システムソフトウェアと
オペレーティング・システム 66－1
（１９９４．９．１６）

アクセス パターンを考慮した分散共有メモリ上のデータマッピング手法

李 暁傑　　原田 賢一

慶應義塾大学 理工学部
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、階層メモリをつも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。

Mapping a Global Array to Distributed Memory Systems Based on Access Patterns

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 write data set*, and move these data into the cache to reduce the data traffic for ensuring memory coherence.

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と、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割して、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 時間に比べ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繁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 [14] [9] [3] [4] [6] [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、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、6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+a_1, J+b_1), A(I+a_2, J+b_2), ... ,
                  A(I+a_l, J+b_l)) + G(B(I+c_1, J+d_1),
                  B(I+c_2, J+d_2), ... , B(I+c_h, J+d_h))

    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×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 [7]、連続体モデリング [8] 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。

プロセッサ間通信を、文献 [2] の定義に従いって、次のように分類する。

(1) **データ通信**： 計算に応じて、プロセッサ間で必要なデータを転送するための通信。

(2) **制御通信**： プロセッサ同士が協調に作業する際に、同期をとるための通信。

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。

ESIL を対象としたデータ分割の形状を決定するためには、配列要素 A(I, J) の更新に伴う同一配列への参照パターンを考慮する必要がある。配列 A に関して、参照される要素が、更新対象となる要素 A(I, J) の周辺にある場合、更新と参照とはある範囲に限定される。このようなデータ アクセスの局所性を活かした領域分割には、**幾何分割** (geometric partition) が最適である。

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 メモリの書き換えで済む。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。

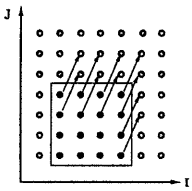

図 3.1 ステンシル集合 $\{(1,2)\}$ のアクセス パターン

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。

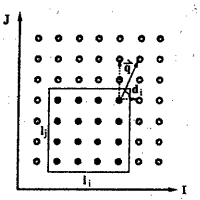

図 3.2 単一アクセス ベクトルと分割領域

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 $(0,5)$ としたとき、q の長さ、d_i, および d_j は、それぞれ 5、90°、0° である。したがって、l_i についてのユニット通信量は $5 \times \sin 90° = 5$、l_j についてのユニット通信量は $5 \times \sin 0° = 0$ である。各アクセス ベクトルに対する通信量は、ベクトルの分解により、各方向のユニット通信量として表す。

アクセス ベクトルを q、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、$u_i = q \times \sin\ d_i$、$u_j = q \times \sin\ d_j$ 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。

$$C_P = l_i u_i + l_j u_j - u_i u_j. \tag{1}$$

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 $\{q_1, q_2, \ldots, q_k\}$ 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 $\{a_1, a_2, \ldots, a_k\}$ と、垂直分解量の集合 $\{b_1, b_2, \ldots, b_k\}$ 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。

定義 2. A をグローバル配列とし、そのベクトル集合 S を $\{(a_1,b_1),(a_2,b_2),\ldots,(a_k,b_k)\}$ とする。このとき、A の配列要素の添字の対を $y=(a,b)$ とすると、$\{(a+a_1,b+b_1),(a+a_2,b+b_2),\ldots,(a+a_k,b+b_k)\}$ を S による y の**参照要素の集合** と呼び、$T_S(y)$ で表す。 □

定義 3. 与えられた分割領域 P と、ベクトル集合 S について、P 内の要素を更新するために参照される要素のうちで、P の外側にあるものを、P の**領域外参照要素の集合**という。S についての P の領域外参照要素の集合 P_S は、次のように定義される。

$$P_S = \{\, x \mid y \in P \text{ and } x \in T_S(y) \text{ and } x \notin P \,\}$$

□

一般に、ベクトル集合が複数のベクトルからなる場合には、同一の配列要素が何回か参照される可能性がある。たとえば、ベクトル集合を $\{(1,3),(1,2)\}$ とすると、A(I, J) を更新するために、A(I+1, J+3) が参照され、別の繰返しで、P 内 にある要素 A(I, J+1) が更新されるときに、A(I+1, (J+1)+2)、すなわち A(I+1, J+3) が再び参照される。以下の解析では、同じ要素への参照が何回あっても、それらは独立に数えるものとする。つまり、各アクセスベクトルによって、独立にデータ通信が引き起こされるものとして扱う。

定理 2. ベクトル集合を $S = \{q_1, q_2, \ldots, q_k\}$ とし、それらの水平分解量を $\{a_1, a_2, \ldots, a_k\}$、垂直分解量を $\{b_1, b_2, \ldots, b_k\}$ 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。

$$u_i = \sum_{h=1}^{k} |b_h| \qquad u_j = \sum_{h=1}^{k} |a_h| \tag{2}$$

証明： 任意の $q_h \in \{q_1, q_2, \ldots, q_k\}$ について、セグメント l_i と l_j とからなる分割領域の領域外参照要素の個数を $|P_{q_h}|$ とすると、次の関係が成り立つ。ベクトル分解量は符号をもつために、(1) 式を用いるに当たって、その絶対値をとる必要がある。

$$|P_{q_h}| = l_i|b_h| + l_j|a_h| - |a_h||b_h|$$

すべてのベクトルの参照による通信量は、各要素についての通信量の和として、次の式で与えられる。

$$\sum_{h=1}^{k} |P_{q_h}| = l_i \sum_{h=1}^{k} |b_h| + l_j \sum_{h=1}^{k} |a_h| - \sum_{h=1}^{k} |a_h||b_h|$$

(1) 式より、セグメント l_i および l_j に対するユニット通信量として、それぞれ $u_i = \sum_{h=1}^{k} |b_h|$、$u_j = \sum_{h=1}^{k} |a_h|$ が得られる。 □

単一ベクトルでも複数ベクトルの場合でも、通信量は、いずれも $C_P = l_i u_i + l_j u_j + \varepsilon$ 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。次の章では、ここに示した通信量の定式化に基づく最適な長方形分割について述べる。

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、キャシュを加えた階層的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 partioning、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分割の問題は、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。

$$\frac{l_i}{l_j} = \frac{u_j}{u_i} \tag{3}$$

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。

$$C_P = l_i u_i + l_j u_j \tag{4}$$

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 $\mathrm{N1} \times \mathrm{N2}/m$ 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 × N2 は繰返し空間の大きさを表す)。

$$l_i\, l_j = n \tag{5}$$

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。とくに、u_i が 0 のとき、最適な分割は、$l_i = \mathrm{N1}$、$l_j = \mathrm{N2}/m$ となる。u_j が 0 のときも、同様である。また、$l_i = l_j$ のときは、正方形分割になる。繰返しが各プロセッサに均等に与えられる場合 (ただし、周辺の部分を除く)、分割領域のセグメントは、次の式によって与えられる。

$$l_i = \sqrt{\frac{u_j}{u_i} \times \frac{\mathrm{N1} \times \mathrm{N2}}{m}} \qquad l_j = \sqrt{\frac{u_i}{u_j} \times \frac{\mathrm{N1} \times \mathrm{N2}}{m}} \tag{6}$$

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 $\{1,2,\ldots \mathrm{N1}\} \times \{1,2,\ldots,\mathrm{N2}\}$ からプロセッサ $\{1,2,\ldots,m\}$ へのマッピングとする。このとき、$(i,j) \to p$ は、配列要素 $\mathrm{A}(i,j)$ を、p のローカル メモリに配置することを表す。

定義 4. データ配置 σ が与えられたときに、プロセッサ p が $\mathrm{A}(i,j)$ 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 5. 繰返し分割 ρ に基づいて、プロセッサ p が $\mathrm{A}(i,j)$ 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。

$$T_d = \sum_{(i,j)\in \mathrm{N1}\times \mathrm{N2}} \lambda_{(p,\sigma)}(i,j)\beta_{(p,\rho)}(i,j) \qquad (7)$$

繰返し分割 ρ は、(3) 式によって決まるものとし、(7) 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 $\mathrm{A}(i,j)$ 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。

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

2 階層のメモリの場合、(7) 式は、次のように書き換えることができる。

$$T_d = \sum_{(i,j)\in L(p)} (F_p(i,j)\ t_l) + \sum_{(i,j)\notin L(p)} (F_p(i,j)\ t_r) \qquad (8)$$

繰返し分割が決められた時点では、$\sum_{(i,j)\in(\mathrm{N1}\times\mathrm{N2})} F_p(i,j)$ 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。

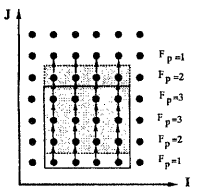

図 4.1: $\{(0,1),(0,2)\}$ によるデータの使用頻度

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 $\mathrm{A}(i,j)$ 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 $\mathrm{A}(i,*)$ 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 $\{(a_1,b_1),(a_2,b_2),\ldots,(a_k,b_k)\}$ とし、各アクセス オフセットは、$a_1 \le a_2 \le \ldots \le a_k$ 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。

$$s_{opt} = \begin{cases} a_h & where\ h = \frac{k+1}{2},\ \ k\ odd \\ a_h & where\ h = \frac{k}{2},\ \ k\ even \end{cases} \qquad (9)$$

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。

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

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。

$$\max_s \left(\sum_{l=1}^{s} (F_p(i_2+l) - F_p(i_1+l)) \right) \qquad (10)$$

$|\ i_1 - i_2\ | \gg \max(|\ a_1\ |,\ a_k)$ とすると、$F_p(i_2+l) = |\ \{\ a_i\ |\ a_i \ge l\ \}\ |$ である (集合に対する記号 $|\ |$ 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 $|\ \{\ a_i\ |\ a_i \ge 1\ \}\ |$ 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、(10) 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。

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

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。

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

および、

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

s_{opt} は a_i 集合を 2 つに分けるので、s_{opt} は (9) 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ので、繰返し分割は依然として最適な分割である。任意の ESIL において、要素の参照しか行われない配列が用いられた場合でも、その配列には、ここに示したのと同様の方法によって、最適なデータ分割配置を行うことができる。

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貫性、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。前節で述べたように、階層的なメモリにおける最適な分割配置は繰返し分割およびデータ配置によって求められる。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 6. ステンシル集合を $S = \{q_1, q_2, \ldots, q_k\}$、その水平分解量を $\{a_1, a_2, \ldots, a_k\}$、垂直分解量を $\{b_1, b_2, \ldots, b_k\}$ とする。このとき、a^+、b^+、a^-、および b^- を次のように定義する。

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

□

上に示した各パラメタは、配列要素 A(I,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 パラメタ**と呼ぶ。

定義 7.

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

□

定義 8.

- プロセッサ p における排他的読込み・書込み集合 (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 における共有読込み・排他的書込み集合 (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 における共用読込み・非書込み集合 (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 で読込みも書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。

1. ERW 集合は、$[\, i_1 + a^+ : i_2 - a^-, j_1 + b^+ : j_2 - b^- \,]$ の要素からなる。
2. SREW 集合は、$\rho(p)$− 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。
3. 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。
4. NRW 集合は、N1 × N2 − (ERW ⋂ SREW ⋂ SRNW) からなる。

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

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

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。

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

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 での共有読込み・非書込み集合 (SRNW) は、次の要素からなる。

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

ここで、(i,j) は次の条件を満たす：$(i \leq i_1 - a^- \textbf{ or } i \leq i_2 + a^+)$ **and** $(j \leq j_1 - b^- \textbf{ or } j \leq j_2 + b^+)$。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。$[\, i_1 : i_2, j_1 : j_2 \,]$ において、d_c をキャッシュに配置するデータの集合とする。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。この場合、プロセッサ p での繰返しにおける配列要素への総アクセス時間を表す式 (7) は、次のように書き換えることができる。

$$T_d = \sum_{(i,j) \in d_c} (F_p(i,j)\, t_c) + \sum_{(i,j) \in L(p) - d_c} (F_p(i,j)\, t_l)$$

$$+ \sum_{(i,j)\notin L(p)} (F_p(i,j)\, t_r) \qquad (11)$$

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換えは必要がない。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。

$$T_d = \sum_{(i,j)\in \mathrm{ERW}} (F_p(i,j)\, t_c) + \sum_{(i,j)\in \mathrm{SREW}} (F_p(i,j)\, t_l)$$

$$+ \sum_{(i,j)\in \mathrm{SRNW}} (F_p(i,j)\, t_r) \qquad (12)$$

[$i_1 : i_2,\ j_1 : j_2$] 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、他のプロセッサさ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。

$$T_d = \sum_{(i,j)\in \mathrm{ERW}\cup \mathrm{SREW}} (F_p(i,j)\, t_c) + \sum_{(i,j)\in \mathrm{SRNW}} (F_p(i,j)\, t_r) \quad (13)$$

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 分割配置法の評価

本研究では、並列マシン ASPIRE[13] 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 のアーキテクチャを図 5.1 に示す。

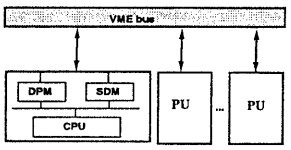

図 5.1: ASPIRE のアーキテクチャ

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 0.4μs(6 クロック)、SDM のアクセス待ち時間は 0.6 μs(10 クロック)、リモート SDM へのアクセス待ち時間は 2.1 μs(34 クロック) である。

評価には、文献 [8] 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 5.2 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))/6.0
 ENNDOALL
 ENDDOALL
 配列 A1 を配列 A にコピー
ENDDO
```

図 5.2: 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 5.3 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。

本分割配置法 : 正方形分割 : GSS = 1 : 1.093 : 1.162

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 5.4 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 5.2 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 5.5 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。

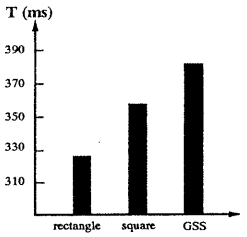

図 5.3: 典型的な分割法との比較

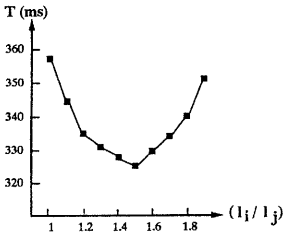

図 5.4: 分割境界を変化させたときの実行時間

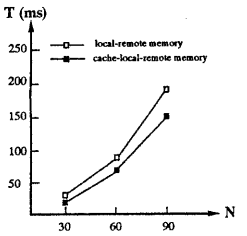

図 5.5: 異なるメモリ システムを用いた実行時間の比較

6 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ケット 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、マルチプロセッサ上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 Compt. Conf., Charleston, SC, April 1990.

[6] F. Irigoin and R. Triolet, *"Supernode Partition,"* in Proc. 15th Annu. ACM SIGACTSIGPLAN Symp. Principles of Programming Languages(1988), pp.319-329.

[7] G. Fox and S. Otto, *"Algorithms for Concurrent Processors,"* Physics Today, Vol.37(May 1984), pp.50-59.

[8] H. Stone, High Performance Computer Architecture, Reading, MA: Addison-wesley, 1987.

[9] H. Zima, H. Bast, and M. Gerndt, *" SUPERB : A Tool for Semi-Automatic MIMD/SIMD parallelization,"* Parallel Comput., Vol.6(1988), pp.1-18.

[10] 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.

[11] J. Ramanujan and P. Sadayappan, *"A Methodology for Parallelizing Programs for Multicomputers and Complex Memory Multiprocessors,"* in Proc. Supercomput.'89(Nov. 1989), Reno, NV, pp.637-646.

[12] M. Gupta and P. Banerjee, *"Demonstration of Automatic Data Partitioning Techniques for Parallelizing Compilers on Multicomputers,"* IEEE trans. Parallel and Distributed System, Vol.3, No.2(March 1992), pp.179-193.

[13] N. Yamasaki and Y. Anzai, *"The Design and Implementation of the Personal Robot Hardware Architecture ASPIRE,"*

[14] 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.

[15] 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.