共有メモリ型並列計算機におけるメモリアクセスの局所化技法

2D−10

古川浩史　松本尚　平木敬

東京大学理学部情報科学科

1 はじめに

共有メモリ型並列計算機では、主記憶参照のコストが高いために、キャッシュのヒット率向上が性能向上の鍵となる。従来よりメモリ参照の局所性を高め、キャッシュヒット率を向上させる技法が提案されていた([2][3][5])。これらは、loop permutation や blocking といった技法を適用するものであるが、配列のメモリ上でのレイアウトについてはあらかじめ固定してあるか、せいぜい row major/column major を決定するものであったために、どうしても局所性を抽出できない場合があった。本稿では、UMA 型共有メモリ計算機において、プログラムの大域的な情報を元に、より一般的な配列のレイアウトおよびループ変換を同時に決定をする手法を提案する。

2 問題の定式化

r 次元配列A のレイアウトとは、添字 $\vec{j}=(j_1,j_2,\ldots,j_r)\in J$ からアドレス $a\in A$ への写像 $L_A: J\rightarrow A$ のことをいう。例えば、Fortran では、配列A$(N_1,N_2,\ldots,N_r)$ のレイアウトは、

$$L_A(\vec{j})=a_0+(j_1-1)+N_1\times(j_2-1)+\ldots+(\prod_{k=1}^{r-1}N_k)\times(j_r-1)$$

と表される。ここで a_0 は配列A の先頭アドレスである。本稿では、レイアウトを添字の線形式に限る。すなわち、

$$L_A(\vec{j})=a_0+\sum_{k=1}^{r}a_k\times j_k$$

とし、$\{a_k|1\leq k\leq r\}$ を求める。

また、配列の添字式をループインデックスのアフィン形式に限る。インデックス空間を $\vec{i}=(i_1,i_2,\ldots,i_l)\in I$ とすると (l はループネストの深さ)、添字式は

$$\vec{f}: I\rightarrow J=F\vec{i}+\vec{k}$$

とかける。ここで F は線形変換であり、$\vec{k}$ はループ定数のベクトルである。

3 アルゴリズム

以下の手順に従って、配列のレイアウトを求める。

まず、[6] に述べられている手法を用いて、内側が fully permutable になるようにプログラム中の各ループネストを変形する。

キャッシュの再利用は、ブロック化した場合はブロック内、そうでなければ最内側ループで行なわれると考えられるので、ループネスト l のインデックス空間 I_l に対し、$I'_l\subseteq I_l$ を次のように定義する。

- l が fully permutable なループを持つ時は、I'_l を fully permutable なループすべてのインデックスが張る空間とする。
- そうでないときは、I'_l は最内側のループのインデックスが張る空間とする。

次に、プログラム中の各配列について、以下を実行する。

1. 注目している配列への参照が行なわれるループネストを l とする。l 内の i 番目の配列参照について、添字式を表す変換を F_{li} とする。
2. $D=\bigcap_{l,i,F_{li}(I'_l)\neq 0}F_{li}(I'_l)$ を求める。$rank(D)=0$ となる場合には、$rank(D)>0$ となるまで実行時間がもっとも短いループネストを除いて D を求め直す。
3. キャッシュラインの方向を D の方向に合わせる。$rank(D)>1$ の場合には D 内の任意の方向でよいので基底ベクトルのなるべく簡単な形の結合を選ぶ。以下、D は一次元とし、D の方向ベクトルを $\vec{d}=(d_1,\ldots,d_r)$ とする。

   まず、$\{d_k|1\leq k\leq n\}$ のうち、最大のものを d_i とする。つぎに、$\vec{d}$ の各要素を d_i で割ったものを $\vec{d'}$ とする。割り切れない場合は丸める。

   単位行列 I の i 番目の列を $\vec{d'}$ にし、1列目の列ベクトルと入れ換えた行列を P とする。

Memory Access Localization for Shared Memory Multiprocessors

Hiroshi FURUKAWA, Takashi MATSUMOTO and Kei HIRAKI
Department of Information Science, Faculty of Science, the University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo, 113, Japan
E-mail: {furukawa, tm, hiraki}@is.s.u-tokyo.ac.jp

配列の大きさを $(N_1, ...N_r)$ とする。

$\vec{a} = (1, N_1, \ldots, \prod_{k=1}^{r-1} N_k)$ とすると、求めるレイアウトは $\vec{a'} = P^{-1}\vec{a}$ となる。

最後に、各ループネスト l について内側の fully permutable なループをブロック化する。ただし、fully permutable なループ l_j のインデックスが張る空間を I_{l_j} について、ループ内の配列参照すべてにおいて $F_l i(I_{l_j}) = 0$ となるならば、l_j はブロック化せずに最内側に移動する。

4 例

以下のようなループを考える。

```
L1:DO i
     a(i,i) = ...

L2:DO i
     DO j
       a(i,j) = ...
```

ただし、L2 のループi,j は交換可能とする。

この例では、$I'_{L1} = span\{(1)\}, I'_{L2} = span\{(1,0),(0,1)\}$ であり、

$$F_{L1} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, F_{L2} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

であるから、

$$D = span\{(1,1)\}, P = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

となり、a(N,N) とすると、求めるレイアウトは $a = (1-N, N)$ となる。

5 おわりに

UMA 型共有メモリ計算機において、メモリ参照の局所性を高め、キャッシュヒット率を向上させるための配列の静的なレイアウトおよびループ変換を決定する技法について述べた。本手法は従来の手法に比べ、より一般的な結果を系統的に求めることが出来る。また、今後は [1] や [4] やといった、NUMA/NORMA 型計算機における技法との融合や、実アプリケーションでの評価を行なう予定である。

6 謝辞

執筆に当たり貴重な助言を頂いた京都大学の上原哲太郎氏に感謝いたします。

参考文献

[1] Anderson, J. M. and M. S. Lam, "Global Optimizations for Parallelism and Locality on Scalable Parallel Machines," in *Proceedings of the ACM SIGPLAN '93 Conference on Programming Language Design and Implementation*, pp. 112–125, 1993.

[2] Ju, Y.-J. and H. Dietz, "Reduction of Cache Coherence Overhead by Compiler Data Layout and Loop Transformation," in *Languages and Compilers for Parallel Computing* (U. Banerjee, D. Gelernter, A. Nicolau, and D. Padua, eds.), pp. 344–358, Springer-Verlag, 1992.

[3] Kennedy, K. and K. S. McKinley, "Optimizing for Parallelism and Data Locality," in *Proc. 1992 International Conference on Supercomputing*, pp. 323–334, 1992.

[4] Li, W. and K. Pingali, "Access Normalization: Loop Restructuring for NUMA Compilers," in *ASPLOS-V Proceedings*, pp. 285–295, 1992.

[5] Wolf, M. E. and M. S. Lam, "A Data Locality Optimizing Algorithm," in *Proceedings of the ACM SIGPLAN '91 Conference on Programming Language Design and Implementation*, pp. 30–44, 1991.

[6] Wolf, M. E. and M. S. Lam, "A Loop Transformation Theory and an Algorithm to Maximize Parallelism," *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, no. 4, pp. 452–471, October 1991.