HAL
open science

Évaluation de performances dans l'environnement Pandore II

Yves Mahéo

To cite this version:

Yves Mahéo. Évaluation de performances dans l'environnement Pandore II. 5e rencontres sur le parallélisme (RenPar'93), May 1993, Brest, France. pp.47-50. hal-00426672v2

HAL Id: hal-00426672

https://hal.science/hal-00426672v2

Submitted on 5 Jul 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Évaluation de performances dans l'environnement Pandore II

Yves Mahéo
IRISA Campus de Beaulieu
35042 Rennes cedex
maheo@irisa.fr

1 Introduction

La programmation des architectures parallèles à mémoire distribuée (Apmd) est réputée complexe. Une des solutions proposées est de permettre à l'utilisateur d'exprimer son programme dans un langage séquentiel impératif et de laisser au compilateur la tâche de produire un code distribué. L'état de l'art actuel ne permet pas d'envisager une distribution entièrement automatique, le programmeur doit faciliter le travail du compilateur en donnant un certain nombre d'indications sur son programme. Ceci peut être fait en spécifiant le découpage et la répartition des données, le compilateur pouvant en déduire un ensemble de processus communicants. Cette approche, qui vise essentiellement les applications numériques, a été adoptée dans le système Pandore développé au sein de l'équipe PAMPA à l'IRISA. La validité de cette technique a été montrée mais l'efficacité des codes obtenus pourrait être améliorée. Nous présentons ici l'environnement Pandore II dans son ensemble et plus particulièrement un outil aidant à l'évaluation des performances des codes générés.

2 L'environnement Pandore II

L'environnement Pandore II exploite l'approche de la distribution de programmes impératifs séquentiels par distribution des données [1]. Il se compose d'un compilateur, d'exécutifs pour plusieurs Apmd et d'outils d'analyse d'exécution.

2.1 Le langage source

Le langage de programmation Pandore II est un sous-ensemble du langage C auquel est ajouté un constructeur, la phase distribuée, permettant de répartir les données. Les phases distribuées apparaissent dans le code source comme des procédures (précédées du mot clé `dist`) pour lesquelles, à chaque paramètre, est associée une spécification de répartition. Le découpage des tableaux se fait par blocs rectangulaires dont le placement sur les processus peut être régulier ou cyclique. Par exemple, la définition de l'entête de phase distribuée

```
dist myphase(float A[N][N] by block(N,1) map wrapped(0,1) mode INOUT)
```

indique que le tableau `A` est découpé en colonnes réparties de façon cyclique sur les processus. Le mode indique que `A` est un paramètre d'entrée et de sortie de la phase. La figure 1 montre un exemple de programme Pandore II.

2.2 Le compilateur

A partir du texte source, le compilateur génère un ensemble de processus selon le modèle *hôte/nœuds*. Le processus hôte exécute le code du programme principal, contrôle l'enchaînement des phases distribuées et les entrées/sorties. Les processus nœuds effectuent les calculs décrits dans les phases distribuées. Le schéma de compilation est basé sur la règle dite des écritures locales. Un processus n'exécute que les instructions modifiant les données qu'il possède. Si certaines des variables lues sont placées sur un autre processus, des communications seront générées. Un autre aspect du schéma de compilation est la production d'un code SPMD, c'est-à-dire d'un code identique pour tous les processus mais agissant sur des données différentes. Ainsi, l'affectation $A[i] = B[i+1] + C[i]$ où A, B et C sont des tableaux distribués sera traduite par la séquence d'opérations suivante :

```
refresh({tmp1,tmp2}, {B[i+1],C[i]}, owner(A[i]))
exec(owner(A[i]), tmp1+tmp2)
free({tmp1,tmp2})
```

Lors de l'exécution du `refresh`, les possesseurs de $B[i+1]$ et $C[i]$ envoient leur valeur et le possesseur de $A[i]$ les reçoit dans les temporaires *tmp1* et *tmp2*. La macro `exec` assure le masquage de l'affectation $A[i]$ =*tmp1*+*tmp2* qui n'est exécutée que par le possesseur de $A[i]$. Lorsque la variable affectée est dupliquée sur tous les processus – c'est le cas pour les scalaires –, les valeurs distantes sont donc diffusées sur le réseau. Ce schéma de base ne produit pas en général de bonnes performances ; un schéma optimisé, basé sur l'analyse statique des boucles est actuellement mis en œuvre.

2.3 L'exécutif

L'exécutif PANDORE II permet l'exécution du code produit par le compilateur sur différentes architectures cibles. Il met en œuvre la génération des processus, les mouvements de données, les masquages d'instructions et les accès aux tableaux distribués à l'aide des primitives offertes par le système cible. Il est formé d'un ensemble de macros `cpp` organisées selon deux niveaux. Le premier niveau constitue l'interface avec le compilateur, le code généré ne fait appel qu'aux macros de ce niveau (`refresh`, `exec`, ...). Le second niveau réalise une mise en œuvre du modèle de machine d'exécution sur l'architecture cible. Le modèle retenu repose sur un réseau de processeurs complètement maillé dont l'un joue le rôle du processeur hôte. Les processeurs communiquent par l'intermédiaire de canaux bidirectionnels fiables et FIFO, les émissions étant non bloquantes et les réceptions bloquantes. La structuration de l'exécutif offre une grande souplesse tant du point de vue de la portabilité du système PANDORE II (seul le second niveau est à modifier) que du point de vue de l'expérimentation de nouveaux schémas de compilation. Elle permet également de faciliter la mise en place d'une instrumentation du code généré.

3 Instrumentation

Les performances du code généré par le système PANDORE II dépendent de l'adéquation de la distribution des données spécifiées par l'utilisateur à l'algorithme utilisé mais aussi des choix de mise en œuvre du compilateur et de l'exécutif. Il apparaît donc qu'une évaluation de l'influence de ces paramètres soit nécessaire d'une part pour proposer à l'utilisateur des outils l'aidant à distribuer correctement les données de son programme et d'autre part pour guider les choix des concepteurs du système. Une évaluation dynamique des programmes PANDORE II – par opposition à une estimation statique [2, 3] – offre l'avantage de pouvoir s'appliquer à tous les types de programmes et de permettre une bonne précision des résultats.

3.1 Technique de mesure

Les deux techniques principalement utilisées pour les mesures de performance sont la génération de traces et le profiling. La génération de traces permet d'enregistrer des événements auxquels sont assignés au moins un type et une estampille. L'activité du programme peut ainsi être enregistrée de façon précise. Le principal inconvénient est l'importance du volume des traces générées : l'espace mémoire nécessaire croît avec la durée de l'exécution.

Le profiling consiste à maintenir à jour durant l'exécution un nombre fixe de compteurs reliés à des événements. Une extension de cette technique est utilisée ici : pour chaque événement considéré, en plus de compter son occurrence, on cumule sa durée [4]. Comme le profiling collecte uniquement des totaux, l'espace mémoire requis est déterminé statiquement. Les points d'instrumentation sont insérés dans des versions modifiées de certaines macros du premier niveau de l'exécutif. Le compilateur génère donc un code similaire, que l'on désire ou non des mesures à l'exécution.

Pour être exploitables, les résultats obtenus doivent impérativement pouvoir être mis en relation avec le programme source. Ceci est fait de deux façons : tout d'abord, l'utilisateur définit des *zones d'instrumentation* ; pour chacun de ces fragments de corps de phases distribuées, une série de mesures est effectuée sur les nœuds puis rapatriée sur l'hôte en fin d'exécution. Ensuite, les résultats numériques sont associés aux objets du programme apparaissant dans les zones d'instrumentation : tableaux, scalaires, conditionnelles, boucles.

3.2 Résultats

Outre les temps d'exécution, les principaux résultats fournis par l'exécution d'un code instrumenté concernent les communications et les synchronisations. On peut classer ces résultats en deux catégories : les mesures propres aux phases distribuées (temps de communication avec l'hôte au début et à la fin de chaque phase, temps d'enchaînement des phases) et les mesures rendant compte des affectations dans les zones d'instrumentation.

Ces statistiques donnent des informations sur l'efficacité de la mise en œuvre de l'exécutif. La dernière catégorie permet également d'évaluer la distribution des données. Une affectation d'un élément de tableau distribué par une expression contenant une référence à un tableau distribué peut générer un message entre le possesseur de l'élément en partie droite et le possesseur de l'élément en partie gauche. Le but des mesures est de globalement constituer un graphe dirigé dont les nœuds sont les partitions de variables, les arcs décrivant le trafic entre les deux partitions. Les arcs peuvent être valués par le nombre de messages, le volume transféré ou le temps d'attente sur réception. Par exemple, l'affectation $A[3,5] = B[4]$ augmentera la valeur de l'arc $(B1 \rightarrow A2)$ si l'élément $A[3,5]$ se trouve sur le processeur 2 et $B[4]$ se trouve sur le processeur 1.

De même, une affectation d'une variable dupliquée par un élément de tableau distribué génère systématiquement la diffusion d'un message vers tous les nœuds. On décrit, pour chaque couple (*var,part*), les diffusions partant de la partition *part* dues à une affectation de la variable dupliquée *var* selon le nombre de diffusions, leur volume et le temps d'attente sur les receptions. Par exemple, l'affectation $x = A[3,5]$ augmentera la valeur des compteurs liés au couple $(x, A1)$ si l'élément $A[3,5]$ se trouve sur le processeur 1.

3.3 Exemple

Afin d'illustrer l'obtention du graphe de communications entre partitions, considérons le programme de la figure 1 exécuté sur quatre processeurs. Après examen de la première zone, on partitionne le vecteur V en blocs de N/P éléments et la matrice A en groupes de N/P lignes, une alternative serait de tenir compte plutôt de la deuxième zone et de partitionner A en groupes de N/P colonnes (en

```
#define  N   128
#define  P   4

float A[N][N], B[N][N], float V[N];

dist myphase( float A[N][N] by block(N/P,N) map regular(0,1) mode INOUT,
              float V[N] by block(N/P) map regular(0) mode INOUT)
{
   int i,j;
   for (i=0; i<N; i++)                    /*                          */
      for (j=0; j<N; j++)                 /* Zone d'instrumentation 1 */
         V[i] = f(V[i],A[i][j]);          /*                          */

   for (j=0; j<N; j++)                    /*                          */
      for (i=1; i<N-1; i++)               /* Zone d'instrumentation 2 */
         A[i][j] = g(A[i+1][j], A[i-1][j]); /*                       */
}

main()
{ myphase(A,V); }
```

Figure 1 : Programme source Pandore II

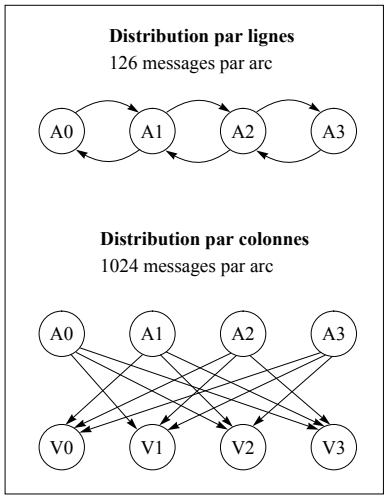

Figure 2 : Graphes des communications

inversant les paramètres de la fonction **block**). La figure 2 donne les graphes de communications pour les deux distributions. Les communications n'interviennent que dans la zone 1 pour la distribution par colonnes et uniquement dans la zone 2 pour la distribution par lignes. Le choix du partionnement par lignes semble donc préférable. Ceci est confirmé par le graphe des temps d'attente sur réception qui fait apparaître de fortes synchronisations pour la version distribuée par colonnes.

4 Conclusion

Si l'approche de la distribution de programmes séquentiels impératifs par décomposition et répartition des données est maintenant reconnue, elle pose encore des problèmes liés aux performances. Nous avons présenté PANDORE II, un environnement de programmation suivant cette approche. Un outil intégré à l'environnement et basé sur la méthode du profiling permet l'analyse des performances des codes distribués. Il s'est déjà révélé utile à l'amélioration du système. Néanmoins, pour pouvoir guider efficacement l'utilisateur dans le choix des distributions de données, les paramètres à mesurer doivent être définis plus précisement, l'interprétation des résultats devant impérativement rester à la portée du programmeur.

Bibliographie

[1] Françoise André, Olivier Chéron, and Jean-Louis Pazat. *Compiling Sequential Programs for Distributed Memory Parallel Computers with Pandore II.* Technical Report 651, IRISA, April 1992.

[2] V. Balasundaram, G. Fox, K. Kennedy, and U. Kremer. A Static Performance Estimator to Guide Data Partitioning Decisions. In *The Third ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, June 1991.

[3] T. Fahringer, R. Blasko, and H.P. Zima. Automatic Performance Prediction to Support Parallelization of Fortran Programs for Massively Parallel Systems. In *Proceedings of the '92 International Conference on Supercomputing*, pages 347–356, ACM press, July 1992.

[4] Carl Kesselman. *Tools and Techniques for Performance Measurment and Performance Improvement in Parallel Programs.* PhD thesis, UCLA, July 1991.