing*, Nov. 1993.

[3] D. Callahan and K. Kennedy. Compiling programs for distributed-memory multiprocessors. *The Journal of Supercomputing*, 2:151–169, 1988.

[4] B. Chapman, P. Mehrotra, and H. Zima. Programming in Vienna Fortran. *Scientific Programming*, 1(1):31–50, 1992.

[5] S. Chatterjee, J. R. Gilbert, F. J. E. Long, R. Schreiber, and S.-H. Teng. Generating local addresses and communication sets for data-parallel programs. In *Symposium on Principles and Practice of Parallel Programming*, 1993.

[6] H. P. F. Forum. *High Performance Fortran Language Specification.* Rice University, Houston, Texas, May 1993. *Version 1.0.*

[7] H. M. Gerndt. *Automatic Parallelization for Distributed-Memory Multiprocessing Systems.* PhD thesis, University of Vienna, 1989.

[8] S. K. S. Gupta, S. D. Kaushik, S. Mufti, S. Sharma, C.-H. Huang, and P. Sadayappan. On compiling array expressions for efficient execution on distributed-memory machines. In *International Conference on Parallel Processing*, pages II–301–II–305, Aug. 1993.

[9] S. Hiranandani, K. Kennedy, and C.-W. Tseng. Compiling Fortran D for MIMD Distributed-Memory machines. *Communications of the ACM*, 35(8):66–80, Aug. 1992.

[10] ISO/IEC. *International Standard ISO/IEC 1539:1991 (E)*, second 1991-07-01 edition, 1991.

[11] K. Li. *Shared Virtual Memory on Loosely Coupled Multiprocessors.* PhD thesis, Yale University, Sept. 1986.

[12] J. M. Stichnoth. Efficient compilation of array statements for private memory multicomputers. CMU-CS-93 109, School of Computer Science, Carnegie Mellon University, Feb. 1993.

[13] C.-W. Tseng. *An Optimising Fortran D Compiler for MIMD Distributed Memory Machines.* PhD thesis, Rice University, Houston, Texas, Jan. 1993.

[14] H. Zima and B. M. Chapman. Compiling for distributed-memory systems. *Proceedings of the IEEE*, Feb. 1993.