# 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

松浦 健一郎[†] 村井 均[†]
末広 謙二[†] 妹尾 義樹[†]

データ分割は分散メモリ型並列計算機向けにプログラムを並列化する際の重要課題である．本稿では，Fortranプログラムにおいて自動的にデータ分割を行う手法を提案する．本手法の利用によりユーザは容易にFortranプログラムを並列化できる．本手法の特徴は，配列アクセス情報を基にループを効率的に並列化するデータ分割の候補を作成し，コントロールフローグラフを基に通信オーバヘッドを抑制するデータ分割を選択することによって，プログラム全体として良好な並列実行性能を達成するためのデータ分割を高速に決定することである．本手法は短時間で複数手続き間にわたるデータ分割を行えるので，高速性を活かした対話的なチューニング作業が可能である．今回本手法を実装し，Fortranプログラムを並列化して，実行時間を計測した．その結果，自動データ分割に要する時間が短いにもかかわらず，同等のMPIプログラムに近い実行速度と台数効果が得られた．

# Fast Automatic Data Layout for Data Parallel Programs

Ken-ichiro Matsuura,[†] Hitoshi Murai,[†] Kenji Suehiro[†]
and Yoshiki Seo[†]

Determining optimal data layout is very important for parallelizing programs on distributed-memory parallel computers. This paper describes a new algorithm of automatic data layout of Fortran programs. The algorithm enables users to parallelize Fortran programs without difficulty. It generates candidates of data layout for efficient parallelization of loops from access patterns, selects data layout to reduce communication overheads, and determines data layout all over the program to achieve good parallel execution performance. It can quickly determine data layout over multiple subroutines, thus it enables interactive tuning cooperating with users. It has been implemented, and evaluated by parallelizing several Fortran benchmark programs. Execution time and scalability of the benchmarks has been close to those of MPI alternatives.

## 1. はじめに

データ分割は，分散メモリ型並列計算機向けにプログラムを並列化する際の重要課題である．分散メモリ型並列計算機では，あるプロセッサが必要とするデータがローカルメモリに存在しない場合，リモートプロセッサとの間でデータ転送が必要である．データ転送は並列プログラムの性能低下の主要因となるため，実行時に発生するデータ転送ができるだけ少なくなるようにデータを分割しプロセッサへ配置することが重要である．

データ分割に関して，High Performance Fortran [1)]，FortranD [2)]，Vienna Fortran [3)]などのデータ並列言語では，データ分割を指示文を用いて記述することにより，Fortranプログラムを並列化することが可能である．このような言語においてデータ分割を決定するのはユーザの仕事であるが，効果的な並列化を実現できるデータ分割を決定するには知識や経験が必要となる．また多くの配列に対してデータ分割を決定するのは負担が大きく，誤りも起こりやすい．したがって，より容易な並列化を可能にするためには，データ分割の決定を自動化することが望ましい．

本稿では，Fortranプログラムを並列化するために，短時間で自動的にデータ分割を行う手法について述べる．本手法は，配列アクセスのパターンを解析してプログラム中の各ループを並列化するために適したデータ分割の候補を生成した後に，手続きごとにコントロールフローグラフを用いて複数ループ間でデータの再分割を少なくするようなデータ分割の組合せを候補の中から選択する．そして手続き間に関しては，手続き呼び出し時の再分割を少なくするよう呼び出し元と

† 新情報処理開発機構並列分散システムNEC研究室
Parallel and Distributed Systems NEC Laboratory, RWCP

被呼び出し手続きのデータ分割を調整する．このように本手法では，各ループを並列化するとともに，手続き内と手続き間におけるデータの再分割を抑制することにより通信オーバヘッドを軽減して，プログラムを効率良く並列実行するためのデータ分割を決定する．

本手法では，最初に実行時間が長いループを 1 つ選んで基点とし，そのループの並列化に適したデータ分割を定める．続いて，データ分割の整合性を考慮しつつ，前後のループのデータ分割を順番に決定していく．このアプローチにより，実行時間が長いループを優先的に並列化して対象プログラムの実行性能を向上しつつ，高速にプログラム全体のデータ分割を決定できる．

ユーザがいくつかの配列に対して明示的にデータ分割を指示した場合，本手法はユーザが指示したデータ分割をプログラムの各所に反映させる．したがって，ユーザと本手法との協調作業によるプログラムの並列化やチューニングが可能である．本手法の高速性は，このような作業をスムーズに進めるうえで役立つ．またユーザがデータ分割を指示する際には，本手法に並行して開発中のグラフィカルなユーザインタフェースが利用できる．

今回，本手法を実装して，いくつかのベンチマークプログラムに関して自動的にデータ分割を決定し，並列実行することによって評価を行った．そして，データ分割を短時間で決定できることと，自動データ分割を行ったプログラムが効率良く並列実行できることを確認した．

従来，プログラムを解析して処理系が自動的にデータ分割を決定する手法については様々な研究が行われてきた．最適なデータ分割を求める問題は，NP 完全であることが知られている[4]．Diderich ら[5]，辰巳ら[6]，Philippsen[7]，Sheffler ら[8]，Anderson ら[9]は，それぞれ，データ分割問題を定式化し，近似解を求めるヒューリスティクスを示した．本稿で述べる手法もヒューリスティクスである点においてはこれらの手法と同じである．しかし，文献 5)，6) は単独のループ，文献 7)，8)，9) は単独の手続きに限定してデータ分割を決定する手法である．本稿の手法は複数のループ間，複数の手続き間にわたってデータ分割を決定できる．

一方，Bixby らは，データ分割問題を 0-1 整数計画問題として定式化できることを示した[10]．そして，Kennedy らは，D システム[11]に，0-1 整数計画問題を解くことによって自動的に最適なデータ分割を行う機能を実装し，その評価を行った[12]．この手法はデータ分割の正確性において優れているが，複数手続きにわたるデータ分割の決定には未対応であり，また処理に要する時間が長い．本稿の手法は複数の手続きにわたって短時間でデータ分割を決定でき，ゆえにユーザとの対話的な協調作業が可能であることが利点である．

以下，2 章では本手法の詳細を述べ，3 章では本手法の実装例の評価を示す．4 章において結論を述べる．

## 2. 自動データ分割手法

### 2.1 本手法の手順

本手法は，Fortran プログラムを並列化するためのデータ分割を，プログラム全体にわたって自動的に決定するものである．大まかな流れは次のとおりである．

(1) コントロールフローグラフ（CFG）の生成
手続きごとに CFG を作成する．作成した CFG は (3) で用いられる．

(2) 並列ループのデータ分割候補の生成
各々のループごとに独立した解析を行って，当該ループを並列化するために適したデータ分割の候補を生成する．複数ループ間におけるデータ分割の不整合については (2) では考慮せず，(3) において解決する．

(3) 手続き内における分割候補の選択
各手続きに対して，ループを効率良く並列化し，通信オーバヘッドを少なくするようなデータ分割を決定する．具体的には，(1) で作成したグラフを用いつつ，(2) で作成したデータ分割の候補の中から，複数ループ間での再分割がなるべく起こらないようなデータ分割を選択する．

(4) 手続き間における分割候補の選択
手続きの呼び出し関係を基に，手続き呼び出し時の通信オーバヘッドを少なくするようなデータ分割を決定する．具体的には，各手続きに関して，すべての呼び出し元におけるデータ分割を調べ，最も多くの呼び出し元に共通するデータ分割を選択する．

以下の各節で上記の各項目の詳細を述べる．

### 2.2 コントロールフローグラフの生成

本節では，手続きごとに作成するコントロールフローグラフ（CFG）について述べる．本手法における CFG とは，並列ループノード，CALL ノード，およびノード間の制御の流れを表すエッジから構成される巡回有向グラフであるとする．

並列ループとは，タイトであり，いずれかのネストレベルで並列化可能であり，かつ外側に並列化可能なループが存在しないような多重または一重ループのことである．並列ループノードとは CFG において並列ループに対応するノードである．並列ループの条件を満たさないループに関しては CFG のノードとせ

ず，データ分割を行わない．また，CALL ノードとは CALL 文に対応するノードである．

本手法では CFG 上の各並列ループノードに関して優先度を求める．並列ループノードの実行回数 × ループの繰返し回数 × ループ内での全配列のアクセス回数，をノードの優先度とする．並列ループノードの実行回数とは，プログラムの実行時にそのノードが実行される回数である．また，CFG 上の各エッジに関しても優先度を求める．エッジの優先度とは，プログラムの実行時にそのエッジに沿って制御が流れる回数である．

ノードやエッジの優先度は 2.5 節で述べるデータ分割の選択処理においてノードやエッジの処理順序を決定するために用いる．並列ループノードの優先度はそのループにおいてアクセスするデータの総量を近似している．本手法では，優先度が高いループすなわちデータアクセス量が多いループは実行時間が長く，またこのようなループを実行する際にデータの再分割を行えば通信オーバヘッドが大きいと見なして，優先度が高いループから順にデータ分割を決める．また，優先度が高いエッジから順に処理することによって，数多く実行されるエッジにおいて再分割が行われないようにする．

CFG の例を**図 1** に示す．図の上部のプログラムにおける各番号が図の下部の CFG の各ノードの番号に対応する．ノード番号 1，2，4，5，6 は並列ループノード，3 は CALL ノードである．各ノードに付随する数字はノードの優先度である．各エッジに付随する数字はエッジの優先度である．

### 2.3 並列ループのデータ分割候補の生成

本節では，各並列ループごとに独立に解析を行って，当該ループを並列化するために適したデータ分割を生成する手順について述べる．ここでは複数ループ間におけるデータ分割の不整合は考慮せずに，ループごとにデータ分割の候補（以下，分割候補）を生成する．そして 2.4 節以降で述べる手順において，複数ループの分割候補の間の整合性を考慮しつつ最終的なデータ分割を候補の中から選択する．

分割候補を生成するために，並列ループ内の配列アクセスの添字式を基に，配列間の整列関係を表すグラフ（以下，整列グラフ）を作成する．本手法では添字式が並列ループの制御変数の線形式であり，かつ 1 つの制御変数が 1 つの次元にのみ現れる場合を考える．この条件を満たさない添字式をともなう配列に関しては，全プロセッサへの複製を分割候補とする．なお，添字式において並列ループの制御変数に掛かる係数をストライドと呼ぶ．

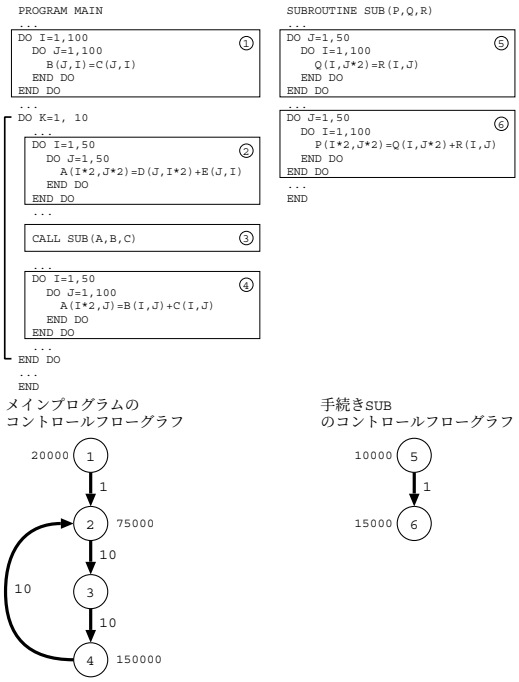


**図 1** コントロールフローグラフ（CFG）
Fig. 1 Control flow graph (CFG).

整列グラフのノードは，各配列ごとの同一のストライドのアクセスを示す．また，ノードに対応する配列アクセスの個数をノードの重みとする．整列グラフのエッジは，エッジの両端のノードのアクセスの間に整列関係が定義できることを示す．ノード A の配列のストライドがノード B の配列のストライドの整数倍になっているとき，ノード B からノード A へのエッジを生成する．同一の配列に対するノードの間にはエッジを生成しない．また，エッジのソースのノードの重みをそのエッジの重みとする．なお，整列グラフは非巡回有向グラフである．

整列グラフを作成した後，以下のように分割候補を定めることによって，当該ループにおける主要な整列関係を分割候補へと反映させる．「ノードの重み」に「そのノードに入るエッジの重み」を加えた値が最大のノード，すなわち最も多くの配列の整列先とすることが可能なノードを整列ターゲットとする．そして制御変数を含む次元での BLOCK 分散を整列ターゲットの分割候補とする．整列ターゲット以外のノードで，整列ターゲットへのエッジがあるノードについては，整列ターゲットに対する整列を分割候補とする．整列ターゲットへのエッジがないノードについては，全プロセッサへの複製を分割候補とする．

ノードの分割候補を，各ノードに対応する配列の分

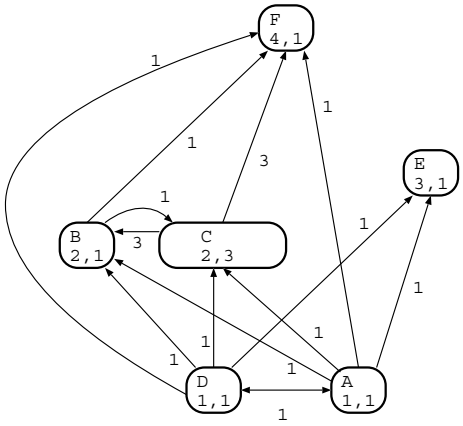


**図 2** 整列グラフ
Fig. 2 Alignment graph.

割候補とする．ある配列が複数の異なるストライドでアクセスされているために対応するノードが複数個あるならば，そのうち最大の重みを持つノードの分割候補を配列の分割候補とする．また，アクセスされる配列が多次元配列である場合，並列ループの制御変数が現れる次元以外は全プロセッサへの複製とする．

以下の並列ループに関して，I ループに対する整列グラフは**図 2** になる．図中の上方のノードで，ノード内の F 4, 1 という表記は，変数名が F，ストライドが 4，ノードの重みが 1 であることを示す．他のノードに関しても同様である．また，エッジに付随する数値はエッジの重みである．

```
DO I=1, 100
  DO J=1, 100
    A(I+1,2*J) = B(2*I,J-1)
               + C(2*J+1,2*I)
               + C(2*J-1,2*I+1)
    D(I-2,4*J+1) = C(2*J,2*I-1)
                 + E(3*I,J+2)
                 + F(4*I+2,J)
  END DO
END DO
```

図 2 を用いて生成した分割候補を HPF の DISTRIBUTE 指示文と ALIGN 指示文を用いて表現すると以下のようになる．

```
!HPF$ DISTRIBUTE F(BLOCK,*)
!HPF$ ALIGN A(I,*) WITH F(4*I,*)
!HPF$ ALIGN B(I,*) WITH F(2*I,*)
!HPF$ ALIGN C(*,I) WITH F(2*I,*)
!HPF$ ALIGN D(I,*) WITH F(4*I,*)
!HPF$ ALIGN E(*,*) WITH F(*,*)
```

上の分割候補を「配列名. 分割次元. 整列ターゲットへの整列幅」という形式で表現すると以下のようになる．整列ターゲットに関しては「配列名. 分割次元.1」，全プロセッサへの複製に関しては「配列名.0.0」と表現する．また「配列名」は整列ターゲットを示す．

A.1.4, B.1.2, C.2.2, D.1.4, E.0.0, F.1.1

多重ループに関しては各々のネストレベルに対して分割候補を作成する．I ループと同様に，J ループに対しては以下の分割候補が得られる．

A.2.2, B.2.4, C.1.2, D.2.1, E.2.4, F.2.4

### 2.4 共通分割候補

2.3 節において各ループごとに分割候補を生成した．次に複数ループ間において分割候補間の不整合を解決する．この際に，分割候補間の整合性を判定し，整合する場合には両分割候補を結合して新たな分割候補を作成する操作を行う．このように整合性を持つ複数の分割候補を結合した分割候補を共通分割候補と呼ぶ．

2 つの分割候補の整合条件は以下のとおりである．分割候補 1，2 の整列ターゲットを各々 $T_1$，$T_2$ とする．この際，$T_1$ の寸法 $\geq T_2$ の寸法となるように分割候補 1，2 を選ぶ．また，分割候補 1，2 に共通する配列が，2.3 節で述べた形式によると，分割候補 1 では $\{A_n.v_n.x_n\}$，分割候補 2 では $\{A_n.w_n.y_n\}$ と表現されるとする．$1 \leq n \leq$ 分割候補 1，2 に共通する配列の数である．このとき，

$T_1$ の分割次元の寸法 =

$T_2$ の分割次元の寸法 $\times N$（$N$ は自然数）

かつ $\forall n$ に対して $v_n = w_n$，$x_n = y_n \times N$

ならば，分割候補 1，2 は整合したとする．

共通分割候補は，分割候補 1 の $\{A_n.v_n.x_n\}$ 以外の配列を $\{B_n.p_n.s_n\}$，分割候補 2 の $\{A_n.w_n.y_n\}$ 以外の配列を $\{C_n.q_n.t_n\}$ とするとき，

$\{A_n.v_n.x_n\} \cup \{B_n.p_n.s_n\} \cup \{C_n.q_n.(t_n \times N)\}$

となる．

たとえば，分割候補（F の分割次元の寸法=600）

A.1.4, B.1.2, C.2.2, D.1.4, E.0.0, F.1.1

と，分割候補（C の分割次元の寸法=300）

A.1.2, B.1.1, C.2.1, G.2.2

は整合し，共通分割候補は

A.1.4, B.1.2, C.2.2, D.1.4, E.0.0, F.1.1, G.2.4

となる．

### 2.5 手続き内における分割候補の選択

本節では，手続き内において複数ループ間のデータ分割の整合性を考慮しつつ適切な分割候補を選択し，必要に応じて再分割を行う手法について述べる．本手法は，greedy な戦略により，ループごとの効率的な並列化とループ間での通信の抑制を目指す．具体的には，CFG 上で優先度が高いノードから始めて，優先度が高いエッジに沿って，順にノードをグループ化してい

く．グループとは，同じデータ分割を共有するノードと，そのノードにつながるエッジの集まりである．優先度が高いノードからグループ化を始めることによって，配列の総アクセス回数が多いループ，すなわち時間がかかると考えられるループのデータ分割を最初に決定し，このようなループが優先的に効率良く並列化されるようにする．また，優先度が高いエッジに沿ってグループ化を進めることによって，数多く実行される個所において優先的にデータ分割を合致させ，このような個所で再分割が起きることを回避し，通信オーバヘッドを抑制する．

グループに新たなノードを入れる際には，2.4 節の手順によってグループの分割候補と新たなノードの分割候補の整合性を判定する．両分割候補が整合するならば共通分割候補を求め，これを新たにグループの分割候補として，同様のグループ化を続ける．もしも両分割候補が不整合であれば，グループと新たなノードの間のエッジにおいて再分割が必要と判断する．なお，共通分割候補を求める際に，ノードが複数の分割候補を持つ場合には，すべての分割候補の組合せに関して整合性を調べ，各々の組合せに対して共通分割候補を求める．以下に手順を示す．なお，本手順におけるノードやエッジとは，CFG のノードやエッジを指す．

(1) グループ X について，$N_X$ をグループに属するノードの集合，$E_X$ をグループに属するエッジの集合，$C_X$ をグループの分割候補の集合とする．

(2) グループ G を新たに作成し，各集合の初期要素を以下のとおりにする．

$N_G = \{$ 未処理ノードで優先度が最高のもの $\}$
$E_G = \{N_G$につながる全エッジ $\}$
$C_G = \{N_G$の分割候補 $\}$

(3) $E_G$ に未処理エッジがある限り (3) から (5) までの手順を繰り返す．ここで，

$e =$ 最高優先度の未処理エッジ（$e \in E_G$）
$n = e$ の一端のノード（$n \notin N_G$）

とする．

$n$ がいずれのグループにも属さず，かつ $n$ につながる全未処理エッジの中で $e$ の優先度が最大のときには (4) へ，$n$ がいずれかのグループに属しているときには (5) へ進む．

(4) $C_G$ と $n$ の分割候補が整合するならば，以下の処理を行う．

$N_G \leftarrow N_G \cup \{n\}$
$E_G \leftarrow E_G \cup \{n$ につながる全エッジ $\}$
$C_G = \{G$ と $n$ の共通分割候補 $\}$

$C_G$ と $n$ の分割候補が整合しないならば，$e$ の位置で再分割を行う．

いずれの場合も $e$ を処理済みとし，(3) に戻る．

(5) $n$ がグループ $H$ に属しているとする．

$C_G$ と $C_H$ の分割候補が整合するならば，以下の処理を行う．

$N_G \leftarrow N_G \cup N_H$
$E_G \leftarrow N_G \cup N_H$
$C_G = \{C_G$と $C_H$の共通分割候補 $\}$

$C_G$ と $C_H$ の分割候補が整合しないならば，$e$ の位置で再分割を行う．

いずれの場合も $e$ を処理済みとし，(3) に戻る．

次に，図 1 を例に，手続き内における分割候補の具体的な選択手順を述べる．まず各ループノードの分割候補を求める．

- ノード 1：
  B.2.1, C.2.1
  B.1.1, C.1.1
- ノード 2：
  A.1.1, D.2.1, E.2.2
  A.2.1, D.1.2, E.1.2
- ノード 4：
  A.1.1, B.1.2, C.1.2
  A.2.1, B.2.1, C.2.1
- ノード 5：
  Q.2.1, R.2.2
  Q.1.1, R.1.1
- ノード 6：
  P.2.1, Q.2.1, R.2.2
  P.1.1, Q.1.2, R.1.2

主プログラム MAIN に関して分割候補を選択する．初期グループ $G$ の要素は以下のように定められる．$E_G$ の要素 $\{4 \rightarrow 2, 4 \rightarrow 3\}$ は，各々ノード 4 を出てノード 2，3 に入るエッジを示す．

$N_G = \{4\}$
$E_G = \{4 \rightarrow 2, 4 \rightarrow 3\}$
$C_G = \{\{A.1.1, B.1.2, C.1.2\},$
$\{A.2.1, B.2.1, C.2.1\}\}$

未処理エッジは優先度が同じであるから任意の順番で処理が可能であるが，ここでは $4 \rightarrow 2$ から処理する．ノード 4 とノード 2 は分割候補が整合するので，ノード 2 を $G$ に加え，各要素を以下のように変更する．

$N_G = \{4, 2\}$
$E_G = \{4 \rightarrow 3, 2 \rightarrow 3, 1 \rightarrow 2\}$
$C_G = \{\{A.1.1, B.1.2, C.1.2, D.2.1, E.2.2\},$
$\{A.2.1, B.2.1, C.2.1, D.1.2, E.1.2\}\}$

次に，エッジの優先度に従って，ノード 3 を加える．

ノード 3 は CALL ノードなので，手続き内においては分割候補に制約がないものとして扱う．次節において手続き間でデータ分割の整合性を考慮し，最終的なデータ分割を決定する．

$N_G = \{4, 2, 3\}$
$E_G = \{1 \rightarrow 2\}$
$C_G = \{\{\underline{A}.1.1, B.1.2, C.1.2, D.2.1, E.2.2\},$
$\{\underline{A}.2.1, B.2.1, C.2.1, D.1.2, E.1.2\}\}$

続いてノード 1 を加える．以下が MAIN の暫定的なデータ分割である．

$N_G = \{4, 2, 3, 1\}$
$E_G = \{\}$
$C_G = \{\{\underline{A}.1.1, B.1.2, C.1.2, D.2.1, E.2.2\},$
$\{\underline{A}.2.1, B.2.1, C.2.1, D.1.2, E.1.2\}\}$

手続き SUB に関しても同様の手順により暫定的なデータ分割を求める．

$N_G = \{6, 5\}$
$E_G = \{\}$
$C_G = \{\{\underline{P}.2.1, Q.2.1, R.2.2\},$
$\{\underline{P}.1.1, Q.1.2, R.1.2\}\}$

続いて，次節で述べる手続き間データ分割候補選択手法に基づき，最終的なデータ分割を決定する．

### 2.6 手続き間関係を考慮した分割候補の選択

本節では，手続き呼び出し時の通信オーバヘッドを少なくするために，手続き間においてデータ分割の整合性を考慮しつつ適切な分割候補を選択するための手法について述べる．具体的には，呼び出される手続きのデータ分割と整合し，かつ最も多くの CALL 文で共通するデータ分割を，手続きを呼び出すすべての CALL 文において選択することによって，手続き呼び出し時の再分割を抑制する．以下に手順を示す．

(1) すべての手続きについて (2) 以下の処理を行う．すべての CALL ノードは未処理とする．
(2) 未処理の CALL ノードがない手続きの 1 つを $P$ とする．
(3) 2.5 節で述べた手続き内の手法により，CFG の各ノードに対して 1 つあるいは複数の分割候補が選択される．$P$ において手続きの入口に相当するノードと，$P$ の全呼び出し元 CALL ノードの分割候補に関して共通分割候補を求める．求めた共通分割候補の集合を $\{c_n\}$ とする．
(4) $\{c_n\}$ の要素，すなわち分割候補が複数あるならば，以下の手順を行う．なければ，(5) に進む．
  (a) 各 $c_n$ に対し，全呼び出し元 CALL ノードのうち $c_n$ に整合する分割候補を持つものの実行回数を総和したスコアを求める．
  (b) もしどの $c_n$ もスコアを獲得できなければ，(4) に進む．
  (c) 最高のスコアを獲得した $c_n$ を，$P$，全呼び出し元 CALL ノード，および各呼び出し元 CALL ノードと同じグループに属する全ノードのデータ分割に決定する．そして全呼び出し元 CALL ノードを処理済みとし，(2) に戻る．
(5) 任意の $c_n$ を，$P$，全呼び出し元 CALL ノード，および各呼び出し元 CALL ノードと同じグループに属する全ノードのデータ分割に決定する．そして全呼び出し元 CALL ノードを処理済みとし，(2) に戻る．

前節の例に引き続いて，図 1 を例に，手続き間におけるデータ分割の整合性を考慮しつつ，最終的な分割候補を選択する具体的な手順を示す．

前節までの手順で，主プログラム MAIN および手続き SUB の暫定的なデータ分割を求めた．SUB には未処理の CALL ノードがないから，$P = SUB$ とする．$P$ の呼び出し元 CALL ノードは MAIN のノード 3 のみであり，分割候補は以下のとおりである．

A.1.1, B.1.2, C.1.2, D.2.1, E.2.2
A.2.1, B.2.1, C.2.1, D.1.2, E.1.2

$P$ の分割候補は以下のとおりである．

P.2.1, Q.2.1, R.2.2
P.1.1, Q.1.2, R.1.2

引数の対応関係 $(A \leftrightarrow P, B \leftrightarrow Q, C \leftrightarrow R)$ を考慮しつつ，上記の 2 つの分割候補の共通分割候補を求める．この場合，共通分割候補は 1 つなので，最終的なデータ分割に決定する．MAIN のデータ分割は，

A.1.1, B.1.2, C.1.2, D.2.1, E.2.2

となり，SUB のデータ分割は，

P.1.1, Q.1.2, R.1.2

となる．

仮に，手続き間の整合性を考慮せず，各手続きで独立に，最外側ループを並列化するようなデータ分割を行うとする．この場合，MAIN のデータ分割は，

A.1.1, B.1.2, C.1.2, D.2.1, E.2.2

SUB のデータ分割は，

P.2.1, Q.2.1, R.2.2

となり，SUB の呼び出し時に再分割が必要となる．すなわち，通信オーバヘッドが発生する．一方，本手法では，複数手続き間にわたるデータ分割の整合性を考慮するので，再分割を回避できる．

また，手続き内の手法と手続き間の手法は密接している．最終的に手続き間で分割候補間の整合性を考慮

して選択が行えるよう，各手続き内における選択処理の時点では複数の分割候補を保持できる構成になっている．

### 2.7 計 算 量

本手法の計算量について考察する．プログラム長を $M$，配列の数を $N$ とする．

CFG の生成（2.2 節）については，ノード数・エッジ数はともに $O(M)$ であると考えられ，CFG の生成コストも $O(M)$ である．ループごとのデータ分割候補生成処理（2.3 節）については，整列グラフの各ノードから他の全ノードへのエッジを張ることを考えると，整列グラフ 1 つにつき $O(N^2)$，整列グラフの総数は並列ノード数に等しいから全体としては $O(MN^2)$ である．共通分割候補 (2.4) については，同じ配列どうしについて比較を行うから $O(N)$ である．

手続き内における分割候補選択処理（2.5 節）については，多くの Fortran プログラムは過度に複雑な分岐構造を持たない，すなわち CFG の 1 つのノードにつながるエッジの数はたかだか数本であり，プログラム長 $M$ に比べると十分に小さいと見なせる．そこで，あるノードにつながるエッジの中で最高の優先度を持つエッジを発見するコストを定数オーダであるとすると，グループに属するエッジをソートする処理は $O(M \log M)$ となり，そして各エッジをグループに加える際に共通分割候補を求めるから，全体としては $O(MN \log M)$ となる．手続き間における分割候補選択処理（2.6 節）については，すべての CALL ノードに関して，呼び出される手続きの分割候補との間で整合性を調べるから，$O(MN)$ である．

## 3. 評 価

本稿において提案した手法をワークステーション上に実装した．そしてベンチマークプログラムを用いて，

(1) 自動分割版
本手法により元の Fortran プログラムに対し自動的に HPF 指示文を追加し並列化したもの

(2) MPI 版
MPI [13] を用いて手作業にて並列化したもの

に関して，並列計算機 NEC Cenju-4 [14] 上で実行時間を計測した．自動分割版のコンパイルには当研究室が RWC プロジェクトにおいて開発中の HPF コンパイラを用いた．また，ワークステーション（NEC EWS4800/470）において自動データ分割に要した時間もあわせて示した．この時間とは，入力ソースの解析，CFG や整列グラフの作成，グラフの解析とデータ分割の決定，指示文の挿入，HPF ソースの出力，のすべての過程に要した時間である．

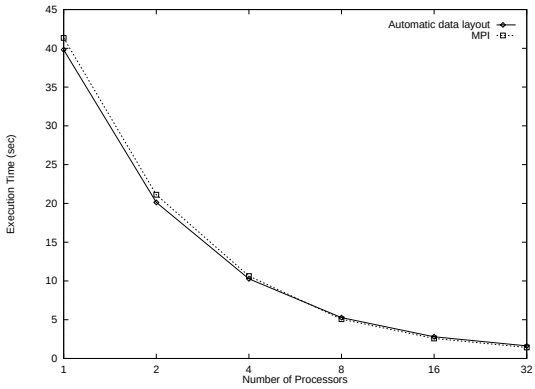


**図 3** tomcatv（513 × 513）の実行時間
Fig. 3 Execution time of tomcatv (513 × 513).

使用したのは以下のベンチマークプログラムである．

(1) APR Benchmark Suite の tomcatv
(2) NCAR Spectral Transform Shallow Water Model（shallow）
(3) NAS Parallel Benchmarks [15] の bt

今回は入力データとして，ソースに加え，ループの繰返し回数とプログラムの各文の実行回数を利用している．筆者らの先行研究[16]ではこれらの情報を手作業で集めたが，今回，自動収集が可能となった．

### 3.1 tomcatv

tomcatv は 2 次元のメッシュ生成プログラムである．自動データ分割の対象とした tomcatv は約 200 行のソースコード，1 個の手続きからなる．tomcatv の主要データは 2 次元配列群であり，主要な計算はこれらの配列群に対する 2 重ループである．MPI 版は逐次 Fortran 版を基に我々が手作業で並列化したものである．MPI 版は主要な配列群を 2 次元目で BLOCK 分割している．

本稿の手法により自動データ分割を行うと，主要な配列群が 2 次元目で BLOCK 分割された．これは MPI 版と同等のデータ分割である．結果として主要な多重ループ群の多くが最外側ループで並列化できた．再分割は生成されなかった．**図 3** に実行結果を示す．自動データ分割に要した時間は約 1 秒であった．

### 3.2 shallow

shallow は大気/海洋循環シミュレーションの主要アルゴリズムの 1 つである．自動データ分割の対象とした shallow は約 600 行のソースコード，7 個の手続きからなる．shallow の主要データは 2 次元配列群であり，主要な計算はこれらの配列群に対する 2 重ループである．MPI 版は逐次 Fortran 版を基に我々が手作業で並列化したものである．MPI 版は主要な配列群

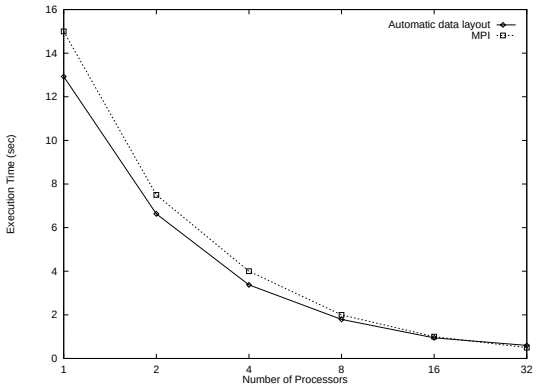


**図 4** shallow（513 × 513）の実行時間
Fig. 4 Execution time of shallow (513 × 513).

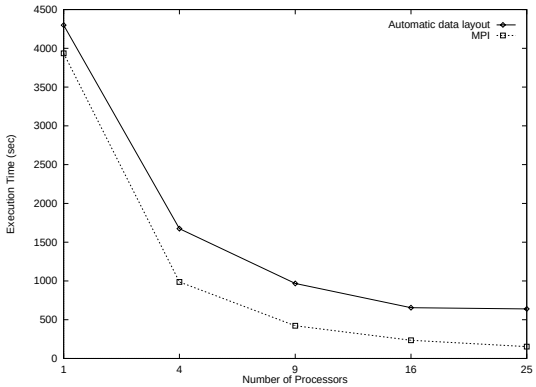


**図 5** bt（Class A）の実行時間
Fig. 5 Execution time of bt (Class A).

を 2 次元目で BLOCK 分割している．

本稿の手法により自動データ分割を行うと，主要な配列群が 2 次元目で BLOCK 分割された．これは MPI 版と同等のデータ分割である．結果として主要な多重ループ群の多くが最外側ループで並列化できた．再分割は生成されなかった．**図 4** に実行結果を示す．自動データ分割に要した時間は約 3 秒であった．

### 3.3 bt

bt（block tridiagonal）は計算流体力学コードである．自動データ分割の対象とした bt は約 4400 行のソースコード，21 個の手続きからなる．bt の主要データは 5 次元配列群であり，主要な計算はこれらの配列群に対する 4 重ループである．MPI 版は NPB2.3b2 を用いた．

bt に関しては，使用した HPF コンパイラのループ並列化可否判定機能に一部制限があるため，INDEPENDENT 指示文のみを含むソースに対して本稿の自動データ分割手法を適用した．bt の主要ループには，最外側で並列化するための主要配列の分割として，5 次元目での分割が適しているループと 4 次元目での分割が適しているループの両方がある．本稿の手法により自動データ分割を行うと，主要な配列群が各ループにおいて最外側における並列化に適した次元で BLOCK 分割された．結果として主要な多重ループ群が最外側ループで並列化できた．**図 5** に実行結果を示す．なお，自動データ分割に要した時間は約 60 秒であった．

### 3.4 考　　察

実験において，本稿の自動データ分割手法はいずれの例でも主要ループを最外側で並列化するのに適するよう配列を分割することができた．実行時間に関しては，tomcatv，shallow については主要部分が単純なループであり再分割もないために，HPF による自動データ分割版が MPI 版と同等の実行時間を達成した．shallow における逐次実行時の性能差は，HPF コンパイラが Fortran + MPI コードへの翻訳過程において変数の宣言順序や使用順序を変化させ，実行時のキャッシュの状態に影響したためと考えられる．

また，shallow では，単独手続きを対象とするデータ分割手法に対し，複数手続き間の整合性を考慮してデータ分割を決定する本手法の有用性を検証するために，手続き間におけるデータ分割の整合性を考慮せず各手続きに関して独立にデータ分割を行った場合についても実行時間を測定した．顕著な差は観測されなかったが，本手法によるものは，各手続きで独立にデータ分割を決定したものに比べ，最大で 5%実行時間が短かった（プロセッサ数=32）．手続き間を考慮しないデータ分割手法の実行時間が長いのは，主プログラムと他の手続きの間で主要配列に対する再分割が生じたためである．

自動分割版の bt の実行時間が MPI 版の実行時間よりも長い理由は，両者のデータ分割の違いに起因すると考えられる．bt では 3 軸 (x,y,z) 方向の方程式系を解く．自動分割版では，主要配列全体をある次元でブロック状に分割し，各ブロックを別々のプロセッサに割り当てる．この場合，途中で主要配列全体を転置する必要があり，通信オーバヘッドが発生する．

一方，MPI 版では主要配列を格子状に区切り，通信が格子の境界のみで起こるよう各格子を別々のプロセッサに割り当てるため，通信オーバヘッドが小さい[15]．自動分割版は HPF を基盤としているため，MPI 版と同等のデータ分割を行うことは困難である．自動分割版と MPI 版の実行時間の差は，この通信オーバヘッドの差に起因すると考えられる．Wijngaart ら[17]は類似する各手法を比較している．

自動データ分割に要した時間に関して，bt に要した

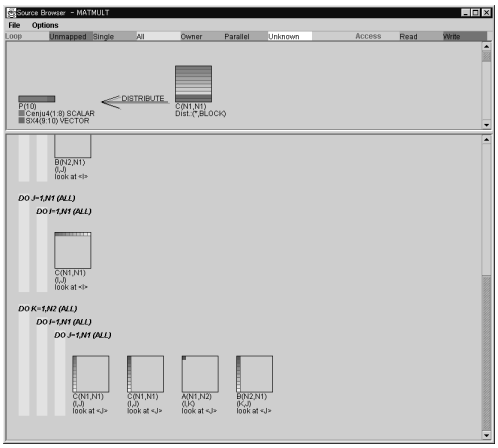


**図 6** ユーザインタフェース
Fig. 6 User interface.

時間は 60 秒と他の例に比べて長いが，ソースの入出力にこのうち約半分の時間を使っている．本稿における自動データ分割手法のみを適用するために要した時間は約 35 秒であった．

### 3.5 ユーザインタフェース

本データ分割手法の研究に並行して，ユーザが Fortran プログラムにおけるデータ分割を簡単な操作で指定できるグラフィカルなユーザインタフェース（UI）の開発も進めている（**図 6**）．この UI を使用することにより，ユーザと本手法との協調作業によるプログラムの並列化やチューニングが可能となる．スムーズに作業を進行させるうえで，本手法の高速性が有効である．

ユーザが一部の配列に関して明示的にデータ分割を指定した場合，本手法は，残りの配列に関し，ユーザ指定との整列関係を解析して，効率の良い並列化を実現し通信を抑制するデータ分割を行う．したがって，ユーザと本手法との協調作業が可能である．たとえば，並列化の鍵となる配列が既知の場合，この配列に関してのみ UI を用いてユーザがデータ分割を指定し，残りの配列に関しては本手法を適用することができる．また，本手法を適用した結果が好ましくない場合には，UI でいくつかの配列に関しデータ分割を明示的に指定した後に本手法を再適用すれば，結果を改善できる．本手法の高速性は，作業の繰返しを早め，チューニング時間を短縮する．

## 4. ま と め

本稿では，Fortran プログラムを並列化するために自動的にデータ分割を行う手法を提案した．本手法は，CFG と並列ループにおける配列アクセスのパターンを解析して，複数の手続き間にわたるデータ分割を短時間で決定する．本手法は実行時間が長いループを優先的に並列化し，ループ間では再分割を抑制することによって，対象プログラムの実行性能を向上する．また，ユーザが指示したデータ分割をプログラムの各所に反映することができる．ユーザが試行を重ねつつ，対話的にチューニング作業を進める際には本手法の高速性が役立つ．

本手法を実装し，ベンチマークプログラムに対して自動データ分割を行った．自動データ分割に要した時間は，500 行程度の入力プログラムに対しては数秒，4000 行程度で手続き数が 20 程度の入力プログラムに対しても約 1 分であった．さらに自動データ分割を行ったプログラムを並列計算機 Cenju-4 上で実際に動作させて，性能評価を行った．その結果，MPI 版とほぼ同等の実行速度と台数効果を得た．この実験を通して，本手法が効率的な並列化のためのデータ分割を短時間で自動的に決定できること，そして HPF コンパイラと組み合わせることにより効率的かつ自動的に Fortran プログラムを並列化できることを示した．

今後の課題は，(1) より複雑なデータ分割が必要なプログラムを用いた評価，(2) 多重ループの複数のネストレベルにわたる並列化，(3) 異機種並列分散環境への適用，などである．

(1) に関して，配列間の整列を生成するための実装を行った後に，より複雑な分散や整列，再分散や再整列を駆使しないと効率の良い並列化ができないようなプログラムに対して本手法を適用し，評価する必要がある．(2) に関しては，本手法では多重ループの 1 つのネストレベルのみを並列化するが，並列マシンによっては複数のネストレベルにわたる並列化が効果を発揮する場合も考えられる．本手法を単純に拡張することによりこのような並列化に対応することは難しいが，今後検討していく予定である．(3) に関しては，性格が異なる種々の計算機を組み合わせた異機種並列分散環境における並列プログラムの開発を容易にすることが目的である．そのために，各々の計算機の性格や性能の違いを考慮したデータ分割を行うよう，本手法の改良を進める予定である．

**謝辞** 本手法の実装および評価作業にご尽力いただいた NEC 情報システムズオープン技術システム事業部第一技術部の白戸幸正主任，渡辺幸光氏，神田大輔氏，吉本幸平氏に感謝申し上げます．

## 参 考 文 献

1) High Performance Fortran Forum: *High Performance Fortran Language Specification*

(1997).
2) Fox, G., et al.: Fortran D language specification, Technical Report TR90-141, Dept. of Computer Science, Rice University (1990).
3) Zima, H., et al.: Vienna Fortran – A Language Specification, Technical Report, Austrian Center for Parallel Computation, University of Vienna (1991).
4) Kremer, U.: NP-completeness of dynamic remapping, *Proc. 4th Workshop on Compilers for Parallel Computers* (1993).
5) Diderich, C.G. and Gengler, M.: A Heuristic Approach for Finding a Solution to the Constant-Degree Parallelism Alignment Problem, *Proc. International Conference of Parallel Architectures and Compilation Techniques* (*PACT'96*), IEEE (1996).
6) 辰巳尚吾ほか：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，Vol.96-PRO-8, pp.25–30 (1996).
7) Philippsen, M.: Automatic Alignment of Array Data and Processes To Reduce Communication Time on DMPPs, *Proc. 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (1995).
8) Sheffler, T.J., et al.: Aligning Parallel Arrays to Reduce Communication, *Proc. Frontiers '95*, pp.324–331 (1995).
9) Anderson, J. and Lam, M.: Global Optimization for Parallelism and Locality on Scalable Parallel Machines, *Proc. SIGPLAN'93 Conference on Program Language Design and Implementaion* (1993).
10) Bixby, R., et al.: Automatic data layout using 0-1 integer programming, *Proc. International Conference on Parallel Architectures and Compilation Techniques* (*PACT94*), pp.111–122 (1994).
11) Adve, V., et al.: Requirements for data-parallel programming environments, *IEEE Parallel and Distributed Technology*, Vol.2, No.3, pp.48–58 (1994).
12) Kennedy, K. and Kremer, U.: Automatic Data Layout for High Performance Fortran, *Proc. 1995 ACM/IEEE Supercomputing Conference* (1995).
13) MPI Forum: *MPI: A Message-Passing Interface Standard* (1995).
14) Nakata, T., et al.: Architecture and the Software Environmant of Parallel Computer Cenju-4, *NEC RESEARCH & DEVELOPMENT*, Vol.39, No.4, pp.38–390 (1998).
15) Bailey, D., et al.: The NAS Parallel Benchmarks, Technical Report RNR-94-007, RNR (1994).
16) 松浦健一郎ほか：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，並列処理シンポジウム JSPP'99 論文集，pp.87–94 (1999).
17) Van der Wijngaart, R.F.: Efficient Implementation of a 3-dimensional ADI Method on the iPSC/860, *Proc. Supercomputing '93*, pp.102–111 (1993).

(平成 11 年 8 月 30 日受付)
(平成 12 年 2 月 4 日採録)

**松浦健一郎**（正会員）

1974 年生．1996 年東京大学工学部電子情報工学科卒業．1998 年同工学系研究科電子工学専攻修士課程修了．同年 NEC 入社．現在，C&C メディア研究所に勤務．並列コンパイラ，並列プログラミング支援ツールの研究開発に従事．

**村井　均**（正会員）

1971 年生．1994 年京都大学工学部情報工学科卒業．1996 年同情報工学専攻修士課程修了．同年 NEC 入社．現在，C&C メディア研究所に勤務．並列化コンパイラの研究開発に従事．

**末広　謙二**（正会員）

1965 年生．1988 年京都大学工学部電気工学科卒業．1990 年同大学院情報工学専攻修士課程修了．1993 年同博士課程単位取得退学．同年 NEC 入社．現在，C&C メディア研究所主任．並列計算機の基本ソフトウェアの研究開発に従事．

**妹尾　義樹**（正会員）

1961 年生．1984 年京都大学工学部情報工学科卒業．1986 年同大学院修士課程修了．同年 NEC 入社．C&C 研究所にてスーパーコンピュータの研究開発に従事．並列処理アーキテクチャ，分散メモリマシンのための並列化言語環境，並列アルゴリズムに興味を持つ．現在 C&C メディア研究所研究マネージャ．工学博士．1988 年本会論文賞受賞．