

階層構造をもつマルチプロセッサにおける 階層データ分割分析法

李 曉傑 原田 賢一

慶應義塾大学 計算機科学専攻

横浜市港北区日吉 3-14-1

概 要

分散共有メモリ型並列計算機におけるプログラムの実行効率を向上させるための課題の1つに、データの分割と配置の問題(分割配置)がある。その結果によって、実行対象となるプログラムの並列性、通信オーバーヘッド、および負荷分散が影響される。科学技術計算のプログラムに対するデータの分割配置には、データのアクセスパターンを記述するために、一般にステンシル構造(stencil structure)が用いられ、その構造にもとづく方法が多く提案されている。

本論文は、ステンシル構造をもとに、階層メモリをもつ分散メモリ型マシンを対象として、メモリアクセスに要するプロセッサ間の通信量を定式化し、通信量を最小にするデータの分割配置法を提案するものである。具体的には、与えられたステンシル構造に対し、アクセスベクトルと呼ぶ概念を導入することによって、その構造が表すデータアクセスに必要な通信量を定式化する。その定式化に基づき、通信量が最小になるようにループの繰返し空間の最適分割を求め、分割されたデータ領域を、ループにおけるデータアクセスの性質に従って、ローカルメモリ+リモートメモリ、さらにはキャッシュメモリ+ローカルメモリ+リモートメモリの各層に配置する。最後に複数のループに対する簡易分割配置法を示す。

A Hierarchical data Partitioning Method for Hierarchical Memory Compilation

Xiaojie LI Ken'ichi HARADA

Department of Computer Science, Keio University
3-14-1, Hiyoshi, Kouhoku-ku, Yokohama, 223, Japan

Abstract

An important problem facing numerous research projects on parallelizing compilers for hierarchical memory machines is that of automatically determining a suitable data partition for a program. A stencil structure can be used for representing a data access pattern, we use stencils to evaluate the communication weight across each memory level, and develop a data partitioning scheme which generates data partitions for an extended sequentially iterated parallel loop to minimize interprocessor communication in hierarchical memory machines. Given a discretion stencil, a vector notation is chosen to quantify communication across a global data set. Communication parameters are used to generate a loop iteration partition that minimize interprocessor communication. Under the given loop partition, the data partition that maximizes local access is obtained by shifting the loop part up by several rows (lines). Furthermore, among the data part this partitioning scheme automatically identifies the *exclusive read-write data set*, and move these data into the cache to reduce the data traffic for ensuring memory coherence. For reducing conflicts among different optimal partitions, a cooperated partitioning method is proposed.

1 まえがき

ハードウェア技術の進歩により、多様なメモリシステムをもつ中規模並列マシンが商用化され、さらに大規模超並列マシンの実用化へ向けての模索が行われている。しかし、中規模並列マシンの場合でも、階層メモリをもつ並列マシンに対して、効率の良いプログラムを作成しようとする、共有メモリマシンに比べて、解決しなければならない問題がある。すなわち、1つの配列をいくつかに分けて、各プロセッサに割り当てる場合、配列の分割(partitioning)の仕方によって、プロセッサ間の通信量が増減し、さらに、メモリ階層によってアクセス速度が異なるために、分割された配列の領域をプロセッサの階層メモリにどのように配置(distribution)するかによって、プログラムの実行効率が左右される。このような状況から、データの分割配置を利用者から開放し、並列化コンパイラによって最適化することが望まれる。

一般に、プロセッサ間でのデータ転送には、プロセッサ間通信が必要となるため、CPU時間に対して大幅な時間がかかる。そこで、個々のプロセッサによって頻りにアクセスされるデータ(配列の一部に対する領域)を特定できれば、その部分をローカルメモリ、あるいはキャッシュメモリに配置することによって、プロセッサ間での通信量を減少させることができる。アクセスするデータが自分のメモリに存在しないときに限って、プロセッサ通信によって、他のプロセッサに保持されているデータを、リモートメモリとしてアクセスすればよい。

本論文は、これまでの種々の技法に基づいて、階層メモリをもつ並列マシンに適用可能なデータの最適分割配置法を提案する。この方法は、従来の研究[3][4][8][9][10][12]に比べて、次の特徴をもつ。

1. 最適なデータ分割配置は、繰返し空間に基づくデータの分割と、各プロセッサの階層メモリに対するデータ配置の2フェーズからなる。
2. 3レベルのメモリ、すなわち、キャッシュ+ローカルメモリ+リモートメモリをもつマシンに対して、最適な分割配置を与える。このほかに、ローカルメモリ+リモートメモリ、キャッシュ+共有メモリに対しても適用が可能である。
3. 通信量を評価するために、アクセスベクトルを採用する。具体的なアーキテクチャに依存しないため、多くの分散メモリマシンにこの技法を応用することができる。

以下、2章では、対象となるプログラムのモデルおよび基本的な用語について述べる。3章では、通信量を評価するために、アクセスパターンに基づく通信量の定式化を行う。4章では、階層メモリをもつ並列マシンに対するデータの最適分割配置法を提案する。5章では、複数のループに対する簡易分割配置法を考案する。6章では、研究用に開発された分散共有メモリマシンASPIREを用いた評価結果を示し、最後に、7章で結論を述べる。

2 対象モデル

本研究では、MIMD型の階層メモリをもつ並列マシンを対象とし、各プロセッサのメモリは、キャッシュ、ローカルメモリ、およびリモートメモリから構成されるものと仮定する。

分散配置の対象を2次元配列AとBとし、それらは、次に示す形のループによってアクセスされるものと仮定する。ここで、Aはデータ依存をもつ配列を、またBはデータ依存のない配列を意味する。

分割配置の目標は、並列ループの実行にあたって、各メモリ階層に配置されるデータへのアクセス時間を最小化することである。

```
DO K = 1, M
DO I = 1, N1
DO J = 1, N2
  A(I, J) = F(A(I+a1, J+b1), A(I+a2, J+b2), ...,
             A(I+aL, J+bL)) + G(B(I+c1, J+d1),
             B(I+c2, J+d2), ..., B(I+cN, J+dN))
ENDDO
ENDDO
ENDDO
```

上に示した形式のループを拡張された逐次反復並列ループ(ESIL: extended sequentially iterated parallel loop)と呼ぶ。ESILは3重ループであり、最も外側のループは逐次的に実行され、内側の2重ループは並列に実行されるものとする。最も外側のループの1回の繰返しをESILの周期(cycle)と呼ぶ。内側のループにおいて、繰返し空間(iteration space)を $N1 \times N2$ と定義する。ループ本体は、2次元配列に対する更新と参照とからなり、左辺の配列要素の更新のために、右辺で参照される配列要素の参照形式を $A(I+a_i, J+b_i)$ とする。ここで、 a_i と b_i は定数とする。関数 $F()$ と $G()$ は、代入文の右辺の式がAとBの配列要素に対する演算からなることを表す。配列要素 $A(I+a_i, J+b_i)$ について、各添字式中のループ制御変数の値に対する増分の対 (a_i, b_i) を、アクセスオフセット(access offset)と呼び、右辺に現れる配列要素に対するアクセスオフセットの集合をステンシル集合(stencil set)と呼ぶ。ループ本体に、関数 $G()$ がなければ、ESILはよく知られるSIL(sequentially iterated parallel loop)となる。SILは、曲面の平滑化[1]、偏微分方程式[6]、連続体モデリング[7]など大量の計算量を必要とする科学計算によく用いられている。また、ESILにおいて、Mが1のときは、正規形アクセスパターンをもつ2重ループになる。

分散メモリ型のマシン上での並列処理においては、データ通信のコストが、制御通信のコストよりもかなり高く、しかも分散メモリにおいて、データ通信は制御通信より頻繁に行われると考えられる。そこで、以下の解析では、簡単のため、データ通信のコストだけを考慮することにする。

本研究では、長方形によるデータ分割(以下、長方形分割と呼ぶ)を用いる。特に、正方形分割は長方形分割の特例である。

3 アクセスベクトルに基づく通信量の定式化

階層メモリをもつ並列マシン上でのデータの最適分割配置を行うには、プロセッサ間での通信量を定式化する必要がある。この章では、ステンシル集合に基づく通信量の定式化を示す。

ステンシル集合を $\{(1,2)\}$ とした場合のアクセスパターンを図3.1に示す。この図において、正方形で囲まれた部分を分割領域とすると、その領域が割り当てられたプロセッサのタスクでは、分割領域の上側と右端にある各要素の更新のために、プロセッサ間通信が必要である。その他の要素の更新は、ローカルメモリの書き換えて済む。図3.1において、データ通信による更新が必要となる要素は、分割境界を越える矢印で示してある。

通信オーバーヘッドを削減するためには、分割領域の周辺にある要素への参照コストを考慮に入れて領域を決定する必要がある。図3.1の例では、アクセスオフセット $(1,2)$ の性質から、J軸方向の要素についての通信量がI軸方向のものよりも多くなることがわかる。一般に、プロセッサ間通信量は、各分割境界における要素数とステンシル集合によって決定される。たとえば、ステンシル集合が $\{(0,5)\}$ の場合には、水平方向のオフセットが0であるため、水平方向の要素についての通信量は0となる。この場合、垂直方向をどのように分割するかによって、分割領域が決定される。

以上の考察から、ステンシル集合がプロセッサ間通信の量および方向を表現でき、分割領域の境界は、通信量が最小となるように決めればよいことがわかる。

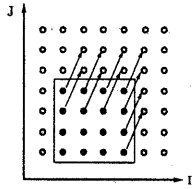


図 3.1 ステンシル集合 $\{(1,2)\}$ のアクセスパターン

最適な分割の決定は、水平および垂直方向の分割境界を決める問題と考えることができる。そのため、ESILにおける繰返しの通信量および要素の参照方向を正確に表現する必要がある。更新対象となる要素を $A(I, J)$ としたとき、右辺の式で $A(I+a, J+b)$ への参照がある場合、その関係を (a, b) で表す。この記述の形式は、アクセスオフセットと同じであるが、 (a, b) は、更新要素から参照要素へ向かう方向と参照距離を表すベクトルと考え、その場合には、それをアクセスベクトル (access vector) と呼ぶ。ステンシル集合が与えられたとき、その通信パターンは、以下に述べるように、各アクセスオフセットによって表現されるベクトルについてのベクトル分解 (vector decomposition) により、定式化することができる。なお、アクセスベクトルは、従来の依存ベクトルに似ているが、同期の意味が含まれない点において依存ベクトルとは異なる。

分割領域の各境界をセグメント (segment) と呼ぶ。長方形分割における水平および垂直セグメントを、それぞれ l_i と l_j で表す。各セグメントについての通信量は、セグメントが 1 単位の長さだけ延びると、通信量がどの程度増えるのかに着目して定式化する。まず、ステンシル集合が単一のアクセスベクトルからなる場合を考える。

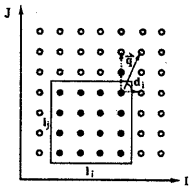


図 3.2 単一アクセスベクトルと分割領域

分割領域を P としたとき、 P の通信量 (communication weight) は、セグメント l_i (または l_j) を越えた領域外要素への参照回数として捉える。

定義 1. セグメントの単位長さ当たりの、分割領域外の要素への参照回数を、ユニット通信量 (communication weight per unit length) と呼ぶ。単位長さは、セグメント上の隣り合う 2 つの要素間の距離である。□

定理 1. アクセスベクトルを q 、セグメント l_i に対する q の角度を d_i とする。このとき、セグメント l_i についての通信量は、 $L \times q \times \sin d_i$ であり、そのときのユニット通信量は $q \times \sin d_i$ である。ここで、式中に現れるベクトルは、その長さを表すものとする。

証明： アクセスベクトル q が l_i を越えて分割領域外の要素を指すとする。それらの要素数は、横方向の長さが L 、縦方向の長さ

が $q \times \sin d_i$ の長方形内に含まれる。したがって、 q に伴う通信量は、 $L \times q \times \sin d_i$ であり、ユニット通信量は $(L \times q \times \sin d_i) / L = q \times \sin d_i$ である (図 3.2 参照)。□

アクセスベクトルを $q = (a, b)$ 、分割領域 P のセグメントを l_i と l_j とする。 l_i についての各方向のユニット通信量を、それぞれ u_i, u_j とすると、 $u_i = q \times \sin d_i = b$ 、 $u_j = q \times \sin d_j = a$ である。このとき、 P 中の要素への代入のために、単一アクセスベクトル q によって引き起こされる通信量 C_P は、(1) 式によって与えられる。

分割領域全体での通信量は、2 つのセグメントについての通信量の和として表されるが、角の部分にある要素を重複して数えるために、その部分の通信量 $u_i u_j$ を除く必要がある。

$$C_P = l_i u_i + l_j u_j - u_i u_j. \quad (1)$$

ここで、式中のセグメントはその長さを表すものとする。

次に、ステンシル集合が与えられたときの通信量を表す式を求める。以下では、ステンシル集合の各要素をベクトルとみなして議論するので、ステンシル集合の代わりにベクトル集合という呼び方をする。

ベクトル集合 $\{q_1, q_2, \dots, q_k\}$ の各ベクトル q_h を (a_h, b_h) とする。このとき、ベクトル分解によって、水平分解量の集合 $\{a_1, a_2, \dots, a_k\}$ と、垂直分解量の集合 $\{b_1, b_2, \dots, b_k\}$ とに分け、単一ベクトルの場合と同様に、通信量を定式化する。

定理 2. ベクトル集合を $S = \{q_1, q_2, \dots, q_k\}$ とし、それらの水平分解量を $\{a_1, a_2, \dots, a_k\}$ 、垂直分解量を $\{b_1, b_2, \dots, b_k\}$ とする。このとき、 S に対するユニット通信量 u_i および u_j は、それぞれ次の式で与えられる。

$$u_i = \sum_{h=1}^k |b_h| \quad u_j = \sum_{h=1}^k |a_h| \quad (2)$$

定理 2 の証明は文献 [14] を参照する。通信量は、 $C_P = l_i u_i + l_j u_j + \varepsilon$ の形をしている。 ε に相当する項は、解析時においては定数となる。この値は、領域の分割の仕方に依存しないので、以下の考察では、この定数項を除いて、 $C_P = l_i u_i + l_j u_j$ として扱う。

4 データの最適な分割配置

データを分割する場合、同じ配列要素を異なる階層のメモリに重複して配置することも考えられる。ここでは、簡単のため、まず、要素の重複がない配置という条件を設けて、ローカルメモリとリモートメモリに対する 2 レベルでの最適な分割配置を考える。そのあと、キャッシュを加えた階層のメモリに対するデータの最適配置について述べる。

4.1 ローカルメモリとリモートメモリに対する最適な分割配置

前章では、 C_P 、すなわち領域外参照要素の個数が、配列要素 $A(I, J)$ を更新するのに必要な通信コストであることを示した。 $A(I, J)$ への代入は、ループ変数 I と J によって制御されるので、セグメント l_i と l_j は、それらの変数によって構成される繰返し空間の分割 (iteration space partitioning)。以下、単に繰返し分割という) ρ による境界として決定される。一般に、並列ループの分割問題は、繰返し空間からプロセッサへのマッピングとして考えることができる。つまり、繰返し空間中の点 (i, j) をどのプロセッサに割り付けるかという問題として捉えることができる。本論文で扱う長方形

割の問題は、繰返し空間をどのようなセグメントに分割すれば通信コストがもっとも低くなるか、ということになる。その分割法は、次の定理によって与えられる。

定理 3. 与えられたステンシル集合 S についてのユニット通信量を u_i, u_j とすると、セグメント l_i と l_j が次の条件を満たすとき、通信量 C_p は最小となる。

$$\frac{l_i}{l_j} = \frac{u_j}{u_i} \quad (3)$$

証明： 分割境界を表すセグメント l_i および l_j についての通信量 C_p は、次の式で与えられる。

$$C_p = l_i u_i + l_j u_j \quad (4)$$

使用可能なプロセッサの数を、 m とするとき、各プロセッサでの繰返しの回数 n は $N1 \times N2/m$ なので、1台のプロセッサについて、次の関係が成り立つ ($N1 \times N2$ は繰返し空間の大きさを表す)。

$$l_i l_j = n \quad (5)$$

この式を用いて、(4) 式の l_j を書き換えると、関数 C_p の値を最小にする l_i の値は $\sqrt{(u_j/u_i) \times n}$ となり、この結果と (5) 式とから、(3) 式が得られる。□

ステンシル集合におけるユニット通信量とプロセッサごとの繰返しの数がわかれば、通信量を最小化する分割境界は (3) 式によって決定できる。繰返しの最適な分割は、2つのセグメントを越える通信量を総合的に考慮した場合の長方形分割である。こうして定められる分割に対しては、次に述べるように領域の配置の仕方を変えることによって、さらに通信量を減らすことができる。データ配置 σ を、配列要素 $\{1, 2, \dots, N1\} \times \{1, 2, \dots, N2\}$ からプロセッサ $\{1, 2, \dots, m\}$ へのマッピングとする。このとき、 $(i, j) \rightarrow p$ は、配列要素 $A(i, j)$ を、 p のローカルメモリに配置することを表す。

定義 2. データ配置 σ が与えられたときに、プロセッサ p が $A(i, j)$ をアクセスするのに要する時間を関数 $\lambda_{(p,\sigma)}(i, j)$ で表す。この関数を待ち時間関数 (latency function) と呼ぶ。□

定義 3. 繰返し分割 ρ に基づいて、プロセッサ p が $A(i, j)$ をアクセスする回数を、関数 $\beta_{(p,\rho)}(i, j)$ で表す。この関数を使用頻度関数 (use frequency function) と呼ぶ。□

どのプロセッサ上のタスクもメモリアクセスに関して同じ時間的特性をもつものと考え、ESILの1周期での1プロセッサ当たりの繰返しにおける配列要素への総アクセス時間を T_d とすると、 T_d は、次の式によって与えられる。

$$T_d = \sum_{(i,j) \in N1 \times N2} \lambda_{(p,\sigma)}(i, j) \beta_{(p,\rho)}(i, j) \quad (6)$$

繰返し分割 ρ は、(3) 式によって決まるものとし、(6) 式から、 T_d が最小となるようなデータ配置 σ について考える。 ρ による分割領域中の左下隅の配列要素の添字を (i_1, j_1) 、右上隅の配列要素の添字を (i_2, j_2) とする。

あるプロセッサ p が $A(i, j)$ をアクセスする回数を $F_p(i, j)$ 、 p に配置されるローカルデータの集合を $L(p)$ とする。また、ローカルメモリ中の1要素をアクセスするのに要する時間を t_l 、リモートメモリ中の1要素をアクセスするのに要する時間を t_r とすると、 $\lambda_{(p,\sigma)}(i, j)$ は、次のように定義される。

$$\lambda_{(p,\sigma)}(i, j) = \begin{cases} t_l & (i, j) \in L(p) \\ t_r & \text{otherwise} \end{cases}$$

2階層のメモリの場合、(6) 式は、次のように書き換えることができる。

$$T_d = \sum_{(i,j) \in L(p)} (F_p(i, j) t_l) + \sum_{(i,j) \notin L(p)} (F_p(i, j) t_r) \quad (7)$$

繰返し分割が決められた時点では、 $\sum_{(i,j) \in (N1 \times N2)} F_p(i, j)$ が定数になる。そこで、ローカルメモリに配置される要素の数を最大化すること、すなわち $\sum_{(i,j) \in L(p)} F_p(i, j)$ を最大化することによって、 T_d の値を最小にすることができる。

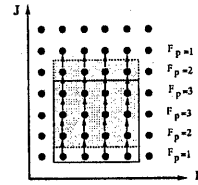


図 4.1: $\{(0, 1), (0, 2)\}$ によるデータの使用頻度

ステンシル集合が $\{(0, 1), (0, 2)\}$ の場合の各配列要素のアクセス頻度を図 4.1 に示す。上から3行目にある各要素は、2回参照されるので、 $F_p(*, 5) = 2$ であり、分割領域中の一番下にある各要素は、代入の際にアクセスされるだけなので、 $F_p(*, 1) = 1$ である。この場合には、最適な繰返し分割による領域を全体に1行だけ上に移動 (シフト) し、その部分 (図 4.1 の網かけの部分) をプロセッサ p のメモリに配置するようにすれば、通信コストをさらに減らすことができる。

上述の例に示したように、与えられたステンシル集合から、I軸とJ軸のそれぞれについて、分割領域の最適な移動量を求めることを考える。ここでは、I軸方向の移動量の決め方を示す。その際、配列要素 $A(i, j)$ の参照回数 $F_p(i, j)$ の取扱いに関しては、J軸成分を固定して、I軸成分 i だけに注目するので、 i 行目のある要素 $A(i, *)$ の参照回数を、以下 $F_p(i)$ で表す。J軸方向への移動についても、I軸の場合と同様に独立に移動量を決めることができる。

定理 4. ステンシル集合 S を $\{(a_1, b_1), (a_2, b_2), \dots, (a_k, b_k)\}$ とし、各アクセスオフセットは、 $a_1 \leq a_2 \leq \dots \leq a_k$ という関係を満たすように並べられているものとする。このとき、

$$\max_s \left(\sum_{i=i_1+s}^{i=i_2+s} F_p(i) \right)$$

を満たすデータ配置のシフト量 s は次の式によって与えられる。

$$s_{opt} = \begin{cases} a_h & \text{where } h = \frac{k+1}{2}, \quad k \text{ odd} \\ a_h & \text{where } h = \frac{k}{2}, \quad k \text{ even} \end{cases} \quad (8)$$

証明： プロセッサ p におけるローカルデータの最大アクセス回数は、次の式で与えられる。

$$\max_s \left(\sum_{i=i_1}^{i_2} F_p(i) + \sum_{i=i_2+s}^{i_2+s} F_p(i) - \sum_{i=i_1+1}^{i_1+s} F_p(i) \right)$$

s に関して、 $\sum_{i=i_1}^{i_2} F_p(i)$ は定数項なので、次の関係を満たす s を求めることを考えればよい。

$$\max_s \left(\sum_{l=1}^s (F_p(i_2+l) - F_p(i_1+l)) \right) \quad (9)$$

$|i_1 - i_2| \gg \max(|a_1|, a_k)$ とすると、 $F_p(i_2 + l) = |\{a_i | a_i \geq l\}|$ である (集合に対する記号 $|\cdot|$ は、その要素数を表すものとする)。 $l=1$ のとき、 $F_p(i_2 + l)$ は $|\{a_i | a_i \geq 1\}|$ である。 $l > a_k$ のとき、 $F_p(i_2 + l)$ は 0 になる。 $F_p(i_2 + l)$ は、 l に関する単調減関数である。

同様に、 $F_p(i_1 + l) = |\{a_i | a_i < l\}|$ が得られ、 $F_p(i_1 + l)$ は l に関する単調増関数である。したがって、 $(F_p(i_2 + l) - F_p(i_1 + l))$ は、 l に関する単調減関数になる。この性質から、それらの項の和は、非負の項だけを順に加えたものとすれば、最大となる。つまり、(9) 式の関係を満たす s は、次の関係を満たす s_{opt} とすればよい。

$$F_p(i_2 + s_{opt}) - F_p(i_1 + s_{opt}) \geq 0$$

および、

$$F_p(i_2 + s_{opt} + 1) - F_p(i_1 + s_{opt} + 1) < 0,$$

が成り立つ。これは、次の関係に対応する。

$$|\{a_i | a_i \geq s_{opt}\}| \geq |\{a_i | a_i < s_{opt}\}|$$

および、

$$|\{a_i | a_i \geq s_{opt} + 1\}| < |\{a_i | a_i < s_{opt} + 1\}|$$

s_{opt} は a_i 集合を 2 つに分けるので、 s_{opt} は (8) 式で与えられる。

□

この移動は、アクセスベクトル自身を変化させないので、繰返し分割は依然として最適な分割である。

4.2 キャッシュを考慮したデータ配置

キャッシュを含む階層メモリシステムにおいては、常に使われるデータをキャッシュにコピーすることによって、データアクセスの効率をさらに向上させることができる。キャッシュの導入には、記憶域におけるデータの一意性、すなわち同一の配列要素について、キャッシュに置かれたデータと主記憶域のデータとが一致する保証を与えるようにしなければならない。この節では、最適な繰返し分割 l_i と l_j に基づいて、キャッシュ、ローカルメモリ、およびリモートメモリに対する最適なデータ配置法を述べる。

定義 4. ステンシル集合を $S = \{q_1, q_2, \dots, q_k\}$ 、その水平分解量を $\{a_1, a_2, \dots, a_k\}$ 、垂直分解量を $\{b_1, b_2, \dots, b_k\}$ とする。このとき、 a^+ 、 b^+ 、 a^- 、および b^- を次のように定義する。

$$\begin{aligned} a^+ &= \max(\{a_h | a_h \geq 0\} \cup \{0\}), \\ b^+ &= \max(\{b_h | b_h \geq 0\} \cup \{0\}), \\ a^- &= |\min(\{a_h | a_h < 0\} \cup \{0\})|, \\ b^- &= |\min(\{b_h | b_h < 0\} \cup \{0\})|, \\ &\quad (1 \leq h \leq k) \end{aligned}$$

□

上に示した各パラメータは、配列要素 $A(l, j)$ への代入を行う際に、その右辺に現れる A 中の要素へのアクセス範囲を規定する値であり、以下アクセスパラメータと呼ぶ。

定義 5.

- 繰返し分割 ρ のもとで、プロセッサ p によって、値の読み込み (参照) が行われるデータの集合を $R(p, \rho)$ とする。
- 繰返し分割 ρ のもとで、プロセッサ p によって、書き込みが行われるデータの集合を $W(p, \rho)$ とする。 □

定義 6.

- プロセッサ p における排他的読み込み・書き込み集合 (ERW: exclusive read and exclusive write) は、次の条件を満たす要素 $A(i, j)$ の集合とする。
 $A(i, j) \in R(p, \rho) \cap W(p, \rho)$ and
 $\forall p_k \neq p (A(i, j) \notin R(p_k, \rho) \cup W(p_k, \rho))$
- プロセッサ p における共有読み込み・排他的書き込み集合 (SREW: shared read and exclusive write) は、次の条件を満たす要素 $A(i, j)$ の集合とする。
 $A(i, j) \in W(p, \rho)$ and $\forall p_k \neq p (A(i, j) \notin W(p_k, \rho))$ and
 $\exists p_k \neq p (A(i, j) \in R(p_k, \rho))$
- プロセッサ p における共用読み込み・非書き込み集合 (SRNW: shared read and no write) を、次の条件を満たす要素 $A(i, j)$ の集合とする。
 $A(i, j) \in R(p, \rho)$ and $A(i, j) \notin W(p, \rho)$ and
 $\exists p_k \neq p (A(i, j) \in W(p_k, \rho))$

□

プロセッサ p が使用しないデータは、 p で読み込みも書き込みも行われない要素の集合 (NRW: no read no write) である。ESIL におけるデータのアクセスパターンを調べれば、プログラムで用いられる配列要素は、ここに示した 4 種類の集合のいずれかに属することがわかる。以下では、コンパイル時に定まるアクセスパラメータ (a^+ , a^- , b^+ , b^-) を用いた、階層メモリへのデータの配置法を述べる。ここで、分割領域の左下隅の添字を (i_1, j_1) 、右上隅の添字を (i_2, j_2) とし、繰返し分割を $[i_1 : i_2, j_1 : j_2]$ で表す。

定理 5. プロセッサ p に対する分割領域を $\rho(p) = [i_1 : i_2, j_1 : j_2]$ とし、その分割境界を l_i と l_j とする。このとき、上で定義した各集合は、次の関係から求められる。

- ERW 集合は、 $[i_1 + a^+ : i_2 - a^-, j_1 + b^+ : j_2 - b^-]$ の要素からなる。
- SREW 集合は、 $\rho(p) - \text{ERW} = [i_1 : i_2, j_1 : j_2] - [i_1 + a^+ : i_2 - a^-, j_1 + b^+ : j_2 - b^-]$ の要素からなる。
- SRNW 集合は、 $[i_1 - a^- : i_2 + a^+, j_1 - b^- : j_2 + b^+] - [i_1 : i_2, j_1 : j_2]$ の要素からなる。
- NRW 集合は、 $N_1 \times N_2 - (\text{ERW} \cap \text{SREW} \cap \text{SRNW})$ からなる。

証明: ESIL の 1 周期において、 p における書き込み集合は $\rho(p) = [i_1 : i_2, j_1 : j_2]$ である。これは排他的書き込み集合でもある。共有読み込み・排他的書き込み集合 (SREW) は排他的書き込み集合の部分集合であり、次のように記述できる。

$$\begin{aligned} \{ (i, j) \in [i_1 : i_2, j_1 : j_2] \text{ and } (i, j) \in T_S(l, k) \\ | (l, k) \notin [i_1 : i_2, j_1 : j_2] \} \end{aligned}$$

ここで、 (l, k) は p 以外の他のプロセッサによって更新される配列要素を意味し、 $T_S(l, k)$ は、定義 2 より、 (l, k) の更新のために参照される配列要素の集合を表す。それらの集合の要素のうちで、上の条件を満たす (i, j) は、 p によって書き込みが行われるものであるから、SREW 集合は、次のように書き換えることができる。

$$\begin{aligned} \{ (i, j) \in [i_1 : i_2, j_1 : j_2] \text{ and } (i < i_1 + a^+ \text{ or } i > i_2 - a^-) \\ \text{and } (j < j_1 + b^+ \text{ or } j > j_2 - b^-) \} \end{aligned}$$

この関係を満たす要素は、 $[i_1 : i_2, j_1 : j_2]$ の中で $[i_1 + a^+ : i_2 - a^-, j_1 + b^+ : j_2 - b^-]$ を除く領域に含まれる要素であることから、SREW 集合は、次の領域の要素からなるものとして定義される。

$$[i_1 : i_2, j_1 : j_2] - [i_1 + a^+ : i_2 - a^-, j_1 + b^+ : j_2 - b^-]$$

ERW 集合は、 $ERW = \rho(p) - SREW$ の関係から、次のように定義される。

$$[i_1 + a^+ : i_2 - a^-, j_1 + b^+ : j_2 - b^-]$$

プロセッサ p での共有読み込み・非書き込み集合 (SRNW) は、次の要素からなる。

$$\{(i, j) \notin [i_1 : i_2, j_1 : j_2] \text{ and } (i, j) \in T_S(l, k) \\ | (l, k) \in [i_1 : i_2, j_1 : j_2]\}$$

ここで、 (i, j) は次の条件を満たす： $(i \leq i_1 - a^- \text{ or } i \leq i_2 + a^+) \text{ and } (j \leq j_1 - b^- \text{ or } j \leq j_2 + b^+)$ 。これは、次の式と等価である。

$$[i_1 - a^- : i_2 + a^+, j_1 - b^- : j_2 + b^+] - [i_1 : i_2, j_1 : j_2]$$

□

プロセッサ p に関して、その分割領域は、配列要素へのアクセスの性質によって、上に示した四つの部分に分けることができる。その結果を用いて、キャッシュを含む3階層のメモリに対するデータ配置を実現することができる。 t_c 、 t_l 、および t_r を、それぞれキャッシュ、ローカルメモリ、およびリモートメモリへの待ち時間とする。簡単なキャッシング技法としては、ERW をキャッシュに配置すればよい。ERW は、他のプロセッサから参照されることがないので、キャッシュとローカルメモリとの間で一貫性を保証するためのデータの入れ替えは必要がない。並列実行が始まる前に、適当なローカル配列を確保して、それをキャッシング可能と宣言し、その中に ERW を格納すればよい。SREW は、ローカルメモリに置く。このときの配列要素へのアクセス時間は、次の式で与えられる。

$$T_d = \sum_{(i,j) \in ERW} (F_p(i,j) t_c) + \sum_{(i,j) \in SREW} (F_p(i,j) t_l) \\ + \sum_{(i,j) \in SRNW} (F_p(i,j) t_r) \quad (10)$$

$[i_1 : i_2, j_1 : j_2]$ をキャッシングすることによって、さらに効率の良い配置が実現できる。その場合には、他のプロセッサからのデータ参照の要求が出されることがあるので、ESILの周期ごとに、SREWに含まれる要素をローカルメモリにコピーしておく必要がある。その書換え時間を考慮しない場合の配列要素へのアクセス時間は、次の式によって与えられる。

$$T_d = \sum_{(i,j) \in ERW \cup SREW} (F_p(i,j) t_c) + \sum_{(i,j) \in SRNW} (F_p(i,j) t_r) \quad (11)$$

上の式において、 t_r を共有メモリへの待ち時間と考えることにより、キャッシュ+共有メモリシステムに対しても、ここで述べた配置法を適用することができる。

5 複数のループに対する簡易分割配置法

これまでで、ESILを対象に、長方形によるデータ分割法を提案した。一つのループに対し、この手法は十分なものと考えられる。しかし、一般にプログラムのある部分にとって最適なデータ分割と、別の部分において最適なデータ分割と異なる場合がある。このコン

フリクトの解消をいかに行なうかが並列プログラムを効率よく実行するための鍵となる。この章では、プログラムの場所毎に最適なデータ分割が異なるといったコンフリクトを解消するための簡単な手法について述べる。

文献 [10] では、プログラム中の各ループで望まれるデータ分割に関する制約フローを求めておき、プログラム全体で実行時間を最小化するデータ分割配置を決定する手法を提案している。文献 [12] では、ループ間で新たにデータを再配置する手法を示している。ここでは、これらの考え方に基づいて、ESILに対するグローバル最適化の協調分割と再配置手法を考案する。対象とするプログラムは次の条件を満たすプログラムとする。(1) 各ループに対して前章で提案した最適分割法が適用可能である。(2) 条件分岐を考慮しない。

```
L1: DO K = 1, 50
L2: DOALL I = 1, 99
      DOALL J = 1, 99
        A1(I, J) = (A(I, J) + A(I, J+1) + A(I, J+2)
                  + A(I+1, J) + A(I+1, J+1) + A(I+1, J+2)
                  + A(I+2, J) + A(I+2, J+1) + A(I+2, J+2)) / 9
      ENDDOALL
    ENDDOALL
L3: DOALL I = 1, 99
      DOALL J = 1, 99
        A(I, J) = (A1(I, J+1) + A1(I, J+2) + A1(I, J-1)
                  + A1(I, J-2) + A1(I-1, J) + A1(I+1, J)) / 6
      ENDDOALL
    ENDDOALL
  ENDDO
```

図 5.1: 最適データ分割がコンフリクトを起こす例

画像スムージングサブルーチンから抽出されたプログラムを図 5.1 に示す。配列 A1 は一時変数として使われるので、A1 のデータ分割は A の分割に従うとする。ループ L2 と L3 では、ステンシル構造の相異から、最適な分割は異なる。配列 A に対するグローバル最適化では、次の二つの手法が考えられる。

(1) 二つの最適な分割から二つのループにとって通信コストの小さい分割を選択する方法を、ここでは協調分割と呼ぶ。任意のループ L1 と L2 について考えてみる。L1 における分割のパラメータを $u_{1i}, u_{1j}, l_{1i}, l_{1j}$ とし、L2 における分割のパラメータを $u_{2i}, u_{2j}, l_{2i}, l_{2j}$ とする。L1 の分割を基準にすると、L2 において通信の増加量 C_{21} と L2 の分割を基準にすると、L1 において通信の増加量 C_{12} は、

$$C_{21} = (l_{1i} \times u_{2i} + l_{1j} \times u_{2j}) - (l_{2i} \times u_{2i} + l_{2j} \times u_{2j}) \\ C_{12} = (l_{2i} \times u_{1i} + l_{2j} \times u_{1j}) - (l_{1i} \times u_{1i} + l_{1j} \times u_{1j})$$

である。両者間に通信のバランスをとるために、通信の増加量の小さい分割を主眼し、協調分割を決定する。すなわち、 $C_{12} < C_{21}$ であれば、 l_{2i}, l_{2j} が L1 と L2 の協調分割になる。

(2) 再配置は、ループ間にそれぞれの最適な分割に従い、データ配置を換える手法である。再配置によって、図れた実行速度の向上と配置のコストが再配置の実施を左右する。再配置を行なった場合、 C_{21} の通信量を節約することができる。一方、データの移動にともなう通信量は、

$$C_{redistribution} = (l_{1i} - l_{2i}) \times l_{1j} + (l_{2j} - l_{1j}) \times l_{2i} \quad (l_{1i} \geq l_{2i}) \\ C_{redistribution} = (l_{2i} - l_{1i}) \times l_{2j} + (l_{1j} - l_{2j}) \times l_{1i} \quad (l_{2i} \geq l_{1i})$$

である。一般に、 $C_{redistribution} < C_{21}$ であれば、再配置を行なえばよい。

N 個ループに対する協調分割の決定は、N 要素の組み合わせ問題となる。配列 A(I, J) を分割するにあたって、ループを次のように分類する。ループ変数が配列 A の添字として使われるループを分割依存ループと呼ぶ。ループ変数が配列 A の添字に使われないループを分割非依存ループと呼ぶ。

定義 7. プログラムフローグラフ (PFG: Program Flow Graph) は分割しようとする配列 A の参照関係を表したものであり、有向グラフ $G = (N, E_D, E_C)$ で表現する。節点 $n_i \in N$ は分割依存ループである。有向辺 $e_d(n_i, n_j) \in E_D$ は A に対する参照順序を表す。有向辺 $e_c(n_i, n_j) \in E_C$ は分割非依存ループ L のスコープを表し、 n_i を先頭ループ、 n_j を終端ループと呼ぶ。G に e_d を実線で、 e_c を点線で引く。 □

図 5.1 のプログラムに対応する PFG を図 5.2(a) に示す。二つの節点はループ L2, L3 を表す。有向辺 L1 は分割非依存ループ L1 を表す。すなわち、L2 と L3 が L1 にネストされることを示している。図 5.2(b) は複数のループを含んだ一般的な PFG である。L2 と L3 が L1 にネストされる場合には、再配置を行なうと、L1 の繰返し毎にデータを新たに配置しなければならないので、データの移動が頻繁に起こり、実際にプロセッサ間の通信によるオーバーヘッドが大きくなると予想される。したがって、グローバルデータ分割配置の基本方針として、分割非依存ループにネストされるループに対しては、協調分割を用い、分割非依存ループに独立したループ間では再配置を用いる。

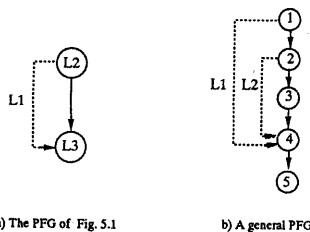


図 5.2: プログラムフローグラフ

N 個のループに対する協調分割を簡単に決定するために、次に最適分割の簡約化について述べる。三つのループ L1, L2 と L3 について考える。まず、L1 と L2 の最適分割に基づき、前に述べたように通信の増加量を比較し、一つの協調分割にまとめる。これを簡約された分割と呼ぶ。次に、この分割と L3 の最適分割に対して、同様に簡約を行ない、その結果を三つのループの協調分割とする。PFG において、分割非依存ループ L の先頭ループから終端ループまで通ったすべての節点から構成されたサブグラフを簡約可能なサブグラフと呼ぶ。簡約は各分割非依存ループ毎に行なわれる。協調分割は、複数の最適分割がコンフリクトを起こすときに、ループのアクセスパターンを考慮して、一つの分割を選択する手法である。N 個の要素の最適解を求めるのに、簡約という簡単な方法を示した。十分に満足させるものになっていないが、ループの数が少い時に、有効である。

6 分割配置法の評価

本研究では、並列マシン ASPIRE[11] を用いて、ここで提案した方法の有効性を実測評価した。ASPIRE は、10 個のプロセッサユニットからなる共有分散メモリ MIMD マシンである。プロセッサには、TMP68301 が用いられ、各プロセッサユニットは 4KB のデュアルポートメモリ (DPM: dual port memory) と 1MB のローカルメモリ (SDM: shared-distributed memory) をもつ。プ

ロセッサユニットは VME バスで結合され、VME バスは 16MHz で動作する。ASPIRE は非一様メモリ参照 (NUMA: non-uniform memory access) マシンである。各プロセッサユニットにおけるメモリ階層は、そのプロセッサの DPM、ローカルメモリ SDM、リモートメモリ (他のプロセッサの SDM) からなる。各メモリ階層において、DPM のアクセス待ち時間は $0.4\mu s$ (6 クロック)、SDM のアクセス待ち時間は $0.6\mu s$ (10 クロック)、リモート SDM へのアクセス待ち時間は $2.1\mu s$ (34 クロック) である。

評価には、文献 [7] より画像の平滑化プログラムをモデル化したものを用いた。その一部を図 6.1 に示す。

```
DO K = 1, 15
DOALL I = 1, N
DOALL J = 1, N
A1(I, J) = (A(I+2, J)+A(I+1, J)+A(I-2, J)
+A(I-1, J)+A(I, J+2)+A(I, J-2))/6
ENDDOALL
ENDDOALL
配列 A1 を配列 A にコピー
ENDDO
```

図 6.1: テストプログラム (一部)

なお、この実験では、データの依存関係を簡単にするために、中間的な配列 A1 を導入し ESIL の 1 周期が終わるたびに、A1 をもとの配列 A にコピーするようにしている。A1 と A については、同じデータ分割配置を用いた。評価用コードのステンシル集合は $\{(2, 0), (1, 0), (-2, 0), (-1, 0), (0, 2), (0, -2)\}$ なので、定理 2 により、 $u_i = 4$ 、 $u_j = 6$ となり、 $l_i/l_j = 1.5$ を満たす繰返し分割が最適となる。定理 4 により、分割領域の移動量は、 $s_i = s_j = 0$ である。

ASPIRE にキャッシュはないので、ローカルメモリ+リモートメモリを対象とし、分割配置の効果を C プログラムの実行時間によって測定した。キャッシュを含む階層的メモリに対する本方法の効果については、疑似的な実験によって評価を行ったので、最後にその結果を示す。

ASPIRE の共有分散メモリで 2 組の実験を行った。まず、本論文で提案した方法と、静的な正方形分割法、および動的な GSS (guided self-scheduling method) [2] によるプログラムの実行時間の比較を行った。GSS の場合には、繰返し分割がコンパイル時に定まらないので、データをプロセッサ p_1 と p_2 に属すローカルメモリに連続的に配置した。N = 120 の場合の実行時間の比較を図 6.2 に示す。本論文で述べた分割によるプログラムの実行時間を 1 として、正方形分割および GSS による実行時間を比較すると、次のようになる。本技法は、正方形分割よりも、実行時間が 8.5% 短く、GSS よりも 13.9% 短くなっている。本分割配置法において、 l_i/l_j の値を変化させたときの実行時間の変化を図 6.3 に示す。この図から明らかなように、実験に用いたプログラムの場合、 $l_i/l_j = 1.5$ のときに、実行時間が最短となり、ここで述べた分割配置法が、実際にも効果のあることがわかる。

ASPIRE の各プロセッサユニットにある DPM は、2 つのポートを用いてデータをアクセスすることができ、一般にローカルメモリのアクセスよりも速い。そこで、キャッシュの代わりとして DPM を用いた、3 レベル階層メモリへのデータ分割配置の実験の結果を示す。実際のキャッシュにおけるデータの更新は、ハードウェアによって行われるが、DPM の場合はソフトウェアによって実現しなければならない。その更新に要する時間差をなくすために、実験では、ERW だけを DPM に、SREW と SRNW をローカルメモリに配置するようにした。図 6.1 に示したプログラムを用いて、配列の大きさ N を、それぞれ 30, 60, 90 とし、DPM を利用した場合と利用しない場合とについて、各実行時間を測定した。その

結果を図 6.4 に示す。この結果から、実際のキャッシュを利用して、4.2 で述べた配置を行えば、実行の効率が大幅に改善できることが期待できる。

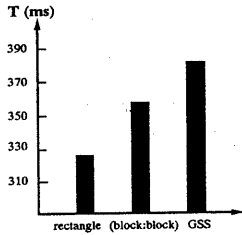


図 6.2: 典型的な分割法との比較

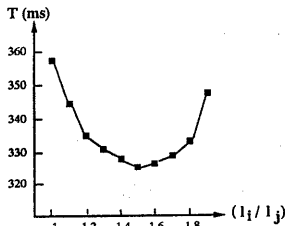


図 6.3: 分割境界を変化させたときの実行時間

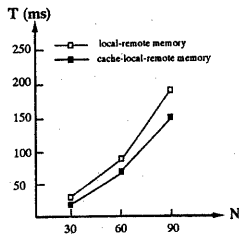


図 6.4: 異なるメモリシステムを用いた実行時間の比較

7 むすび

分散メモリ型の計算機では、ハードウェアのピーク性能に対して実際のプログラムで達成できる性能が低いことが問題として指摘されている。これは、メモリアクセスの非一様性が並列処理に大きな影響を与えるからである。そこで、より簡単に、効率の良い並列プログラムを開発するための手段として、並列コンパイラにおけるデータの自動分割配置が望まれる。コンパイル時にデータアクセスの非一様性を利用することによって、データの自動分割配置が可能である。本論文では、プロセッサ間の通信量を最小化し、さらに階層メモリの特徴を活かしたデータの最適分割配置法を提案した。その通信量の定式化に当たっては、アクセスベクトルと呼ぶ概念を用い、通信量を最小にするための、最適な繰返し分割境界を求める方法を示した。さらに、ERWをキャッシュに、SREWをローカルメモリに、またSRNWをリモートメモリに配置することによって、ESILの並列実行の効率向上が達成できることを示した。

階層メモリをもつ計算機をターゲットマシンとするコンパイラに、この最適分割配置を組み込むことによって、従来の言語を用い

て書かれたプログラムでも効率良く実行できるようになる。本論文に示した論理的アプローチは、分散メモリ型のマシン一般に利用可能である。また、この解析法は、マルチプロセッサでのプログラミングにとって、実行効率を向上させるための一般的な手法としても有用である。

参考文献

- [1] B. Schunk, "Image Flow Segmentation and Estimation by Constraint Line Clustering," *IEEE Trans. Pattern Anal. Machine Intell.*, Vol.11(Oct. 1989), pp.1010-1027.
- [2] C.D. Polychronopoulos and D. Kuck, "Guided Self-Scheduling: A Practical Scheduling Scheme for Parallel Supercomputers," *IEEE Trans. Computers*, Vol.C-36, No.12(Dec. 1987), pp.1425-1439.
- [3] C. Koelbel and P. Mehotra, "Programming Data Parallel Algorithms on Distributed Memory Machines Using Kali," in *Proc. 1991 ACM Int. Conf. Supercomputing* (July 1991), pp.414-423.
- [4] D. Reed, L. Adams, and M. Patrick, "Stencils and Problem Partitionings: Their Influence on the Performance of Multiple Processor Systems," *IEEE Trans. Computers*, Vol. C-36(July 1987), pp.845-858.
- [5] D. Socha, "An Approach to Compiling Single-Point Iterative Programs for Distributed Memory Computers," in *Proc. Fifth Distributed Memory Comput. Conf.*, Charleston, SC, April 1990.
- [6] G. Fox and S. Otto, "Algorithms for Concurrent Processors," *Physics Today*, Vol.37(May 1984), pp.50-59.
- [7] H. Stone, *High Performance Computer Architecture*, Reading, MA: Addison-wesley, 1987.
- [8] H. Zima, H. Bast, and M. Gerndt, "SUPERB: A Tool for Semi-Automatic MIMD/SIMD parallelization," *Parallel Comput.*, Vol.6(1988), pp.1-18.
- [9] J. Li and M. Chen, "Index Domain Alignment: Minimizing Cost of Cross-Referencing between Distributed Arrays," in *Frontiers90: 3rd Symp. Frontiers Massively Parallel Computat.* College Park, MD, Oct. 1990.
- [10] M. Gupta and P. Banerjee, "Demonstration of Automatic Data Partitioning Techniques for Parallelizing Compilers on Multi-computers," *IEEE trans. Parallel and Distributed System*, Vol.3, No.2(March 1992), pp.179-193.
- [11] N. Yamasaki and Y. Anzai, "The Design and Implementation of the Personal Robot Hardware Architecture ASPIRE," *Proceedings of the 48th Annual Convention IPS Japan*, pp. 6-91, Mar. 1994.
- [12] S. Hiranandani, K. Kennedy, and C. Tseng, "Evaluation of Compiler Optimizations for Fortran D on MIMD Distributed-Memory Machines," in *Proc. 1992 ACM Int. Conf. Supercomputing*(July 1992), pp.1-14.
- [13] V. Balasundaram, G. Fox, K. Kennedy, and U. Kremer, "An Interactive Environment for Data Partitioning and Distribution," in *Proc. Fifth Distributed Memory Comput. Conf.*, Charleston, SC, April 1990.
- [14] X. Li and K. Harada, "Mapping a Global Array to Distributed-shared Memory Systems Based on Access Patterns," *IPSI SIG Notes*, Vol.94, No.86(1994), pp.1-8.
- [15] X. Li and K. Harada, "An Optimal Mapping of a Global Array to Hierarchical Memory Systems with Three Levels," To appear in *International Symposium on Parallel Architectures, Algorithms and Networks*, 1994.