情報処理学会論文誌　プログラミング　Vol. 3　No. 1　41–53 (Mar. 2010)

配列整合解析に基づく自動データ分割手法

窪田昌史[†1]　北村俊明[†1]

大規模な分散メモリ型計算機向けにプログラムを並列化する際に，HPF などの並列プログラミング言語を用いることでプロセス間のデータ通信などを含む並列プログラムを自動的に生成することがが可能となってきた．しかしながら，HPF などでプログラムを記述するには，プログラマが配列変数などの分割を決定して手動で記述する必要がある．本論文では，このデータ分割を自動的に決定するための新たな配列整合解析手法を提案する．従来のデータ分割決定手法としては，Component Affinity Graph（CAG）と呼ばれるグラフを用いた複数の配列の次元間の関係を解析する手法が提案されているが，提案手法では，多重ループにおける多次元配列の参照関係を，ループの並列性を考慮しながら解析することで，より的確なデータ分割を求めることを可能とする．本手法を用いて NPB 3.2-SER BT ベンチマークのデータ分割を解析して HPF ディレクティブが挿入された Fortran プログラムを生成することが可能となった．生成された HPF プログラムを商用の HPF コンパイラを用いて並列化して PC クラスタで実行したところ，NPB3.0-HPF と同様の台数効果が得られた．

An Automatic Data Distribution Method Based on Array Alignment Analysis

Atsushi Kubota[†1] and Toshiaki Kitamura[†1]

When a programmer parallelizes programs for large-scale distributed memory machines, it has been enabled for programmers to generate parallelized programs with inter-process data transfers by HPF language. However, it is required for programmers to specify distribution for arrays when they write HPF programs. Therefore, this paper proposes a new method for array alignment analysis to determine array data distribution automatically. One of the conventional data distribution methods is the Component Affinity Graph (CAG) based method where relations between multiple array dimensions accessed in nested loops are analyzed. With our method, it is possible to determine more proper data distribution for arrays in multiply nested loops by analyzing relations of dimensions of the arrays under consideration of parallelism of the loops. With the proposed method, the Fortran program where HPF data distribution directives are inserted can be generated automatically. The generated HPF program was parallelized by the PGI HPF compiler and executed on a PC cluster. We confirmed that scalability of the benchmark is close to those of NPB3.0-HPF.

1. はじめに

並列計算機，PC クラスタ，マルチコアプロセッサなどの普及により，並列処理を行う環境は急速に広まってきている．Message Passing Interface（MPI）[1),2)] は，現在，並列プログラミング，特に大規模な並列処理のためのライブラリとして最も普及しているといえる．しかしながら，MPI を用いてプログラムを記述する場合に，プロセス間の同期やメッセージ通信を逐一記述することは，作業量が多く，また，間違いを犯しやすいため，プログラマにとって大きな負担となることが指摘されている．

そこで，並列プログラミングの生産性向上のため，様々な高水準並列プログラミング言語が提供，提案されている．これらの言語では，並列性をどのように記述するか，大規模な配列データをどのように並列プロセスに割り付けるかが特徴となる．

High Performance Fortran（HPF）[3)–5)] は Fortran を拡張した言語である．HPF では並列ループで処理される大規模な配列変数に着目し，それらの配列変数を並列計算機内で分散されたメモリにどのように配置するかをプログラマが指示文（ディレクティブ）で指定し，HPF コンパイラがその指示文に従ってプロセス間のメッセージ通信などの呼び出しを自動的に挿入して並列プログラムを生成する．近年，HPF コンパイラの技術開発の進展（文献 6) など）により，地球シミュレータ向けの HPF/ES [7)] のような大規模な並列計算機で利用可能な HPF コンパイラから，PC クラスタでも利用可能な PGI 社の HPF コンパイラまで，様々なプラットフォームで HPF コンパイラが利用可能となっている．

ループなどの並列性を指示文で明示的に指定する並列プログラミングの API としては OpenMP [8)] が普及している．マルチコアプロセッサや SMP などの典型的な共有メモリ型並列計算機での並列処理に向いているが，ソフトウェア，あるいはハードウェアによる分散共有メモリに対する実装により，大規模な配列を扱うことも試みられている．

また，Co-Array Fortran [9)]，X10 [10)]，Chapel [11)]，Fortress [12)] などの並列プログラミン

†1 広島市立大学
Hiroshima City University

グ言語が提案されているが，これらの言語を用いて並列プログラムを記述する場合には，並列性を明示的に，あるいは暗黙的に指定する必要がある．Co-Array Fortran や X10，Chapel などは Partitioned Global Address Space（PGAS）言語と呼ばれ，分散メモリに対するデータ分割を意識してプログラムを記述しなければならない．

このように，高水準の並列プログラミング言語を用いる場合，ループの並列性や大規模な配列変数の分割を意識する必要がある．そのため，これらの並列性やデータ分割の解析を自動化することで，並列化を支援するための研究が多数行われてきた．ループの並列性の解析については，データ依存方程式の解を求める手法などがあり[13]，商用コンパイラなどでも広く用いられている．一方，データ分割を自動的に求める手法についても，様々なアルゴリズムが提案されてきた．

Li ら[14] はデータ分割問題が NP 完全であることを示し，Component Affinity Graph（CAG）という複数の配列の次元間の関係を表すデータ構造を用いて近似解を求めるヒューリスティクスを提案した．CAG を用いる手法を基に Gupta ら[15]，辰巳ら[16] はデータ分割を求める手法を提案した．Kennedy ら[17),18] はループネストごとに CAG を用いて配列の次元間の関係を表現し，複数ループネスト間の最適なデータ分割の問題を 0-1 整数計画問題として定式化した．Kennedy らの手法はヒューリスティクスではあるものの，データの再分割を求めるのに有効な手法であるが，処理時間が長いことが報告されている．松浦ら[19] は手続き間解析を含めた自動データ分割手法を現実的な時間で求める手法を提案した．廣岡ら[20),21] は，分散共有メモリ計算機を対象にして，ループの並列性も考慮した効率的なデータ分散のための手法を提案した．

本論文では，データ分割を自動的に決定するための新たな配列整合解析手法を提案する．本手法では，多重ループにおける多次元配列の参照関係を，ループの並列性を考慮しながら解析することで，より的確なデータ分割を求めることを可能とする．

提案する手法は，ループの並列性が抽出されたデータ並列 Fortran プログラムを解析対象とし，求めたデータ分割は HPF の指示文としてプログラムに挿入する．

以下，2 章でデータ分割の従来手法として知られている CAG とその問題点について説明し，3 章で我々の提案する配列整合解析について述べる．4 章で，提案手法を用いて自動データ分割を行ったプログラムの性能評価を行い，5 章でまとめとする．

2. 並列プログラムにおける配列データの分割

2.1 ループの並列性とデータ分割

配列の各次元の添字には，ループ制御変数はたかだか 1 つしか現れないことが多い．そのため，以下のような典型的な並列ループでは，配列の各次元をループに対応づけることが容易である．

```
DO J=1,N
  DO I=1,N
    A(I,J) = B(I,J)
  ENDDO
ENDDO
```

一方，ループが並列に実行できる場合でも，データ参照にともなってプロセス間の通信が必要になるためにデータ分割を考慮すべき場合がある．

```
DO I=1,N-1
  A(I) = B(I-1)+B(I)+B(I+1)
ENDDO
```

このループは並列化可能であり配列 A は分割すべきである．しかし，配列 B が分割配置されていると，配列 B への参照にともなってプロセス間通信を行う必要がある．一般的には，このプロセス間通信がオーバヘッドとなり実行速度が低下する．一方，配列 B を重複分割してプロセス間通信が生じないようにすることも可能である．その場合，重複分割することでプログラムの他の部分では並列性が損なわれることもある．プロセス間通信を生じさせてでも配列を分割するかどうかは，このようなトレードオフを考慮しながら判断する必要がある．

また，1 次元の配列ではなく多次元の配列を考える場合には，複数の次元のうち，並列性が得られる次元を選ぶことになる．そのような次元が複数存在する場合は，配列要素への参照によるプロセス間通信によるオーバヘッドを基準に分割すべき次元の選択肢をさらに絞り込むことになる．

```
DO J=1,N
   DO I=1,N-1
      A(I,J) = B(I-1,J)+B(I,J)+B(I+1,J)
   ENDDO
ENDDO
```

```
DO K=1,N
  DO J=1,N
    DO I1=1,N
      FJAC(I1)=U(I1,J,K)+SQUARE(I1,J,K)
    ENDDO
    DO I2=2,N-1
      LHS(I2)=FJAC(I2-1)+FJAC(I2)+FJAC(I2+1)
    ENDDO
    DO I3=2,N-1
      RHS(I3,J,K)=RHS(I3,J,K)+LHS(I3)
    ENDDO
  ENDDO
ENDDO
```

図 1　NPB3.2-SER BT x_solve サブルーチン（一部抜粋）
Fig. 1　A Fragment of NPB3.2-SER BT x_solve subroutine.

この例では配列 B の 1 次元目は分割すべきではないことになる．配列 B の 2 次元目を分割することで，プログラム全体の並列性が得られるのであれば，配列 B の 1 次元目を分割する必要はない．配列 A，B ともに 2 次元目で分割するためには，HPF では以下のような指示文を挿入することになる．

```
!HPF$ DISTRIBUTE A(*,BLOCK)
!HPF$ DISTRIBUTE B(*,BLOCK)
```

2.2　CAG を用いたデータ分割手法

前節で述べたように，配列データの分割を決定するには，データの並列性や並列プログラムのプロセス間通信を考慮する必要がある．

図 1 の例は，NAS Parallel Benchmarks（NPB）3.2 のアプリケーション BT の主要カーネルの一部を，ここでの説明のために簡略化したものである．この節では，このカーネル部分のデータ分割を，CAG を用いる従来手法[14] で決定する場合の問題点について考察する．

このカーネルでは，K，J，I1，I2，I3 のすべてのループが並列化可能である．しかし，配列 FJAC を分割すると，ループ I2 では FJAC の参照にプロセス間の通信が必要となる．そこで，FJAC をすべてのプロセスについて重複分割することを考えてみる．このとき，ループ I1 を並列化しても，FJAC が重複分割されていることにより性能向上は望めない．よって，このループネスト全体は，外側のループ J または K を並列化する方が性能向上を期待できる．このとき，FJAC は重複分割されているので各プロセス上で**私有化**（*privatization*）される．

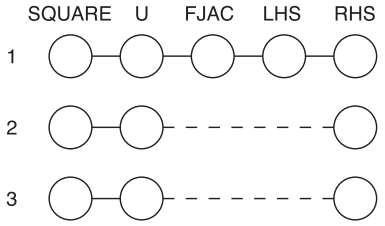


図 2　NPB BT x_solve に対する CAG
Fig. 2　CAG for NPB BT x_solve subroutine.

あるループを並列化するときには，そのループ内で参照される各配列のどの次元を分割すべきかを解析する必要がある．異なる配列の次元の添字に同じループ制御変数が現れる場合，それらの次元間の関係を表すために，**図 2** に示すような *Component Affinity Graph*（*CAG*）[14] というグラフが用いられてきた．

たとえば図 1 の配列 U の 1 次元目と配列 FJAC の 1 次元目にはともにループ制御変数 I1 が現れるので，図 2 に示すようにそれぞれの配列の 1 次元目に対応するノード間がエッジで接続される．

CAG においてエッジで接続されているノードは同一グループにまとめられる．図 2 に示すように，すべての配列の 1 次元目がグループ化され，さらに，配列 SQUARE と U の 2 次元目，3 次元目がグループ化される．配列 RHS の 2 次元目と 3 次元目は，それぞれ単独でグループを構成する．

データ分割の解析は，これらのグループを対象として行われる．あるグループが分割されることになると，グループ内の各配列の次元は一括して分割される．分割対象のグループとして選択されるのは，並列化したときに最も高い性能が期待できるものである．このとき，グループ内の各ノードを接続するエッジに対応するループも並列化の対象となる．たとえば，図 2 の 1 次元目のグループを分割対象する場合は図 1 の I1，I2，I3 の 3 個のループが並列化の対象となる．

さて，上の配列の私有化の議論で述べたように，図 1 のループは J または K のループで並列化することが望ましい．その場合に，ループ J の制御変数が現れる SQUARE，U，

RHS の 2 次元目を分割するか，ループ K の制御変数が現れるそれらの配列の 3 次元目を分割するかを選択することになる．しかし，図 2 の CAG では配列 SQUARE，U と配列 RHS の 2 次元目や 3 次元目が接続されないため，点線で示すような次元間の関係が抽出できず，配列 SQUARE，U，RHS は 2 次元目（あるいはそれぞれの 3 次元目）で分割すべきであるという解析結果を得ることは難しい．

我々は，このような問題が生じているのは，ループの並列性と配列の次元の間に強い関連があるにもかかわらず，CAG ではその関連を適切に表現できないためと考える．次章では，この問題を解決するための新たな手法として，アクセス関数を用いた配列整合解析手法を提案する．

3. 配列整合解析

本章では，ループネスト中でアクセスされる配列の分割すべき次元と，並列化すべきループを，配列の添字式の解析に基づいて求めるアルゴリズムについて述べる．まず，手続き内の配列の分割と並列化すべきループを求める手法について述べ，次に手続き間の再分割についての解析手法について述べる．手続き内の解析については，さらに，分割すべき配列の次元と並列化すべきループの候補を求める手法と，それらの候補の中から，実際のプロセッサ集合へと分割すべき配列の次元とループを求める手法に分けて述べる．

3.1 手続き内解析

3.1.1 アクセス関数

以下のような並列多重ループを考える．

```
DO J=1,N
  DO I=1,N
    A(I,J) = B(I,J)
  ENDDO
ENDDO
```

このようにループ内で参照される配列の添字を，以下のような**アクセス関数**[13] を用いて $\mathbf{a}=\mathbf{A}\mathbf{i}+\mathbf{c}$ と表現することができる．

$$\begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} J \\ I \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad (1)$$

ここで $\mathbf{a}=(a_1, a_2)^T$ の各次元はそれぞれ配列 A の 1 次元目，2 次元目の添字を表す．ループ制御変数のベクトル $\mathbf{i}=(J, I)^T$ の係数行列 $\mathbf{A}$ をデータアクセス行列，定数項 $\mathbf{c}=(0, 0)^T$ をデータオフセットベクトルと呼ぶ．

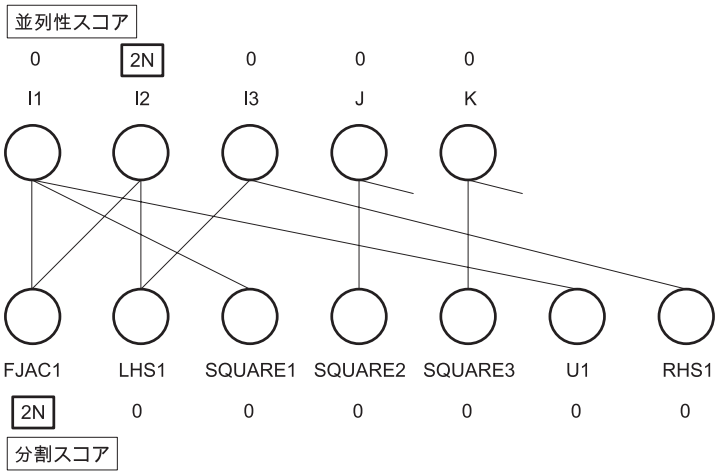


図 3　NPB BT x_solve の初期スコア
Fig. 3　Initial scores for NPB BT x_solve subroutine.

このアクセス関数で表すことのできる添字式は，いくつかの変数の 1 次式であるアフィン式である．アフィン式で表すことのできない添字が配列の次元に現れるような配列参照が並列ループ内に現れると，当該配列は分散メモリ上に分割配置されることは不可能であるとして分割対象から外される．

配列の次元 d の添字にループ制御変数 l が現れる場合，アクセス関数のデータアクセス行列の d 行 l 列の要素は非零となる．この関係を**図 3** に示すようなグラフのノード間をエッジで接続することで表現する．このグラフは図 1 の BT の例について，各ループに対応するループノードを上側に，配列の各次元に対応する配列次元ノードを下側に配置した 2 部グラフである．ループ制御変数と配列の次元間の関係を求める操作は，データアクセス行列の非零要素を求める操作として実装することができる．なお，各ループノードの並列性スコアと各配列次元ノードの分割スコアについては 3.1.2 項以降で詳述する．

我々の解析手法においては，複数のアクセス関数の**併合**が行われる．アクセス関数の併合は，各ループごとに同じ配列のアクセス関数について行われる．I，I+1，I−1 などの場合は I + [−1:1] のように併合される．ループごとという意味は，多重ループでは着目してい

るループのループボディに現れる実行文に現れる配列参照のアクセス関数が併合されるという意味である．同一のループボディ内に

```
... = B(I)
... = B(I+1)
... = B(I-1)
... = B(I)
```

というように，複数の実行文にわたって同じ配列が参照される場合にもアクセス関数は併合される．ただし，着目しているループの内側ループの配列参照のアクセス関数は併合の対象にはならない．

3.1.2　分割次元決定フェーズ

このフェーズでは，ループを以下のように分類し，**並列性スコア**を定める．ここで，並列性スコアは 0 以上の値とする．0 の場合が並列実行に支障がないことを表す．また，スコアが大きな値を持つほど，並列実行に支障が出ることを示す．

(1)　並列実行可能で，通信が不要．並列性スコア=0.
(2)　並列実行可能だが，通信が必要．並列性スコア=通信量.
(3)　並列実行可能で，当該ループ内では通信が不要だが，他のループでは通信が必要な配列を参照している．並列性スコア=ϵ.
(4)　並列実行不可能．並列性スコア=ループ内で参照される配列全要素のサイズ.

さらに，上の (1)–(4) のループのループ制御変数が添字に現れるかどうかで，配列の次元の**分割スコア**を決定する．ここで，分割スコアも 0 以上の値とする．0 の場合に分割に支障がないことを表す．また，スコアが大きな値を持つほど，分割に支障が出ることを示す．

(1)　分割可能．分割スコア=0.
(2)　分割可能だが，通信の制約あり．分割スコア=通信量.
(3)　分割可能だが，他の配列との関連により制約あり．分割スコア=ϵ.
(4)　分割不可．分割スコア=配列全要素のサイズ.

これらの並列性スコア，分割スコアを求めるため，まずループネスト内での解析によってスコアを求め，次に，ループネスト間の配列の参照関係によってスコアを伝播させる．以下，ループネスト内のスコア設定とネスト間のスコア伝播に分けて説明する．

3.1.3　ループネスト内のスコア設定

あらかじめ，各ループの並列実行可能性をデータ依存解析，データフロー解析によって調べ，各ループを DO ALL 型実行可能か否かで分類しておく．ループのイタレーション間でデータ依存が運搬されるループ運搬依存の場合は DO ALL 型実行は不可能であるとする．データ依存が各イタレーション内でのみ存在し，イタレーション間では運搬されない場合は，DOALL 型実行可能とする．

並列ループ内の分割スコア

ループ l が DO ALL 型ループで並列実行可能な場合，配列の次元 d の添字によって，プロセス間通信が必要な場合とそうではない場合がある．これは，ループボディに現れる代入文によって以下のように判断する．

- A(I) = B(I) のような場合は通信は不要である．
- A(I) = B(I+1) のような場合，通信は不要である．
- A(I) = B(I-1) + B(I) + B(I+1) のような場合，通信が必要となる．

最後の場合は，配列 B のアクセス関数は，併合により $\mathrm{I} + [-1:1]$ となり，データオフセットベクトルの要素数が 3 となり 1 ではない．このようにデータオフセットベクトルの要素数が 1 となるかどうかによって通信が必要となるかどうかを判定する．この場合に必要となる通信量は，隣接するプロセス間で shift 型の通信が行われるものとして求める．求めた通信量は，配列の次元 d の分割スコアへ加算される．通信量は，一般的には実行時に使用されるプロセス数によって変化する．提案する手法ではプロセス数が与えられなくても通信量を見積もることができるよう，通信量が最も多くなる場合の通信量を求めるものとする．

たとえば，上記の A(I)=B(I−1)+B(I)+B(I+1) のような代入文で，配列 A，B の要素数が N であるとする．A(I) と B(I) が同じプロセスに配置されるとすると，B(I−1) と B(I+1) の参照にはプロセス間通信を生じる可能性がある．配列 B が 1 要素ずつ N プロセスに分割して配置される場合，B(I−1) と B(I+1) の参照はすべてプロセス間通信が必要となり，それぞれの通信量は N となる．よって，この代入文で発生する通信量は，2N であると見積もられ，これが配列 B の分割スコアとなる．

逐次ループ内の分割スコア

ループ l が DO ALL 型ループではない場合，ループ運搬データ依存が存在する．この場合，ループの制御変数 l が現れる配列の次元 d の分割スコアを配列の全要素数とする．これは，ループ l の並列実行が不可能となり逐次実行されるにもかかわらず，配列の次元 d が分割されることになれば，この配列の全要素を送受信するような大量のプロセス間通信が発生して性能が低下することを，次元 d の分割スコアを増加させることで表している．

ループの並列性スコア

ここまでで，ループ l で参照されるすべての配列の次元 d について，分割スコアが設定

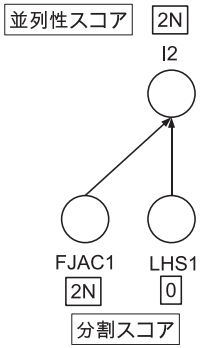


図 4　ループ I2 の初期スコア
Fig. 4　Initial scores for loop I2.

されたことになる．次に，ループ l の並列性スコアを求める．ループの制御変数 l が，配列 A1 の次元 d_1，配列 A2 の次元 d_2 ... 配列 AN の次元 d_n の添字に現れ，それぞれの次元の分割スコアを s_1, $s_2 \ldots s_n$ とする．ループ l を並列実行するときに実行時間を増加させるオーバヘッド $\sum_{i=1}^{n} s_i$ を，ループ l の並列性スコアとして設定する．

手続き内の分割スコア

ある配列が複数のループネストで参照され，それぞれのループネストでその配列の次元の分割スコアが異なる可能性がある．手続き内での配列の次元の分割コストは，それぞれのループネストにおける分割スコアの和として求められる．

例：NPB-BT

図 1 の BT の例では，すべてのループが並列化可能であり，これらの並列ループ内で参照される配列の分割にともなうプロセス間通信の発生の有無を解析することになる．ループ I2 には，FJAC(I2−1)+FJAC(I2)+FJAC(I2+1) という参照が現れる．この配列 FJAC のアクセス関数を併合すると I2 +[−1 : 1] となり，データオフセットベクトル [−1 : 1] の要素数が 3 であり 1 ではないため，通信が発生する可能性があると判定される．ループ I2 を並列実行する際のプロセス間の通信量は，配列 FJAC の要素数が N であれば，2N と見積もられる．この 2N が FJAC の 1 次元目の分割スコアとなる．他の配列については，それらの配列の各次元を分割しても並列実行時にプロセス間通信が発生する可能性がないため，分割スコアは 0 である．

各ループの並列性スコアは，そのループ内で参照される配列の各次元の分割スコアの総和となる．ループ I2 の並列性スコアは，**図 4** に示すように FJAC の 1 次元目の分割スコア 2N と LHS の 1 次元目の分割スコア 0 の和 2N となる．他のループは，そのループ内で参照される配列の次元の分割スコアがすべて 0 であるため，それらの和となる並列性スコアも 0 となる．

以上により，分割スコアと並列性スコアの初期値が確定する．図 3 に示すとおり，FJAC の 1 次元目のノードの分割スコアが 2N となり，ループ I2 のノードの並列性スコアが 2N となる．他の分割スコアと並列性スコアはすべて 0 である．

なお，配列 U，RHS の 2，3 次元目のノードはそれぞれループ J，K のノードと接続され分割スコアは 0 となる．配列 SQUARE の 2，3 次元目と同様であるため図 3 からは省略している．

3.1.4　ループネスト間のスコア伝播

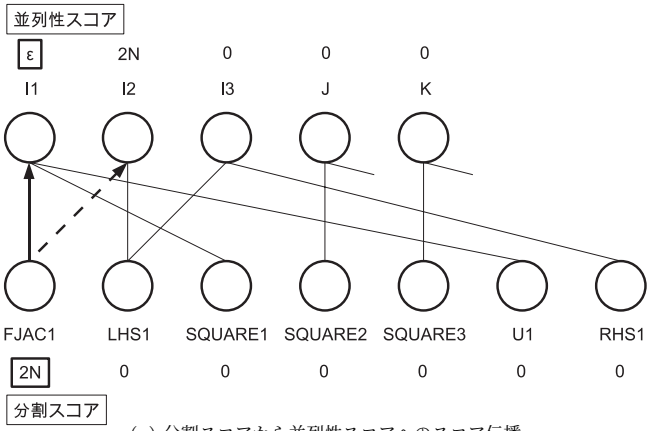
ここまでで，図 1 の BT の例では，FJAC の 1 次元目には分割スコア 2N が設定されていることになる．このとき，ループ I1 は並列性スコアが 0 であるが，ループ I1 で参照される配列 FJAC の 1 次元目の分割スコアが 0 ではないため，ループ I1 は並列化の候補からは除外すべきである．ただし，このループ I1 を並列化してもプロセス間通信のオーバヘッドが必ず生じるというわけではないため，限りなく 0 に近いが 0 ではないという意味で並列性スコアを ϵ とする．

このように，他のループネストで分割スコアあるいは並列性スコアが 0 ではないと判定されたために，その情報が伝播されることで 0 ではない ϵ のスコアが設定される．この伝播は，すべてのループ l について，アクセス関数を用いてそのループ内で参照される配列の次元 d を求め，以下のように行う．

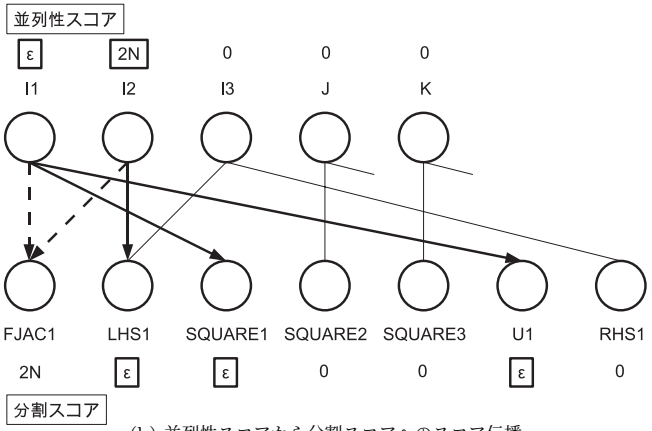
(1)　配列の次元 d の分割スコアが 0 でなく（分割配置にともなうオーバヘッドを生じる），ループ l の並列性スコアが 0（オーバヘッドなしで並列実行可能）の場合，ループ l の並列性スコアを ϵ へ更新する．

(2)　ループ l の並列性スコアが 0 でなく（並列実行によるオーバヘッドを生じる），次元 d の分割スコアが 0（分割配置してもオーバヘッドは生じない）の場合，次元 d の分割スコアを ϵ へ更新する．

図 1 の BT の例では，まず，分割スコアが 0 ではない FJAC1 から**図 5** (a) に示すようにループの並列性スコアへの伝播が起こる．FJAC1 ノードに接続されているのは，I1 と I2 のノードであるが，このうち，I2 はすでに 0 ではない並列性スコアが設定されているので更新されないのに対し，I1 は並列性スコアが 0 であるため，その並列性スコアが ϵ に更新

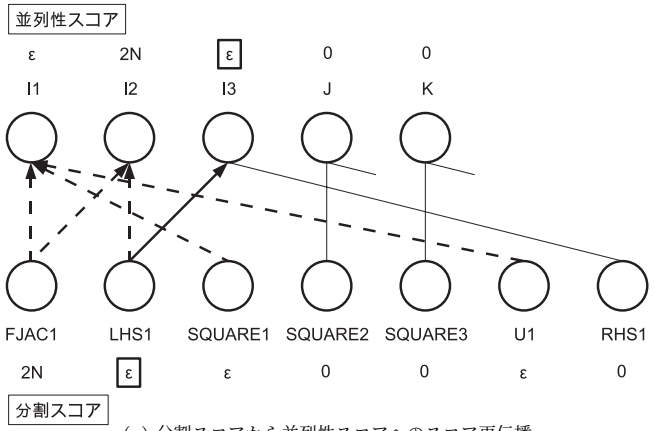


(a) 分割スコアから並列性スコアへのスコア伝播

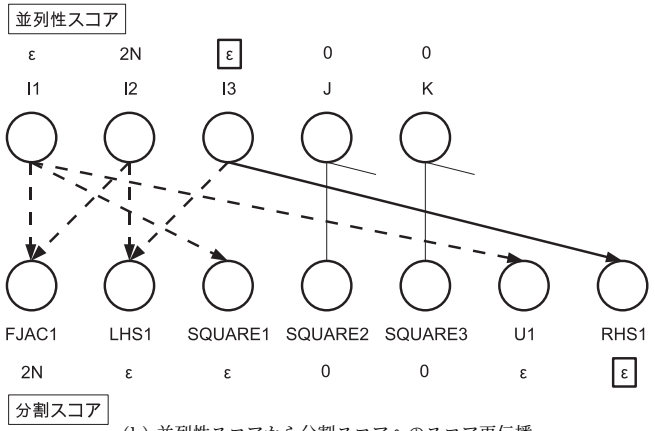


(b) 並列性スコアから分割スコアへのスコア伝播

図 5　NPB BT x_solve のスコア伝播
Fig. 5　Propagation of scores for NPB BT x_solve subroutine.



(a) 分割スコアから並列性スコアへのスコア再伝播



(b) 並列性スコアから分割スコアへのスコア再伝播

図 6　NPB BT x_solve のスコア再伝播
Fig. 6　Repropagation of scores for NPB BT x_solve subroutine.

される．

次に，図 5 (b) に示すように並列性スコアから分割スコアへの伝播が起こる．並列性スコアが 0 でないのは I1 と I2 である．I1 に接続されているノード FJAC1，SQUARE1 と U1 のうち，FJAC1 はすでに 0 ではない分割スコアが設定されているので更新されないが，他の SQUARE1 と U1 は分割スコアが ϵ に更新される．同様に I2 に接続されているノードのうち，LHS1 の分割スコアが 0 から ϵ に更新される．

さらに，**図 6** に示すように，分割スコアから並列性スコアへのスコアの伝播と，並列性スコアから分割スコアへのスコアの伝播が繰り返され，I3 と RHS1 のノードのスコアが ϵ に更新される．これ以上は，スコアの伝播を試みても新たにスコアが ϵ へ更新されるノードはない．

最終的に，図 6 (b) に示すように，J と K のループの並列性スコアや，配列 U，SQUARE，RHS の 2，3 次元目の分割スコアは 0 のままであり，これらのループや配列が並列化，分割に適しているということになる．

3.1.5　スコア伝播アルゴリズムの計算量

スコアの伝播は，ϵ に新たに更新される並列性スコアまたは分割スコアがある限り繰り返される．最悪の場合，全配列の次元が分割不可とされ，全ループが並列化が望ましくないと判断されて停止する．この伝播アルゴリズムの 1 回の繰返しで配列の次元 1 つずつが分割不可とされるか，ループが 1 つずつ並列化が望ましくないと判断されるとすると，最悪の場合で，全配列の次元の総和と，全ループ数の和の回数だけ繰り返されてアルゴリズムは停止する．

現実のコードについては，NPB3.2 の逐次版である NPB3.2-SER の BT では並列性，分割スコアの初期化の後，ループから配列への伝播，配列からループへの伝播を 2 回ずつ行い，それ以上の変更を行わずにアルゴリズムは停止し，実用上は問題のない時間内に解析可能であることを確認している．

3.1.6　分割次元の決定

ここまでで，すべてのループの並列性スコアとすべての配列の次元の分割スコアが確定したことになる．これらのスコアをもとに，手続き内で分割すべき配列の次元と，その分割に対応するループの並列化を以下のように決定する．

(1) ループを並列性スコアの小さい順にソートする．並列性スコアが最も小さいループを並列化の候補とする．この時点では並列化の候補は複数存在しうる．

(2) アクセス関数が表すアフィン関係に基づき，並列化候補のループが添字に表れる配列の次元を求める．並列化の候補となるループが 1 つの場合は終了する．

(3) 並列化の候補となるループが複数存在する場合は，Fortran が列優先（column major）であることを考慮し，手順 (2) で求めた次元が高い配列が多く含まれるようなループを選択する．

図 1 の BT の例では，K，J の両ループが並列性スコア 0 で並列化の候補となる．ループ K を並列化すると配列 U，SQUARE，RHS の 3 次元目が分割対象となるが，ループ J を並列化するとそれらの配列の 2 次元目が分割対象となり，次元の高い要素で分割することになるループ K が選択される．

列優先であることを考慮すると，高い次元で分割した方が，配列の連続要素が各プロセスに分割配置されやすくなる．配列の再分割がおこる場合にも，連続要素が多い方がプロセス間のデータ転送時のメッセージのパッキング，アンパッキングに要するオーバヘッドが小さくなる．そのため，上記の手順 (3) では高い次元が含まれるようにループが選択される．これについては，4 章でも並列プログラムを実行して検証する．

3.2　手続き間解析

手続き内解析で決定された配列の分割すべき次元をもとに，手続き間の解析を行う．この解析は，計算量の多い手続きの分割を優先するというヒューリスティクスを用いて解析に要する計算量を小さくしている．まず，すべての手続きの計算量を見積もり，これを手続き p の計算量 C_p とする．C_p の大きい手続きから順に，配列分割を他の手続きに伝播させる処理を行う．なお，本アルゴリズムではデータ並列性の高い Fortran プログラムを仮定しており，このようなプログラムでは COMMON ブロック内の大規模な配列を用いることが多いことから，本アルゴリズムでは COMMON ブロック内の配列にも対応している．

Callee への伝播

手続き p から呼び出される手続き callee p1 を考える．

- p で宣言され，p1 で宣言されていない COMMON 配列は，p の分割と同じ COMMON 配列として p1 でも宣言する．
- p で宣言され，p1 で宣言されていない配列については，COMMON 配列でなければ，p1 では当該配列が必要とされていないことになるので，特に何の処理も行わない．
- p で宣言され，p1 でも宣言されている配列（p から p1 へ引数として渡されている配列）がある場合，その分割形状を比較する．分割が同じであれば，再分割などの必要はない．分割が異なれば，p から p1 を呼び出す前後に再分割のコードを挿入する．

Caller への伝播

手続き p を呼び出す手続き caller p2 を考える．

- p で宣言され，p2 で宣言されていない COMMON 配列は，p の分割と同じ COMMON 配列として p2 でも宣言する．
- p で宣言され，p2 で宣言されていない配列については，COMMON 配列でなければ，p2 では当該配列が必要とされていないことになるので，特に何の処理も行わない．
- p で宣言され，p2 でも宣言されている配列（p2 から p へ引数として渡されている配列）がある場合，その分割形状を比較する．分割が同じであれば，再分割などの必要はない．分割が異なれば，p2 から p を呼び出す前後に再分割のコードを挿入する．

このアルゴリズムの問題点として，callee 内で分割された配列が使用されるが定義されない場合，callee から戻ってきた時点での再分割による書き戻しは不要であるため，その再分

割はオーバヘッドとなることがあげられる．同様に，callee を呼び出した時点での配列の各要素の値が callee 内でまったく使用されない場合は，callee を呼び出す直前の再分割は不要であるため，オーバヘッドとなる．これらの再分割を削除するためにはデータフロー解析が必要であるが，現時点では実装していない．

4. 性能評価

前章で手続き内と手続き間に分けて，配列の自動分割のための解析手法について述べてきた．この章では，本手法を実装したコンパイラと，そのコンパイラが生成したプログラムの性能を評価した結果について述べる．

4.1 コンパイラの実装

本コンパイラは，Fortran プログラムを入力とし，データ分割の指示文が挿入された HPF プログラムを生成する．コンパイラ全体は C++言語約 34,000 行で実装されており，そのうち本論文で提案しているデータ分割手法を実装している部分は約 3,000 行である．

データ分割解析を実行するには，各ループが並列実行可能であるかどうか，並列実行可能である場合も，DO ALL 型の並列実行が可能であるかどうかが判定されていなくてはならない．本コンパイラでは，DO ALL 型の並列ループをデータ依存解析とデータフロー解析によって判定するとともに，解析が困難なループについては HPF の指示文に類似の形式でループの並列性を指定することも可能としている．

提案する手法には手続き間の解析も含まれているため，解析対象にしたい手続きのソースコードは，すべてコンパイラへの入力として与える．各手続き内では，配列の再分割は起こらず，同一の分割で実行され，配列の再分割は手続き呼び出しの前後でのみ行われるものとする．

出力とする HPF でのプロセッサ配列は 1 次元とし，配列も複数の次元のうちの 1 つの次元のみの分割とする．これは，複数次元のプロセッサ配列に対応すると，解析のアルゴリズムが複雑になり，解析に要する計算量も増大すること，および，現状の HPF コンパイラが生成するコードでは，1 次元分割で良好な性能が期待できることがその理由である．

4.2 評価環境

HPF プログラムの実行には，**表 1** に示す 8 ノード 16CPU 構成の PC クラスタを用いた．HPF プログラムは PGI 社のコンパイラによって MPI の実行形式プログラムに変換される．MPI プログラムの実行には，SCore[22] が提供する Myrinet-2000 の MPI ライブラリを利用した．

表 1　PC クラスタの仕様
Table 1　Specification of PC cluster.

CPU	
CPU	Intel Xeon 2.8 GHz
L2 Cache	512 KB
ノード	
CPU 数	2
主記憶	1 GB
Network	Myrinet-2000 双方向 2.0 Gbps
OS	Fedora Core 3
Compiler	PGI HPF 7.2
SCore	5.8.3
ノード数	8

評価に用いたプログラムは，NAS ParallelBenchmarks（NPB）3.2-SER（逐次）と 3.0-HPF の BT アプリケーションである．NPB ではアプリケーションごとに問題サイズが小さい方から順に Class S，W，A，B，C，D などが用意されているが，今回はそのうち Class W と Class A を用いた．

4.3 スケーラビリティの評価

我々の実装したコンパイラの配列データ分割解析では，現時点では，サブルーチン呼び出しの際の部分配列を扱えない．そのため，一部のサブルーチンは手動でインライン展開して解析している．具体的には，NPB3.2-SER BT の matvec_sub，matmul_sub，binvcrhs，binvrhs などのサブルーチンを，呼び出し側の x_solve，y_solve，z_solve などのサブルーチンにインライン展開している．

このインライン化されたプログラムが入力として与えられ，我々の実装したコンパイラによって配列のデータ分割指示文が挿入された HPF コードが生成され，さらに PGI HPF コンパイラによって MPI の実行形式が出力される．Class W と Class A を PC クラスタの 16 台までの CPU を用いて実行したところ**表 2** の auto_dist W と auto_dist A の行の結果が得られた．比較のため，3.0-HPF を実行したところ，表 2 の hpf W と hpf A の行の結果が得られた．さらに，それらの結果のスケーラビリティを求めると**図 7** と**図 8** のようになった．

なお，逐次版 3.2-SER BT のインライン展開により，サブルーチン呼び出しのオーバヘッドが小さくなり，オリジナルの SER より実行が高速化される．3.2-SER BT Class W を実行すると 8.10 秒であるのに対し，インライン化すると 7.90 秒となる．同様に Class A を実

表 2　NPB3.2-SER BT を並列化した実行結果（秒）
Table 2　Execution time of NPB3.2-SER BT (Time in second).

	1	2	4	8	16
auto_dist W	11.190	7.375	4.986	3.460	3.443
hpf W	19.713	11.356	6.865	4.240	4.170
auto_dist A	227.219	157.183	94.066	58.419	36.172
hpf A	413.164	225.281	126.920	71.188	49.974

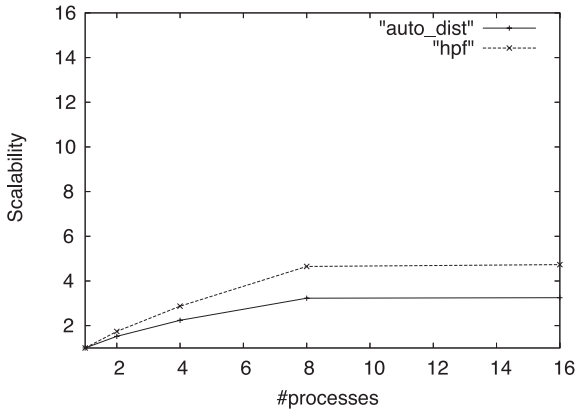


図 7　NPB BT Class W のスケーラビリティ
Fig. 7　Scalability of NPB BT Class W.

行すると 183.85 秒であるのに対し，インライン化すると 180.32 秒となる．3.0-HPF では，これらのサブルーチンはインライン化されていないため，実行時間はオリジナルの SER とほぼ同じである．そのため，表 2 の実行時間は異なる逐次時間のプログラムを並列化した結果となっていることに注意されたい．

これらの結果のスケーラビリティを求めた図 7 と図 8 では，auto_dist と hpf の逐次実行時間が異なることもあり，それぞれの並列プログラムの 1 プロセスでの実行時間を 1 として求めた．図 7 と図 8 から分かるように，提案手法によって自動的にデータ分割が求められた場合にも，オリジナルの HPF と同様のスケーラビリティが得られている．オリジナルの HPF の方が，データ分割だけでなく各ループの並列性と，そのループにおける私有変数（private variables）の指定が指示文（INDEPENDENT 文）によって細かく指定されて

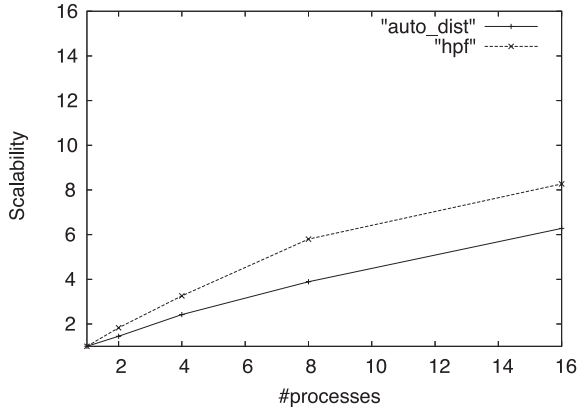


図 8　NPB BT Class A のスケーラビリティ
Fig. 8　Scalability of NPB BT Class A.

いるために，ややスケーラビリティが向上していると思われる．Class W と Class A では，データサイズが Class A の方が大きいため，問題が持つ並列性が高く，スケーラビリティも良好であると推察できる．

4.4　選択された分割の評価

本節では，提案手法によって自動的に選択された配列のデータ分割が最適となっているかどうかを確認するため，他のデータ分割の指示文を手動で挿入した場合の実行結果との比較を行う．

NPB3.2-SER BT では，カーネルループで以下のようなサブルーチンが繰り返し呼び出される．

```
DO STEP=1,NITER
  ...
  CALL ADI
  ...
ENDDO

SUBROUTINE ADI
  CALL COMPUTE_RHS
```

表 3　NPB BT の各サブルーチンにおける分割スコア
Table 3　Distribution scores in subroutines in NPB BT.

	X(I)	Y(J)	Z(K)
COMPUTE_RHS	ϵ	ϵ	ϵ
X_SOLVE	+	0	0
Y_SOLVE	0	+	0
Z_SOLVE	0	0	+
ADD	0	0	0

表 4　NPB3.2-SER BT Class W の実行結果（秒）：配列分割による影響
Table 4　Execution time of parallelized NPB3.2-SER BT Class W for several distribution patterns (time in second).

	1	2	4	8	16
ZZZYZ	11.190	7.375	4.986	3.460	3.443
XYXXX	9.956	18.004	19.223	17.818	18.800
AZZYZ	9.666	7.916	7.733	10.073	19.569

```
  CALL X_SOLVE
  CALL Y_SOLVE
  CALL Z_SOLVE
  CALL ADD
END
```

BT で用いられる主要な配列は X，Y，Z の 3 次元空間のデータを表す．それぞれのサブルーチンでは，並列性やプロセス間通信の存在により，X，Y，Z のいずれかの次元の分割によって並列性スコアや分割スコアが正の値をとる．これをまとめたものが**表 3** であり，主要な配列の X，Y，Z のどの次元を分割するかによって，配列の次元の分割スコアと，対応するループ K，J，I の並列性スコアが ϵ またはそれ以上の正の値（+）になるかどうかを示している．

表 3 から分かるように，すべてのサブルーチンで分割スコアが 0 になるような最適なデータ分割は存在しない．そのため，我々の提案するアルゴリズムでの解析の結果，最もペナルティが少なくなるように，サブルーチン Z_SOLVE では Y の次元で配列が分割され，それ以外のサブルーチンでは Z の次元で配列が分割される．これを 5 つのサブルーチンの呼び出しの順に ZZZYZ と呼ぶことにする．

比較のため，サブルーチン X_SOLVE のみ Y の次元で分割し，他のサブルーチンでは X の次元で分割する XYXXX や，COMPUTE_RHS ではどの次元でも分割が適当ではないとして重複分割し，他のサブルーチンでは ZZZYZ と同じ分割とした AZZYZ の 2 種類の分割を，手動で指示文を挿入することで作成し実行した．その実行結果が**表 4** である．

表 5　NPB3.2-SER BT Class W を ZZZYZ 分割で並列化した実行結果（秒）
Table 5　Execution time of parallelized NPB3.2-SER BT Class W with ZZZYZ distribution (time in second).

	1	2	4	8	16
Total	11.190	7.375	4.986	3.460	3.443
compute_rhs	3.880	2.589	2.043	1.507	1.506
x_solve	1.968	0.990	0.538	0.277	0.191
y_solve	2.203	1.095	0.571	0.283	0.189
z_solve	2.911	2.621	2.104	1.709	3.344
adi	0.008	0.007	0.006	0.005	0.006
add	0.135	0.065	0.032	0.014	0.011

表 6　NPB3.2-SER BT Class W を XYXXX 分割で並列化した実行結果（秒）
Table 6　Execution time of parallelized NPB3.2-SER BT Class W with XYXXX distribution (time in second).

	1	2	4	8	16
Total	9.956	18.004	19.223	17.818	18.800
compute_rhs	2.306	12.447	15.771	15.223	15.688
x_solve	2.837	3.246	2.858	2.484	2.982
y_solve	2.229	1.240	0.733	0.428	0.237
z_solve	2.277	1.246	0.723	0.412	0.229
adi	0.007	0.007	0.007	0.006	0.008
add	0.171	0.094	0.044	0.022	0.011

我々の提案手法により選択された ZZZYZ では，1 プロセスでの実行時間は長いが，それ以外は最も実行時間が短く，プロセス数を増加させるとスケーラビリティが得られている．これに対し，他の分割ではスケーラビリティが得られていない．

さらに，再分割に要する時間などを見積もるために主要サブルーチンの実行に要する累積実行時間を計測した．その結果を**表 5**，**表 6**，**表 7** に示す．

ZZZYZ 分割では，z_solve サブルーチンの前後で配列の再分割が行われているため，ほぼ同じ計算量の x_solve や y_solve のサブルーチンよりも累積実行時間が長い．同様に XYXXX 分割では x_solve サブルーチンの前後で配列の再分割が行われていることが累積実行時間から確認できるが，それだけでなく，compute_rhs，x_solve，y_solve，z_solve などの累積実行時間全体も ZZZYZ 分割よりも増加している．このことから，3.1.6 項で述べたように，

表 7　NPB3.2-SER BT Class W を AZZYZ 分割で並列化した実行結果（秒）
Table 7　Execution time of parallelized NPB3.2-SER BT Class W with AZZYZ distribution (time in second).

	1	2	4	8	16
Total	9.666	7.916	7.733	10.073	19.569
compute_rhs	2.374	3.123	4.801	8.143	19.494
x_solve	1.963	0.994	0.544	0.281	0.206
y_solve	2.209	1.101	0.574	0.282	0.195
z_solve	2.888	2.630	2.203	1.961	2.246
adi	0.008	0.007	0.007	0.006	0.008
add	0.137	0.062	0.032	0.014	0.010

Fortran が列優先であるために，より高い次元で分割すべきであるという決定手法の有効性が示されたことになる．

AZZYZ 分割では配列の再分割を行っている compute_rhs，z_solve の実行時間が増大している．compute_rhs ではループの並列性スコアが 0 のループで並列化することができず，どのような分割を行ってもプロセス間の通信が発生する．このプロセス間通信を削除するために，AZZYZ 分割では，サブルーチンの呼び出しの前後にすべての配列の要素を転送して各プロセスで重複分散させている．しかし，ZZZYZ 分割の実行時間と比較すると，このような重複分散は高速化には結び付いていないことが分かる．よって，並列性スコアや分割スコアに ϵ を導入することで ZZZYZ のような適切な分割が選択されていることが確認できる．

5. おわりに

本論文では，大規模な分散メモリ型計算機向けにプログラムを並列化する際に，配列の次元のデータ分割を自動的に決定する手法を提案した．本手法では，ループの並列性と配列の次元の分割を統一的に扱うためにアクセス関数を用いて配列整合解析を行っている．

本手法を用いて NPB 3.2-SER BT ベンチマークのデータ分割を解析して HPF ディレクティブが挿入された Fortran プログラムを生成することが可能となった．生成された HPF プログラムを商用の HPF コンパイラを用いて並列化して PC クラスタで実行したところ，NPB3.0-HPF と同様の台数効果が得られた．また，Fortran が列優先であることを考慮して分割次元の選択を行うことが有効であることを，他の次元を分割した場合と比較することによって明らかにした．

本論文で提案したデータ分割の機能は，コンパイラによる自動並列化を目指す場合に重要な位置を占める．バックエンドである HPF コンパイラの技術の進展と，本論文での提案手法の改良により，データ並列型プログラムに対する自動並列化が進むことが期待できる．

今後は，より多くのプログラムでのデータ分割を自動化を可能とするよう提案手法を改良するとともに，逐次のプログラムを OpenMP に変換し，それをさらに分散メモリ型計算機向けに並列化することを可能とするよう，提案手法を応用することを検討したい．

参 考 文 献

1) Message Passing Interface Forum: MPI: A Message-Passing Interface Standard (Version 1.1), Technical report (1995). http://www.mpi-forum.org
2) Message Passing Interface Forum: MPI-2: Extensions to the Message-Passing Interface, Technical report (1997). http://www.mpi-forum.org
3) High Performance Fortran Forum: High Performance Fortran Language Specification (Ver. 1.0), Technical Report CRPC–TR92225, Rice University (1993).
4) High Performance Fortran Forum: High Performance Fortran Language Specification (Ver. 2.0), Technical report, Rice University (1997).
5) High Performance Fortran Forum: High Performance Fortran 2.0 公式マニュアル，シュプリンガー・フェアラーク東京 (1999).
6) 岩下英俊，青木正樹：HPF トランスレータ fhpf における分散種別を一般化したコード生成手法，情報処理学会論文誌：コンピューティングシステム，Vol.47, No.SIG12(ACS 15), pp.329–339 (2006).
7) 村井　均，岡部寿男：地球シミュレータ上の HPF による NAS Parallel Benchmarks の実装と評価，*SACSIS2004* (2004).
8) OpenMP Architecture Review Board: OpenMP Application Program Interface, Version 3.0, Technical report (2008). http://www.openmp.org
9) Numrich, R.W. and Reid, J.: Co-Array Fortran for parallel programming, *ACM SIGPLAN Fortran Forum*, Vol.17, No.2, pp.1–31 (1998).
10) Charles, P., Donawa, C., Ebcioglu, K., Grothoff, C., Kielstra, A., von Praun, C., Saraswat, V. and Sarkar, V.: X10: An Object Oriented Approach to Non-Uniform Cluster Computing, *OOPSLA'05: Proc. 20th Annual ACM SIGPLAN Conf. Object Oriented Programming, Systems, Languages and Applications*, pp.519–538 (2005).
11) Cray Inc.: The Chapel Language Specification Version 0.4, Technical report (2005).
12) Allen, E., Chase, D., Hallett, J., Luchangco, V., Maessen, J.-W., Ryu, S., Steele Jr., G.L. and Tobin-Hochstadt, S.: The Fortress Language Specification Version 1.0, Technical report, Sun Microsystems, Inc. (2008).
13) Wolfe, M.: *High Performance Compilers for Parallel Computing*, Addison-Wesley (1995).
14) Li, J. and Chen, M.: Index Domain alignment: Minimizing cost of cross-referencing

between distributed arrays, *The 3rd Symposium on the Frontiers of Massively Parallel Computation*, Jaja, J. (Ed.), pp.424–433, IEEE Computer Society Press (1990).
15) Gupta, M. and Banerjee, P.: PARADIGM: A Compiler for Automatic Data Distribution on Multicomputers, *Int'l Conf. on Supercomputing*, Tokyo, Japan, pp.87–96, ACM (1993).
16) 辰巳尚吾，窪田昌史，五島正裕，森眞一郎，中島　浩，富田眞治：並列化コンパイラ TINPAR における自動データ分割部の実現，情報処理学会研究報告 96-PRO-8-5，情報処理学会，*SWoPP'96* (1996).
17) Kennedy, K. and Kremer, U.: Automatic Data Layout for High Performance Fortran, *Proc. Supercomputing* (1995).
18) Kennedy, K. and Kremer, U.: Automatic data layout for distributed-memory machines, *ACM Trans. Prog. Lang. Syst.*, Vol.20, No.4, pp.869–916 (1998).
19) 松浦健一郎，村井　均，末広謙二，妹尾義樹：データ並列プログラムに対する高速な自動データ分割手法，情報処理学会論文誌，Vol.41, No.5, pp.1420–1429 (2000).
20) 廣岡孝志，太田　寛，菊地純男：ファーストタッチ制御による分散共有メモリ向け自動データ分散方式，情報処理学会論文誌，Vol.41, No.5, pp.1430–1438 (2000).
21) 廣岡孝志，太田　寛：分散共有メモリ向け手続き間自動データ分散方法の実装と評価，情報処理学会論文誌，Vol.42, No.4, pp.898–909 (2001).
22) SCore. http://www.pccluter.org

(平成 21 年 7 月 10 日受付)
(平成 21 年 10 月 5 日採録)

窪田　昌史（正会員）

1993 年京都大学大学院工学研究科情報工学専攻修士課程修了．同年日本 IBM（株）入社．1998 年京都大学大学院工学研究科情報工学専攻博士後期課程単位認定退学．同年広島市立大学情報科学部助手．現在，同助教．主として並列化コンパイラ，ハイパフォーマンスコンピューティングに関する研究に従事．IEEE-CS，ACM 各会員．

北村　俊明（正会員）

1955 年生．1978 年京都大学工学部情報工学科卒業．1983 年同大学大学院博士課程研究指導認定退学．同年富士通（株）入社．汎用コンピュータ，スーパコンピュータ VPP シリーズの VLIW 型 CPU，M アーキテクチャ・命令エミュレーション，米国 HAL 社において SPARC プロセッサ等の研究開発に従事．博士（工学）．2000 年京都大学総合情報メディアセンター助教授を経て，2002 年広島市立大学情報科学部教授．電子情報通信学会，IEEE，ACM 各会員．