ツイステッドデータレイアウト

進藤達也† 岩下英俊† 土肥実久†
萩原純一† 金城ショーン†

本論文では，分散メモリ型並列計算機のための新しいデータレイアウト法としてTwisted data layoutを提案する．データレイアウトは分散メモリ型並列計算機で並列プログラムを効果的に実行させる上での重要な要素である．配列データの最適なデータレイアウトパターンは，プログラム全体を通して一つに決まらず，プログラムの部分ごとに異なる場合がある．Twisted data layoutは，最適なデータ分散法に関するこのようなコンフリクトの解消に用いることができる．並列計算機AP1000を用いた実験で，Twisted data layoutの性能評価を行う．

Twisted Data Layout

TATSUYA SHINDO,† HIDETOSHI IWASHITA,† TSUNEHISA DOI,†
JUNICHI HAGIWARA † and SHAUN KANESHIRO †

This paper proposes *twisted data layout* as a novel and efficient data layout technique for distributed memory parallel processors (DMPP). Data layout is an important aspect in efficiently executing a parallel program on DMPPs. The optimal data layout pattern for an array may differ throughout the program. *Twisted data layout* can be used to resolve the conflicts among the optimal array distributions. Experimental results on the AP1000 multicomputer measure the performance of the *twisted data layout* scheme.

1. はじめに

データ分散法の決定は，分散メモリ型並列計算機のプログラミングをする上で重要な課題である．なぜならば，データ分散法はプログラムから抽出できる並列性，通信オーバヘッド，負荷分散に大きく影響を与えるからである．

これまでに分散メモリ型並列計算機を対象に，ブロックあるいはサイクリックによるデータ分散を表現できる言語が提案されている[1)~7)]．また，近年，ブロックあるいはサイクリックによるデータ分散に基づいてデータレイアウトを自動で決定する手法も提案されている[8),9)]．ブロックとサイクリックによって表現するデータ分散法はシンプルであり，最適なデータ分散パターンが一つに決定できるようなプログラムにとっては充分なものである．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レイアウトと，同一のプログラム内の別の部分において最適なデータレイアウトとが異なる場合がある．

本論文では，プログラムの場所ごとに最適なデータレイアウトが異なるといったコンフリクトを解消するための手法としてtwisted data layoutを提案する．さらに，twisted data layoutに基づくコードジェネレーション法を紹介し，分散メモリ型並列計算機AP 1000[16)]を用いた実験結果を示す．

以下，2章では，本研究の背景とtwisted data layoutの概要を紹介する．3章では，twisted data layoutの定義を行う．4章では，twisted data layoutを実現するためのコードジェネレーション法を提案する．コードジェネレーションは，データアロケーション，添字変換，計算マッピングから構成される．5章と6章では，AP 1000を用いた実験結果を示し，twisted data layoutを他のデータレイアウト法と比較し議論する．7章では関連する研究について述べる．

2. 対象とする問題

2.1 2フェーズデータレイアウト

データレイアウトとは，並列プログラムを効率良く実行させることを目的に，データのアラインメントとプロセッサへの分散を決定することである．一つのプログラム内で，ある一つのループにとって最適なデータレイアウトが他のループに対するデータレイアウト

† 富士通(株)
Fujitsu Ltd.

```
1           DO 100 I = 0, N-1
2              DOALL 100 J = 0, N-1
3                 A(I,J) = ...
4     100   CONTINUE
5
6           DOALL 200 I = 0, N-1
7              DO 200 J = 0, N-1
8                 ... = f(A(I,J))
9     200   CONTINUE
```

List 1 最適データ分散がコンフリクトを起こす例
List 1 An example of conflicting optimal distributions.

とコンフリクトを起こすことがあり得る．この問題を扱うために，我々はデータレイアウトをローカルデータレイアウトとグローバル最適化の2フェーズで決定する．まず，ループネストの並列性を最大限に引き出すために，ローカルなデータレイアウトを個々のループネストごとに決定する．次に，ループネスト間での最適データレイアウトのコンフリクトを解消するために，グローバル最適化を試みる．このコンフリクトの解消をいかに行うかが，分散メモリ型並列計算機上で並列プログラムを効率良く実行するための鍵となる．

2.2 問題例

List 1 に例を示す．このプログラムで最初のループネスト (100) にとっては，内側のループの並列性を活かすために配列Aの2次元目を分散させるレイアウトが最適である．しかしながら，二番目のループネスト (200) にとっては，外側のループの並列性を活かすために配列Aの1次元目を分散させるレイアウトが最適である．

この例では二つのループネストの最適なデータレイアウトが互いに異なっているため，一方のループネストにとって最適な分散を両方に適用してしまうと，もう一方の性能を損なう可能性がある．このようなコンフリクトを解消するアプローチの一つは配列Aを2次元格子状のプロセッサ上に1次元目と2次元目の両方の次元で分散する方法である[9]．しかし，例のような問題に対してこのようなアプローチでは，一つの行あるいは一つの列上のプロセッサのみが動作を行い，他のプロセッサはアイドル状態になってしまう．別の解消法としては，実行時にデータレイアウトの変更を行うデータの再分散[1),10)]がある．データの再分散では，データ移動に伴う通信のオーバヘッドから，粒度の大きな計算を含んだプログラムに対してのみ実用的である．次の節では，3番目のアプローチとしてtwisted data layoutを考える．

2.3 Twisted Data Layoutによるコンフリクトの解消

データレイアウトのコンフリクトが，データ分散の対象となる次元が異なることから発生する場合に，twisted data layoutを用いることができる．例のプログラム ($N=4$ の場合) に対してtwisted data layoutを適用したデータレイアウトを**図1**に示す．このレイアウトはどちらのループネストも並列実行を可能にし，さらにすべてのプロセッサをその計算処理に参加させることを可能とする．例えば，配列要素A(0, 0), A(1, 0), A(2, 0), A(3, 0)はそれぞれ異なるプロセッサにマッピングされており，ループネスト100によって並列処理することができる．同様に，ループネスト200において，A(0, 0), A(0, 1), A(0, 2), A(0, 3)がそれぞれ異なるプロセッサにマッピングされていることから並列処理することができる．

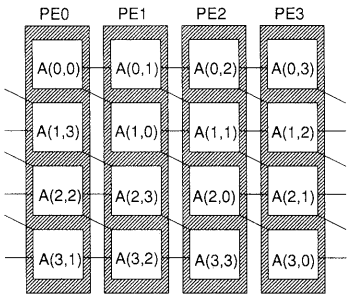


図1 4×4配列を1次元4プロセッサ構成のマシン上に，twisted data layoutを用いてマッピングする例
Fig. 1 An example of twisted data layout for a 4×4 array on a 4 processor configuration.

3. Twisted Data Layoutの定義

Twisted data layoutは，従来型のブロックあるいはサイクリックによる分散法[1)~7)]の拡張として定義される．Twisted data layoutを適用する前に，配列やテンプレートはまず，HPFのデータ分散モデル[1)]と同様にバーチャルプロセッサ上にマッピングされる．ここで，バーチャルプロセッサの次元数は，配列やテンプレートの分散対象となる次元数と等しくする．最後に，バーチャルプロセッサはtwisted data layoutによって，1次元の物理プロセッサ上にマッピングされる．HPFを拡張した表記として，以下のようにtwisted data layoutを宣言することができる．プロセッサPの添字((1, 2))が拡張表現であり，配列の1次元目と2次元目をtwistして，Pの1次元目にマップすることを表している．

```
!HPF$ PROCESSOR P(4)
      REAL A(8,8, 8), B(8, 8)
!HPF$ DISTRIBUTE A(BLOCK, BLOCK)   ONTO P((1, 2))
!HPF$ DISTRIBUTE B(CYCLIC, CYCLIC) ONTO P((1,2))
```

ここで宣言されている配列 A, B の分散を図2に示す．8×8 の配列は最初に，4×4 の 2 次元バーチャルプロセッサにブロックあるいはサイクリックで分散される．次に，twisted data layout で，4 台の 1 次元物理プロセッサ上にマッピングする．

定義：

m をバーチャルプロセッサの次元数，l_k をバーチャルプロセッサの k 次元目のサイズとする．n を物理プロセッサ数とする．V をバーチャルプロセッサ集合とし，P を物理プロセッサ集合とする．

$$V=\{\vec{v}=(v_1,\cdots,v_m)|1\leq k\leq m,\ 0\leq u_k<l_k\}$$

$$P=\{p|0\leq p<n\}$$

Twisted data layout T は，バーチャルプロセッサ $\vec{v}$ とそれをマッピングする物理プロセッサ p のペアの集合として表せる．

$$T=\{(\vec{v},p)|\vec{v}\in V,\ p\in P,\ \left(\sum_{k=1}^{m}u_k\right)\bmod n=p\}$$

4. コードジェネレーション

本章では，twisted data layout を実現するためのコードジェネレーション法としてデータアロケーション，添字変換，計算マッピングについて述べる．なお，以下の例では 2 次元配列を用いるが，3 章の定義より 3 次元以上の配列に対しても同様に適用可能である．

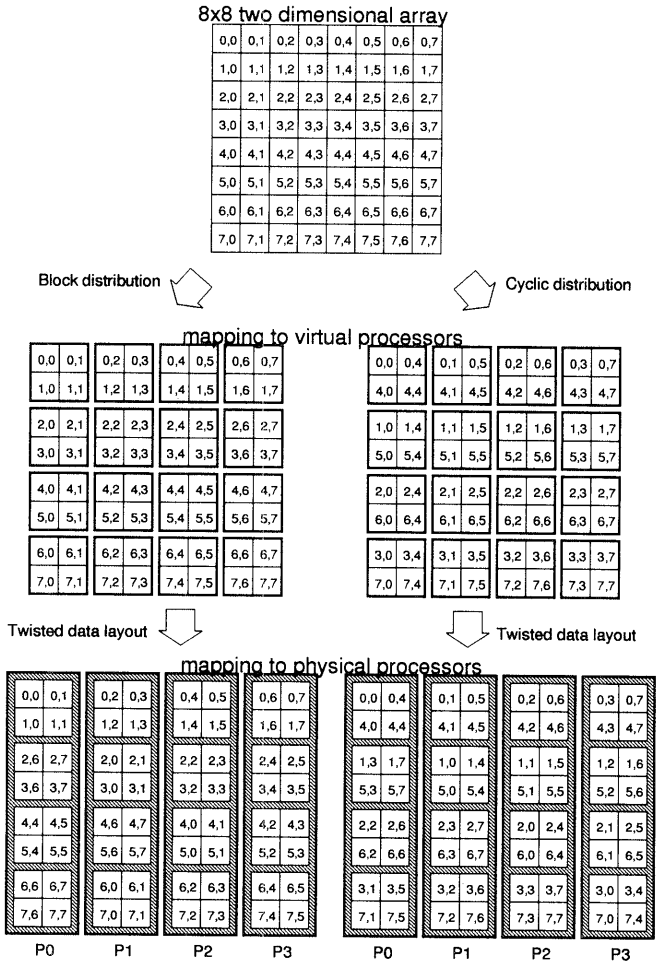


図2 ブロックあるいはサイクリックな分散をもった twisted data layout
Fig. 2 Twisted data layout with block or cyclic distribution.

4.1 データアロケーション

本節では，twisted data layout によって分割されたデータの領域を各プロセッサ内でどのようにアロケーションするかを述べる．配列 A ($D_1, \cdots, D_N$) を次元 i のサイズが D_i である N 次元配列とする．配列 A はまず，各次元のサイズが n の m 次元バーチャルプロセッサに分割される．各バーチャルプロセッサには，以下のサイズの配列 A′を割り当てる．

$$A'(M_1, \cdots, M_N)$$

$$M_i=\begin{cases} D_i, & \text{if dimension-i is not decomposed.} \\ \lfloor (D_i-1)/p \rfloor+1, & \text{if dimension-i is decomposed.} \end{cases}$$

次に，m 次元のバーチャルプロセッサを，1 次元の物理プロセッサにマッピングする．各物理プロセッサには $(\prod_{i=1}^{N} M_i) \times n^{m-1}$ 要素の配列 A″ をアロケートする．

$$A''(M_1, \cdots, M_N, \underbrace{n, \cdots, n}_{m-1 \text{ times}})$$

例

図 2 で示される例では，もとのソースコードにおいて配列 A は以下のように宣言されている．

```
REAL A(8, 8)
```

この配列 A は分割配置され，各プロセッサ内で

```
REAL A″(2, 2, 4)
```

と，アロケートされる．

4.2 添字変換

本節では，4.1 節で述べた方法で分散されたデータにアクセスするために，もとのソースコードにおいて配列 A に表れる添字から，その要素の owner のプロセッサ番号とそのプロセッサ内の配列 A″の添字の組を求める方法を述べる．これは，もとのソースコードにおける添字から，バーチャルプロセッサ番号とバーチャルプロセッサ内での添字の組を求める変換 f_{cb} と，バーチャルプロセッサ番号から，実プロセッサ番号と実プロセッサ内の添字の組を求める変換 f_{twist} の二つの変換の合成によって求められる．

$$(I_1, \cdots, I_N) \xrightarrow{f_{cb}} ((V_1, \cdots, V_m), (I'_1, \cdots, I'_N)) \xrightarrow{f_{twist}} (p, (I'_1, \cdots, I'_N, J_1, \cdots, J_{m-1}))$$

前者の変換 f_{cb} は，ブロックあるいはサイクリックによる分割として，以下のようにその変換法が知られている[6),9)]．

i 次元目がブロック分割の場合

$$V_i=\lfloor I_i/M_i \rfloor$$
$$I'_i=I_i \bmod M_i$$

i 次元目がサイクリック分割の場合

$$V_i=I_i \bmod n$$
$$I'_i=\lfloor I_i/n \rfloor$$

なおここで，配列 A が i 次元目でブロック分割されている場合にそのブロックサイズを M_i で表す．

また，バーチャルプロセッサから実プロセッサへの変換 f_{twist} が twisted data layout を実現するための拡張であり，3 章の定義より以下のように表せる．

$$p=\left(\sum_{k=1}^{m} v_k\right) \bmod n$$
$$J_i=V_i \quad (1 \le i \le m-1)$$

例

図 2 で示される例において，もとのソースコード内の配列 A に対するアクセス A (I_1, I_2) は，以下の式のように，プロセッサ p 内の A″(I'_1, I'_2, J'_1) へのアクセスとして実現される．

ブロック分割の場合

$$p=(\lfloor I_1/2 \rfloor+\lfloor I_2/2 \rfloor) \bmod 4$$
$$I'_1=I_1 \bmod 2$$
$$I'_2=I_2 \bmod 2$$
$$J'_1=\lfloor I_1/2 \rfloor$$

サイクリック分割の場合

$$p=((I_1 \bmod 2)+(I_2 \bmod 2)) \bmod 4$$
$$I'_1=\lfloor I_1/2 \rfloor$$
$$I'_2=\lfloor I_2/2 \rfloor$$
$$J'_1=(I_1 \bmod 2)$$

4.3 計算マッピング

Twisted data layout したデータを対象に計算の分散法を示す．計算マッピングではバーチャルプロセッサのインデクスを表すループとそのバーチャルプロセッサ内の計算のインデクスを表すループを用いて実行順序をスケジューリングする．これを実現するため，バーチャルプロセッサの各次元にマッピングされるループは，バーチャルプロセッサインデクス用と内部計算用インデクス用の二つのループにストリップマイニングされる．以下のようにステップサイズを 1 に正規化した m 重ループを考える．

```
do i1=1, N1
  do i2=1, N2
    …
    do im=1, Nm
      …
    end do
    …
  end do
end do
```

i_lが並列ループで各ループがそれぞれバーチャルプロセッサの各次元にマッピングされると仮定すると，i_l以外のすべてのループはストリップマイニングされ，各プロセッサにおけるループは以下のような(m×2−1)重ループとしてコードジェネレーションされる．

```
do V1=0, n-1
do i1=L(V1), U(V1), S
  ...
  do Vl-1=, n-1
  do il-1=L(Vl-1), U(Vl-1), S
    do Vl+1=0, n-1
    do il+1=L(Vl+1), U(Vl+1), S
      ...
      do Vm=0, n-1
      do im=L(Vm), U(Vm), S
        Vl=g(V1, ..., Vl-1, Vl+1,,, Vm)
        do il=L(Vl), U(Vl), S
          ...
      end do
      end do
      ...
    end do
    end do
  end do
  end do
  ...
end do
end do
```

バーチャルプロセッサ内の計算のインデクスの上下限を表すLとUおよびステップSは次のように表せる．ここで，B_kはブロック分割における1プロセッサあたりのブロックサイズとする．

ブロック分割の場合

$L(V_k)=V_k*B_k+1$

$U(V_k)=\min(N_k, V_k\times(B_k+1))$

$S=1$

サイクリック分割の場合

$L(V_k)=V_k+1$

$U(V_k)=N_k$

$S=n$

並列に実行すべきバーチャルプロセッサで，物理プロセッサPが担当するものを決定するために，インデクスV_lは，$V_1, \cdots, V_{l-1}, V_{l+1},,, V_m$の値から3章の定義より以下の関数$g$によって求められる．

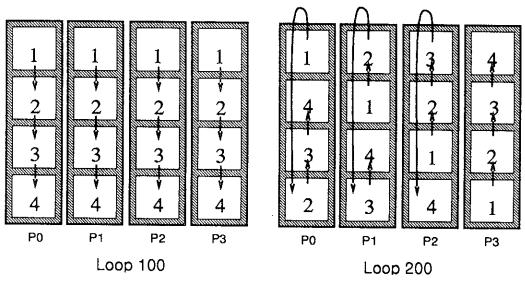


図3 バーチャルプロセッサの実行順序
Fig. 3 Execution order of virtual processors.

$$V_l=g(V_1, \cdots, V_{l-1}, V_{l+1}, \cdots, V_m)$$
$$=\left(p+\left(\sum_{k=1}^{l-1}(n-V_k)+\sum_{k=l+1}^{m}(n-V_k)\right)\right) \bmod n$$

例

図2で示されるブロック分割の例においてList 1のプログラム($N=8$の場合)をowner computes ruleに従って実行する場合のコードを示す．

```
      do 100 Vi=0, 3
        do 100 I'=2 * Vi, 2 * (Vi+1)-1
          Vj=(p+(4-Vi)) mod 4
          do 100 J'=2 * Vj, 2 * (Vj+1)-1
            A(I', J')=...
100   continue
      do 200 Vj=0, 3
        do 200 J'=2 * Vj, 2 * (Vj+1)-1
          Vi=(p+(4-Vj)) mod 4
          do 200 I'=2 * Vi, 2 * (Vi+1)-1
            ...=f(A(I', J'))
200   continue
```

本コードに従って，各実プロセッサにおいて配列Aがアクセスされる様子を**図3**に示す．ここで，正方形は各バーチャルプロセッサに対応する領域，正方形内の数字はアクセス順序を示す．

5. 実験結果

Twisted data layoutの効果を調べるため，Eispackライブラリのサブルーチンの1つである*ELMHES*にtwisted data layoutを適用し，AP 1000上で実行した．AP 1000は富士通で開発された分散メモリ型並列計算機である[16]．

まず，**List 2**に示す*ELMHES*サブルーチンにおける配列Aのローカルデータレイアウトを求める．本実験で用いた配列Aのサイズは512×256($NM\times N$)である．ループ110 (20-24行) と140 (37-38行) において並列性を抽出するためには，配列Aは2次元目が分散されなければならない．しかしながら，ループ

```
      SUBROUTINE ELMHES(NM,N,LOW,IGH,A,INT)
      INTEGER I,J,M,N,LA,NM,IGH,KP1,LOW,MM1,MP1
      DOUBLE PRECISION A(NM,N)
      DOUBLE PRECISION X,Y
      INTEGER INT(IGH)
      LA = IGH - 1
      KP1 = LOW + 1
      IF (LA .LT. KP1) GO TO 200
      DO 180 M = KP1, LA
      MM1 = M - 1
      X = 0.0D0
      I = M
      DO 100 J = M, IGH
      IF (DABS(A(J,MM1)) .LE. DABS(X)) GO TO 100
      X = A(J,MM1)
      I = J
  100 CONTINUE
      INT(M) = I
      IF (I .EQ. M) GO TO 130
      DO 110 J = MM1, N
      Y = A(I,J)
      A(I,J) = A(M,J)
      A(M,J) = Y
  110 CONTINUE
      DO 120 J = 1, IGH
      Y = A(J,I)
      A(J,I) = A(J,M)
      A(J,M) = Y
  120 CONTINUE
  130 IF (X .EQ. 0.0D0) GO TO 180
      MP1 = M + 1
      DO 160 I = MP1, IGH
      Y = A(I,MM1)
      IF (Y .EQ. 0.0D0) GO TO 160
      Y = Y / X
      A(I,MM1) = Y
      DO 140 J = M, N
  140 A(I,J) = A(I,J) - Y * A(M,J)
      DO 150 J = 1, IGH
  150 A(J,M) = A(J,M) + Y * A(J,I)
  160 CONTINUE
  180 CONTINUE
  200 RETURN
      END
```

List 2 ELMHES サブルーチン
List 2 ELMHES Subroutine.

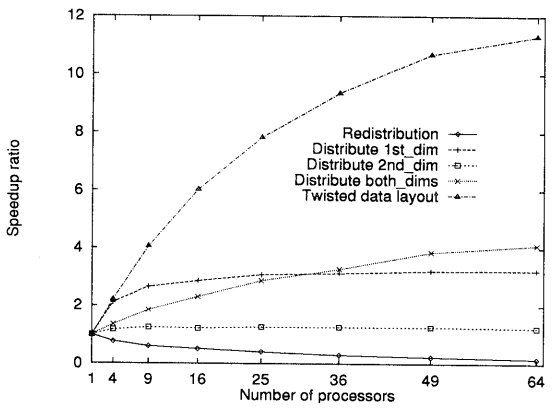


図4 AP 1000 における ELMHES サブルーチンの性能測定結果
Fig. 4 Performance of the ELMHES subroutine using different data layout schemes on the AP 1000.

120 (25-29 行) と 150 (39-40 行) において並列性を抽出するためには，配列 A は 1 次元目が分散されなければならない．

次に，グローバル最適化として，このコンフリクトを解消するために twisted data layout を適用する．ここで，計算処理の分散は owner computates rule に従うものとする．性能評価の結果を**図4**に示す．twisted data layout によるスピードアップ性能 (図中 "Twisted data layout") を再分散による性能 (図中 "Redistribution") と従来型のブロック分散と比較する．ブロック分散では，1 次元目を分散するもの (図中 "Distribute 1st_dimension")，2 次元目を分散するもの (図中 "Distribute 2nd_dimension")，それぞれの次元に $\sqrt{n}$ プロセッサ (n はプロセッサ総数) を割り当てて両方の次元について分散するもの (図中 "Distribute both_dimensions") に分け測定する．速度向上率 (speedup ratio) は，並列化する前のコードを AP 1000 の 1 プロセッサで逐次実行した実行時間をもとに正規化している．本実験結果は，最も良い速度向上率が twisted data layout によって得られることを示している．

6. 考　　察

まず，従来型のブロック分散で配列 A を 1 次元分散した版と 2 次元分散した版とを比較する．プロセッサ台数が 36 未満の場合には，配列 A の 1 次元目を分散する版の性能が，両方の次元について分散する版の性能を上回っている．しかし，より大きなプロセッサ台数では，配列 A の両方の次元について分散するほうが，どちらか 1 次元方向に分散する版の性能を上回っている．本結果は，対象とするコードの並列効果を引き出すためには，配列はすべての並列実行可能な方向に対して分散されなければならないことを示している．これは，アムダールの法則から明らかである．例えば，全実行時間の半分に相当するコードしか並列化できないとしたら，何台プロセッサを使おうと 2 倍以上の速度向上は得られない．多数のプロセッサにより高い性能を得ようとする場合には，プログラム中のできるだけ多くの部分を並列化することが重要である．

Twisted data layout による版は，常にこれらブロック分散による版を上回る性能を示した．これは，twisted data layout が，両次元分散の持つすべての並列性を引き出す効果を持ち，さらにすべてのプロセッサを並列ループの処理に参加させることが可能であるためと考えられる．Twisted data layout は，データ分散の方向に関するコンフリクトを解消するための効果的なテクニックといえる．

すべての並列性を抽出する別の方法であるデータの再分散を twisted data layout と比較する．*ELMHES* の例では，ループにおける粒度が小さいことが原因で，再分散のオーバヘッドは大きく 1 プロセッサで実行するより悪い結果が出ている．一般にデータの再分散を行うか，twist を採用するかは，それぞれに伴う通信の

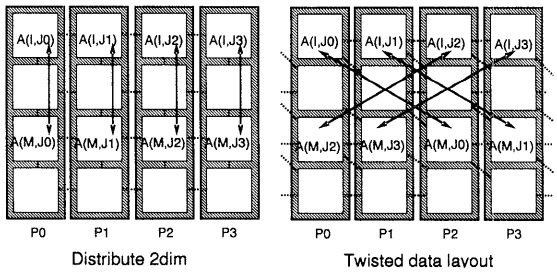


図5 Twisted data layout と2次元分割配置における通信パターンの比較

Fig. 5 Comparison of data communication pattern for twisted data layout and distributing along the second dimension of array A.

オーバヘッドの大きさで決定できる．Twisted data layout では並列化の対象となるすべての次元がプロセッサ間に分散される．そのため，並列化の対象にならない次元に沿ったアクセスで通信が必要になる場合がある．例として，*ELMHES* のループ140をとりあげる．例えば，配列Aが2次元方向に分散されていれば，A(M, J)のアクセスは通信無しに行えるが，配列Aがtwistされている場合には，通信を必要とする場合がある．図5において，"Distribute 2dim" は再分散により2次元目を分散した最適な配置を示し，"Twisted data layout"は twisted data layout を適用した配置を示す．ここで，矢印は配列データの参照関係を示す．ループ140にとって最適な配置の場合には，データの参照は同一プロセッサ内のみであり，通信を必要としない．一方，twisted data layout を用いた場合には，通信を必要とする．Twisted data layout が有効なケースは，この通信に伴うオーバヘッドがデータ再分散のオーバヘッドより小さい場合である．また，twisted data layout の扱える対象は，分散すべき次元が異なることから発生するデータレイアウトのコンフリクトに限定されている．従って，他の原因によるコンフリクトに対してはデータ再分散のようなその他のコンフリクト解消法が必要になる．データレイアウトのコンフリクトを起こすその他の原因としては，配列のアラインメントの際のオフセットの違いや，幅付きのサイクリックが許されている場合にその幅の違いによるものがある．

7. 関連する研究

多くの研究者が分散メモリ型並列計算機のためのデータレイアウト問題に取り組んでいる．

Yale 大学の Li と Chen は配列内の通信コストを最小化するデータアラインメントのテクニックを開発した[8]．Illinois 大学の Gupta と Banerjee は，Li と Chen の方式を基本にコンストレイント・ベースのアプローチを FORTRAN 77 に適用した[9]．どちらの研究もブロックあるいはサイクリックによるデータ分散方式を採用している．Stanford 大学の Anderson と Lam は，自動のデータ分散と計算処理の分散に対しアルゴリズムによるアプローチを提案している[10]．彼女らはグローバル最適化として，データの再分散を最小化する動的な分散のためのアルゴリズムを紹介した．

データスキューイング（data skewing）は，SIMD アレイやベクトルパイプライン型プロセッサのメモリシステムにおけるコンフリクトを避けるためのテクニックとして，広範囲にわたって研究されてきた[12)~15)]．我々は，スキューイングの考えを分散メモリ型並列計算機に拡張したデータレイアウト法として twisted data layout を提案した．我々は"twist" という言葉を採用することで，アフィン変換による "スキュー" したマッピング[11]との混同を避ける．

ブロックあるいはサイクリックによるデータ分散を指定するディレクティブを提供した並列処理用の高級言語が，分散メモリ型並列計算機のために数多く開発されている[1)~7)]．Twisted data layout テクニックは，さらなるデータレイアウトの最適化のために，これらのレイアウト用のディレクティブとともに用いることができる．

8. おわりに

分散メモリ型並列計算機上における配列のデータレイアウトは，ローカルデータレイアウトとグローバル最適化の2レベルで行う．我々は，第1のレベルで決定されるローカルデータレイアウト間でのコンフリクトを解消するためのグローバル最適化として，twisted data layout テクニックを提案した．Twisted data layout は，バーチャルプロセッサから物理プロセッサへのツイストしたマッピングの導入により，従来型のブロックやサイクリックによるデータ分散法を拡張している．本手法は，ディレクティブを用いた人手によるデータ分散指定されたプログラムの最適化にも，自動データレイアウトのモデルにも導入することができる．本論文では，twisted data layout を実現するためのコードジェネレーション法を紹介し，実験結果により，その効果を示した．

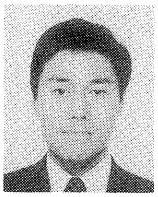
謝辞 本研究に対し御支援，御指導いただいています石井光雄氏，白石 博氏，佐藤弘幸氏，池坂守夫氏に感謝いたします．本研究に関する討論に参加し貴重な意見を下さりました和田英一東大名誉教授，河村

薫氏，堀江健志氏に感謝いたします．また，著者の一人進藤が客員研究員としてスタンフォード大学滞在中に本研究の動機を与えて下さりました Monica Lam 教授に感謝いたします．

参 考 文 献

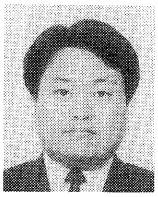
1) High Performance Fortran Forum: *High Performance Fortran Language Specification Ver. 1. 0.* (1993).
2) Hiranandani, S., Kennedy, K. and Tseng, C.: Compiler Optimizations for Fortran D on MIMD Distributed-Memory Machines, *Proc. Supercomputing'91*, pp. 86-100 (Nov. 1991).
3) Rühl, R. and Annaratone, M.: Parallelization of FORTRAN Code on Distributed-memory Parallel Processors, *Proc. Int. Conf. SUPERCOMPUTING*, pp. 342-353 (June 1990).
4) Zima, H., Bast, H. and Gerndt, M.: SUPERB: A Tool for Semi-automatic MIMD/SIMD Parallelization, *Parallel Computing*, Vol. 6, pp. 1-18 (1988).
5) Rogers, A. and Pingali, K.: Process Decomposition through Locality of Reference, *Proc. ACM SIGPLAN'89 Conf. Programming Language Design and Implementation*, pp. 69-80 (June 1989).
6) Koelbel, C. and Mehrotra, P.: Compiling Global Name-Space Parallel Loops for Distributed Execution, *IEEE Trans. Parallel and Distributed Systems*, pp. 440-451 (Oct. 1991).
7) 進藤達也，岩下英俊，土肥実久，萩原純一：AP 1000 を対象とした VPP Fortran 処理系の実現と評価，SWoPP 鞆の浦 '93 HPC 研究会，Vol. 93-HPC-48-2, pp. 9-16 (Aug. 1993).
8) Li, J. and Chen, M.: Index Domain Alignment: Minimizing Cost of Cross-Referencing Between Distributed Arrays, *Proc. Frontiers'90 : The Third Symposium on the Frontiers of Massively Parallel Computation*, pp. 424-433 (Oct. 1990).
9) Gupta, M. and Banerjee, P.: Demonstration of Automatic Data Partitioning Techniques for Parallelizing Compilers on Multicomputers, *IEEE Trans. Parallel and Distributed Systems*, pp. 179-193 (Mar. 1992).
10) Anderson, J. M. and Lam, M. S.: Global Optimizations for Parallelism and Locality, *Proc. ACM SIGPLAN'93 Conf. Programming Language Design and Implementation*, pp. 112-125 (June 1993).
11) Amarasinghe, S. P. and Lam, M. S.: Communication Optimization and Code Generation for Distributed Memory Machines, *Proc. ACM SIGPLAN'93 Conf. Programming Language Design and Implementation*, pp. 126-138 (June 1993).
12) Budnik, P. and Kuck, D. J.: The Organization and Use of Parallel Memories, *IEEE Trans. Computers*, Vol. C-20, pp. 1566-1569 (Dec. 1971).
13) Lawrie, D. H.: Access and Alignment of Data in an Array Processor, *IEEE Trans. Computers*, Vol. C-24, pp. 1145-1155 (Dec. 1975).
14) Harper, D. T. III and Jump, J. R.: Vector Access Performance in Parallel Memories Using a Skewed Storage Scheme, *IEEE Trans. Computers*, Vol. C-36, pp. 1440-1449 (Dec. 1987).
15) Nortron, A. and Melton, E.: A class of Boolean Linear Transformations for Conflict-free Power-of-two Access, *Proc. 1987 Int. Conf. Parallel Processing*, pp. 247-254 (Aug. 1987).
16) 石畑宏明，稲野 聡，堀江健志，清水俊幸，池坂守夫：高並列計算機 AP 1000 のアーキテクチャ，信学論 (D-1)，Vol. J75-D-1, No. 8, pp. 637-645 (1992).

(平成 6 年 9 月 21 日受付)
(平成 7 年 1 月 12 日採録)



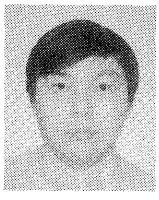
進藤 達也（正会員）

1983 年早稲田大学理工学部電子通信学科卒業．1983 年より（株）富士通研究所．1990-1992 年スタンフォード大学客員研究員．1995 年より富士通（株）．CAD 専用マシン，並列処理の研究に従事．1986 年篠原記念学術奨励賞．



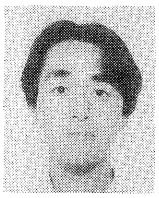
岩下 英俊（正会員）

1986 年愛媛大学工学部電子工学科卒業．1988 年同大学院工学研究科修士課程修了．同年（株）富士通研究所入社，1995 年より富士通（株）．並列 Fortran 言語の設計，並列化コンパイラの研究開発に従事．電子情報通信学会会員．



土肥 実久（正会員）

1988 年大阪府立大学工学部電気工学科卒業．1990 年同大学院工学研究科電気工学専攻博士前期課程修了．同年（株）富士通研究所入社．1995 年より富士通（株）機械翻訳，CAD システム，並列コンパイラの研究に従事．



萩原　純一（正会員）

1991年早稲田大学理工学部電子通信学科卒業．1993年同大大学院修士課程修了．同年（株）富士通研究所入社．1995年より富士通（株）に勤務．並列処理システム，並列化コンパイラの研究に従事．IEEE，ACM各会員．

Shaun Y. Kaneshiro

1991年マサチューセッツ工科大学Department of Electrical Engineering and Computer Science卒業．1993年マサチューセッツ工科大学修士課程修了．1993年よりMIT-Japan Internship Programにより（株）富士通研究所勤務．並列処理システム，並列化コンパイラの研究に従事．Tau Beta Pi member．