CIDRE : une bibliothèque pour la distribution de tableaux partagés

F. André et Y. Mahéo

IRISA, Campus de Beaulieu, 35042 Rennes cedex
Email : (fandre | maheo)@irisa.fr

1 Introduction

La modularité et l'extensibilité constituent deux points forts incontestables des architectures à mémoire distribuée. Cependant, leur programmation demeure assez complexe en raison de leurs mémoires distribuées qui contraignent à utiliser des opérations de communication. Le modèle de programmation qui à ce jour fait l'unanimité pour des applications importantes est celui de la programmation parallèle à variables partagées. Ce modèle permet de mettre en évidence le parallélisme qui sera source de performances tout en gardant une vue globale des structures de données manipulées.

L'amélioration de l'utilisation des architectures à mémoire distribuée passe donc par la mise en œuvre du modèle de processus communiquant par variables partagées sur ces architectures. Les systèmes de mémoire virtuelle partagée constituent une solution à ce problème. Les variables partagées sont placées dans la mémoire virtuelle et sont donc adressées de manière uniforme. Cependant plusieurs inconvénients viennent tempérer cette apparente facilité d'emploi. Le plus important est lié au coût des communications des pages virtuelles. La taille des pages est indépendante de celle des variables accédées : il s'ensuit un volume de communication plus important que celui nécessairement requis. Mais surtout, lorsqu'une page contient plusieurs variables accédées en parallèle par différents processeurs, des scénarii de communications «ping-pong» se produisent.

Notre objectif est de proposer une alternative à ce système de mémoire virtuelle partagée sous forme d'une bibliothèque de tableaux partagés. Vis à vis des divers travaux concernant les mémoires virtuelles partagées dont on pourra trouver une bibliographie conséquente dans [2], CIDRE se caractérise par une technique de mise en œuvre des tableaux partagés originale et

efficace sur les deux points essentiels de leur gestion. Sur le plan de l'adressage : elle offre à l'utilisateur un adressage uniforme de ses variables mais l'interprétation en terme d'adresses locales dans les mémoires distribuées est optimisée grâce à une technique de pagination logique des adresses. Sur le plan des communications : elle utilise des communications directes des éléments requis mais profite au maximum des techniques de vectorisation, en liaison avec le mécanisme de pagination logique pour limiter le nombre et le volume des communications.

2 Le modèle de programmation

Le modèle de programmation repose sur l'existence de processus parallèles. La décomposition en processus est du ressort du concepteur de l'application. Les processus peuvent être, comme c'est souvent le cas lors de l'utilisation d'architectures fortement parallèles, bâtis sur le même modèle. Typiquement nous avons un code SPMD dans lequel chaque processus, selon son identité, détermine le travail qu'il doit faire.

L'exemple de la figure 1 illustre ce cas. P processus exécutent le code décrit (qui représente un algorithme de type Jacobi). Les variables partagées sont déclarées en appelant la fonction de bibliothèque `create`. Cette fonction prend en paramètre les dimensions des tableaux puis leur répartition, qui peut être cyclique ou par blocs, sur les P processeurs de l'architecture. Dans notre exemple, les matrices A et B de taille $N \times N$ sont créées. Elles sont toutes deux décomposées en blocs de taille $N/P \times N$. L'adressage des variables partagées est global, sans mention explicite de la localisation de l'élément de tableau référencé.

Dans cet exemple le processeur P_i possède le bloc de lignes $(N/P \times i)$ à $(N/P \times (i+1) - 1)$. Pour effectuer ses calculs à l'étape k, il a besoin de la ligne $(N/P \times i - 1)$, après qu'elle ait été calculée par le processeur P_{i-1} à l'étape $k-1$, et de la ligne $(N/P \times (i+1))$ calculée par P_{i+1} à l'étape $k-1$. Réciproquement : P_{i-1} a besoin de la ligne $(N/P \times i)$ et P_{i+1} a besoin de la ligne $(N/P \times (i+1) - 1)$. Pour assurer que chaque processeur travaille sur des données à jour à l'étape k (c'est-à-dire résultant de l'étape $k-1$), nous utilisons deux opérations de synchronisation–cohérence par processus. La première coordonne les processeurs P_i et P_{i-1} pour l'échange de leurs lignes $(N/P \times i - 1)$ et $(N/P \times i)$; la seconde coordonne les processeurs P_i et P_{i+1} pour l'échange des lignes $(N/P \times (i+1) - 1)$ et $(N/P \times (i+1))$.

D'une manière générale, la fonction `coherence` précise les éléments d'un

tableau partagé (section de tableau spécifiée par un triplet borne inférieure–borne supérieure–pas dans chaque dimension) qui sont à rendre cohérents pour le groupe de processus cités en dernier paramètre de cette fonction. Après l'exécution de la fonction, tous les processus du groupe sont assurés d'avoir la même vision de la structure de donnée partagée.

---

process `myself`

A = `create`(N, N, `BLOCK`, N/P, `BLOCK`, N) ; B = `create`(N, N, `BLOCK`, N/P, `BLOCK`, N)
prev_set = {`myself`, `myself`−1} ; *next_set* = {`myself`, `myself`+1}
first_line = N/P*`myself` ; *last_line* = N/P*(`myself`+1) −1

for k=1 **to** *nloop*
 if (`myself` ≠ 0) `coherence`(A, *first_line*−1, *first_line*, 1, 0, $N-1$, 1, *prev_set*)
 if (`myself` ≠ $P-1$) `coherence`(A, *last_line*, *last_line*+1, 1, 0, $N-1$, 1, *next_set*)

 for i = *first_line* **to** *last_line*
  for $j = 0$ **to** $N-1$
   `write`(B,i,j, f(`read`(A,$i+1$,j), `read`(A,$i-1$,j), `read`(A,i,$j+1$), `read`(A,i,$j-1$)))

 for i = *first_line* **to** *last_line*
  for $j = 0$ **to** $N-1$
   `write`(A,i,j, `read`(B,i,j))

FIG. 1 - *Exemple de code: algorithme de Jacobi*

3 Mémoire paginée logique

La gestion des accès aux tableaux distribués de la bibliothèque CIDRE est issue de celle utilisée dans le compilateur HPF Pandore [1]. Elle est fondée sur la pagination logique des tableaux dirigée par la distribution en blocs rectangulaires qui a été spécifiée par l'utilisateur lors de la création du tableau. L'objectif est de fournir un accès élémentaire rapide tout en maintenant à un niveau raisonnable le surcoût mémoire induit par la représentation.

L'espace d'adresses multi-dimensionnel défini par chaque tableau est linéarisé et découpé en pages. Ces pages sont utilisées pour stocker les données locales et les copies temporaires des données distantes. Les éléments sont accédés uniformément : à partir du vecteur d'indices globaux sont calculés un numéro de page et un offset dans cette page. Ce couple (PG,OF) permet d'accéder, via la table des pages stockée sur chaque processeur, à l'élément mémoire correspondant. Pour chaque tableau, on définit une direction des

pages et une taille des pages en fonction des paramètres de distribution du tableau. La direction des pages correspond à la dimension selon laquelle la taille des blocs est la plus grande. On choisit comme taille de page une puissance de deux afin d'accélérer les calculs d'adresse : le calcul du couple (PG,OF) ne fait intervenir que des opérations logiques peu coûteuses (décalages, masques).

Outre les accès élémentaires efficaces, la gestion paginée des tableaux permet l'optimisation des communications qui peuvent être mises en jeu lors de l'opération de synchronisation-cohérence. Les communications sont organisées en *segments* (un segment est une portion contiguë au sein d'une page). Pour les gros segments, on utilise des communications directes dans lesquelles on transfère une zone mémoire contiguë à la fois chez l'émetteur et chez le récepteur sans qu'il soit nécessaire de passer par un tampon de communication. Les petits segments sont eux agrégés dans un tampon plus grand afin de minimiser l'effet de la latence des messages. Une description détaillée de ces mécanismes peut être trouvée dans [3].

4 Conclusion

La bibliothèque CIDRE met l'accent sur la gestion mémoire et les communications relatives aux tableaux partagés, avec le souci de performances. Elle se situe donc juste au dessus du système de communication de la machine cible. Elle est essentiellement destinée à être intégrée dans un environnement de programmation de plus haut niveau qui offrirait par exemple des protocoles de cohérence de données évolués ainsi qu'un langage d'interface avec l'utilisateur plus convivial que celui utilisé dans notre exemple.

Références

[1] F. André, M. Le Fur, Y. Mahéo, and J.-L. Pazat. The Pandore Data Parallel Compiler and its Portable Runtime. In *HPCN Europe '95*, LNCS 919, Springer Verlag, Milan, Italie, mai 1995.

[2] M.R. Eskicioglu. A Comprehensive Bibliography of Shared Distributed Memory. *ACM Operating Systems Review*, 30(1):71–96, janvier 1996.

[3] Y. Mahéo. *Environnements pour la compilation dirigée par les données: supports d'exécution et expérimentations.* PhD thesis, IFSIC / Université de Rennes I, juillet 1995.