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法

廣岡孝志[†] 太田寛[†] 菊池純男[†]

分散共有メモリ向けコンパイラにおける自動データ分散方法を提案する．まず，ファーストタッチ方式と呼ばれるOSのデータ分散方式をコンパイラで制御する「ファーストタッチ制御方法」を提案する．本方法により，カーネルループにおいて一部のみが参照される配列や，手続き間で宣言形状が異なる配列に対して，最適なデータ分散を実現することができる．その結果として，このような配列のデータローカリティを向上させ，またメモリアクセス集中を削減することができる．次に，ファーストタッチ制御方法，データ再分散解析，手続き間解析を用いた分散共有メモリ向け自動データ分散方法を提案する．ベンチマークプログラムNPB2.3serialのFTを用いた評価により，本自動データ分散を行わない場合に比べて16プロセッサ時で約76%性能が向上することを確認した．

Automatic Data Distribution Method by First Touch Control for Distributed Shared Memory

Takashi Hirooka,[†] Hiroshi Ohta[†] and Sumio Kikuchi[†]

We propose an automatic data distribution method for distributed shared memory compilers. First, We propose the first touch control method by which the compiler controls the first touch data distribution of the operating system. This method can achieve appropriate data distribution when there are partial array accesses in the kernel loop and when the declarations of array shape differ interprocedurally. As a result, it improves the data locality and reduces the memory hot-spot in such cases. Next, we propose an automatic data distribution method for distributed shared memory using the first touch control method, data redistribution analysis, and interprocedural analysis. By preliminary evaluation by the benchmark FT of NPB2.3serial, we show performance improvement of 76% in the case of 16 processors.

1. はじめに

分散共有メモリ（DSM）型並列計算機は，共有メモリの容易な並列プログラミング環境を提供しながら，分散メモリのスケーラビリティを確保できるアーキテクチャとして注目を集めている[8)~11),15)]．並列計算機を効率良く利用するためには，並列化率，ロードバランス，データローカリティが重要となる．本研究では，データローカリティの最適化に着目した．物理的に分散されたメモリを有するDSMでは，最適なデータ分散が必要不可欠であり，このためコンパイラの果たすべき役割は大きいと考えられる．そこで，本稿では，コンパイラで最適なデータ分散を自動的に決定する方法を示す．

従来，物理分散メモリを有する並列計算機に対するデータ分散方法として，データ分散指示文による方法[6),12)]，OSが行うファーストタッチ方式データ分散方法[1)]などが研究されてきた．また，自動データ分散方法としては，主にデータ分散指示文を自動生成する方法が利用されてきた[2)~5),16),17)]．最適なデータ分散を決定する問題はNP完全であるため，Kennedyら[4)]，Guptaら[5)]，辰巳ら[16)]，松浦ら[17)]などから，近似解を求める様々なヒューリスティクが提案されてきた．ところが，従来のデータ分散方法では実プログラム中に比較的よく表れるある種のプログラムパターンにおいて最適なデータ分散が実現できなかった．それは，カーネルループにおいて配列の一部のみが参照される場合や，手続きごとに引数配列の宣言形状が異なる場合などである．本稿では，これを解決するデータ分散方法として，OSが行うファーストタッチ方式データ分散をコンパイラで制御するファーストタッチ制御方法を提案し，ファーストタッチ制御方法，手続き間解析，データ再分散解析を用いた自動データ分散方法を示す．本稿で提案する自動データ分散方法では，配列ごとに手続き間でカーネルループを決定し，カーネルループ向けのデータ分散を配列の初期データ分散とする．さらにデータ再分散解析を行い，動的データ分散

† 新情報処理開発機構マルチプロセッサコンピューティング日立研究室
RWCP Multiprocessor Computing Hitachi Laboratory

を決定する．

実プログラムで幅広く効果のある自動データ分散方法を設計するためには，一定の効果が見込めるデフォルトデータ分散と，場合ごとに最適化する方法の組合せが必要となる．OSが行うファーストタッチ方式データ分散は，デフォルトデータ分散としてある程度効果の見込めることが示されている[1]．この方式は，単独でもある程度の効果を得たが，コンパイラで有効性を引き出す方向に誘導してやればさらなる効果が期待できる．本方法は，データ分散方法としてこれを利用したファーストタッチ制御方法とデータ分散指示文を併用し，従来，主に利用されてきたデータ分散指示文の不得手とする場面を克服するところが利点である．

本稿は次の構成をとる．2章では，3章で示すDSM向け自動データ分散方法の特徴技術であるDSM向けデータ分散方法のファーストタッチ制御方法を提案する．3章では，このファーストタッチ制御方法，手続き間解析，データ再分散解析を用いたDSM向け自動データ分散方法の実装を述べ，4章で評価する．5章でまとめる．

2. ファーストタッチ制御方法

本章では，従来のデータ分散方法では最適なデータ分散が実現できないいくつかの場合を示し，これを解決する新しいデータ分散方法を示す．

2.1 従来のデータ分散方法

DSMを実現する手段の1つとして，仮想メモリ空間をページ単位で切り分け，各ノードの物理メモリに割り付ける方法がある．このとき，各ノードに分散された物理メモリへデータを割り付ける方法には，いくつかの方法がある．今回，検討のプラットフォームとして用いた代表的DSM型並列計算機SGI/Origin2000には，ローカル参照の割合，すなわちデータローカリティを向上させる目的で用意されたデータ分散方法が2つある．

(1) データ分散指示文によるデータ分散[6),12)]

プログラム中のユーザが挿入したデータ分散指示文に従って，コンパイラがデータを各ノードに割り付ける．

(2) ファーストタッチ方式データ分散[1)]

OSが，各ページを最初にアクセスしたノードに割り付ける．

これら従来のデータ分散方法では，最適なデータ分散が実現できない場合がある．次節でその例を示し，これを解決するファーストタッチ制御方法を提案する．

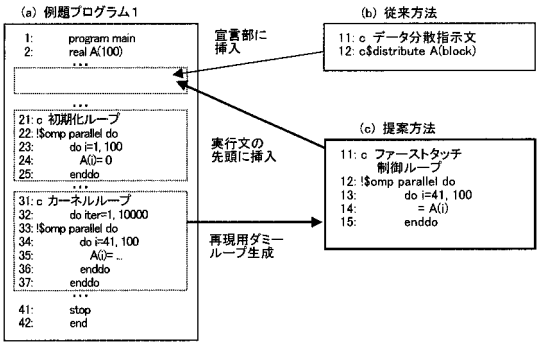

図1 ファーストタッチ制御方法
Fig. 1 First touch control method.

2.2 ファーストタッチ制御方法

2.2.1 基本アイディア

本節では，OSが行うファーストタッチ方式データ分散をコンパイラが制御することによるデータ分散方法を提案する．初めは，簡単のため再分散については考えない．この場合，最適な静的データ分散は，プログラム中の実行比率の高い部分，すなわちカーネルループでデータローカリティの高いデータ分散となる．

ファーストタッチ制御方法の具体的な内容を説明する．まず，コンパイラがカーネルループ中の参照パターンを再現するダミーループを生成してプログラムの先頭に挿入し，OSが行うファーストタッチ方式データ分散に従ってデータ分散させることにより，カーネルループ向けのデータ分散を実現させる．なお，以下ではデータ分散の対象となる配列を**ターゲット配列**と呼ぶ．

図1 (a)に例題プログラム1を示す．本プログラムは，初期化ループにおいて配列Aの全要素を参照するが，カーネルループでは部分配列A(41:100)のみを参照する．このようなタイプのプログラムはFortranプログラムでは比較的よく現れる．実プログラム上では手続き呼び出しの際に部分配列を引き渡すことにより，意味的に本例のようなプログラムとなることが多い．例では簡単のため手続き内のソースイメージで説明する．

従来，図1(b)に示すようなデータ分散指示文を宣言部に挿入してデータ分散を行っていた[14)]．この方法によると，データは全要素が**図2**(a)に示すように各プロセッサに均等に割り付けられる．ところが，ループ分散方法をblock分散と仮定すると，カーネルループにおける参照範囲は図2(b)に示すようにA(41:100)を均等に各プロセッサに割り付けた状態となる．したがって，図2(a)と図2(b)の差である図2(c)に示す範囲，実にカーネルループの全参照における67%がリ

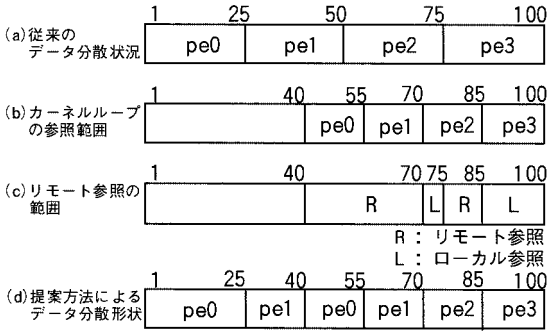

図 2 データ分散状況
Fig. 2 Illustration of data distribution.

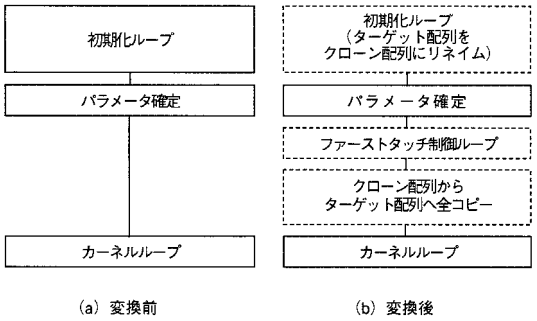

図 3 パラメータ確定位置が初期化ループより後の場合の変換方法
Fig. 3 Transformation when parameters are defined after the initalization loop.

モート参照となる．これではデータローカリティの低さから十分な並列性能を得ることができない．もう1つの従来方法として，プログラム変更を行わず，OSの行うファーストタッチ方式データ分散に従う方法があるが，初期化ループが全要素を参照するため，先の方法と同様に図2(a)に示すようなデータ分散となり，データローカリティに関して同じ結果となる．

そこで，カーネルループ中におけるターゲット配列Aの参照パターンを再現する図1(c)に示すようなダミーループを生成する．以後，このループを**ファーストタッチ制御ループ**と呼ぶ．次に，ファーストタッチ制御ループを実行文の先頭に挿入し，OSが行うファーストタッチ方式データ分散に従って各要素を各プロセッサに割り付ける．その結果，図2(d)に示すようなデータ分散が実現でき，カーネルループにおける参照は100%ローカル参照となる．

なお，本例では，基本アイディアを簡潔に説明するため従来方法の一例としてblock分散を用いたが，従来でもHPFのINDIRECT分散[12]などの複雑な仕様の指示文を用いれば図2(d)に示すようなデータ分散の実現は可能であった．しかし，ユーザにとって面倒なデータ分散指示文を記述する手間を削減できることは大きな利点になると考えられる．

また，本方法は以下のような場合にも有効である．実プログラムにおいては，配列を引数として手続き間で渡していくことが多い．引数配列に指示文で静的なデータ分散を指示する場合，引数配列が出現する最親手続きで指示する必要がある．ところが，引数配列の宣言形状が引き渡されていく各手続きで異なる場合，最適データ分散はカーネルループを含む手続きにおける宣言形状をベースに表現されるため，最親手続きでそれを指示することができない．このような場合にも，ファーストタッチ制御方法によれば，手続き呼び出し関係ごとカーネルループの参照パターンを再現するコードを生成し，プログラムの先頭で実行させ，OSの行うファーストタッチ方式に従ってデータ分散させることにより，最適データ分散が実現できる．

2.2.2 本方法の拡張

以下に示すようなパラメータが変数である場合，パラメータの値が確定する位置によってファーストタッチ制御ループの挿入位置を変える必要がある．

- ターゲット配列の宣言寸法
- カーネルループのループ制御変数の上限，下限，増分
- ターゲット配列の添字式に含まれるループ制御変数以外の変数

パラメータが定数の場合は，単純に実行文の先頭にファーストタッチ制御ループを挿入すればよいが，パラメータが変数の場合は，パラメータが確定した後にファーストタッチ制御ループを挿入しなければならない．さらに，パラメータの確定位置が初期化ループの前か後ろかで対処の方法が異なる．パラメータの確定位置が初期化ループの前の場合，パラメータ確定位置の直後にファーストタッチ制御ループを挿入する．パラメータの確定位置が初期化ループの後の場合，**図3**に示すような変換を行う．

まず，ターゲット配列と同じ宣言形状の配列（以下，**クローン配列**）を生成し，初期化ループ中のターゲット配列の参照をクローン配列にリネイムする．次に，パラメータ確定の直後にターゲット配列に対するファーストタッチ制御ループ，およびクローン配列からターゲット配列への全要素コピーを挿入する．以上のような方法により，パラメータが変数であった場合にも本ファーストタッチ制御方法の適用が可能となる．

さらに，データ再分散への適用を考える．配列参照パターンの異なる複数のカーネルループが存在して，途中で動的にデータ再分散を行いたい場合，以下の変

換を行う．

- ターゲット配列のクローン配列を生成し，クローン配列を再分散後のデータ分散形状にデータ分散させるファーストタッチ制御ループを生成して実行文の先頭に挿入する．
- データ再分散位置でターゲット配列の全要素の値をクローン配列にコピーする．
- データ再分散位置以降のターゲット配列をクローン配列にリネイムする．

このように変換したプログラムを，OSが行うファーストタッチ方式データ分散に従ってデータ分散させることにより，データ再分散と同等の効果を得ることができる．3章の自動データ分散方法では，後述の3.2節(6)に示す基準に従って，本方法とOSのデータ再分散機能を併用する．

3. DSM向け自動データ分散方法

本章では，2章で示したファーストタッチ制御方法，データ再分散解析，手続き間解析を用いたDSM向け自動データ分散方法について述べる．

3.1 システム構成

図4にシステム構成を示す．本方法は，DSM並列化トランスレータのデータ分散部として実装中である．本トランスレータは，Fortran77で記述された逐次ソースプログラムを入力し，これに処理分散指示文，およびデータ分散実施コードを挿入したソースプログラムを出力する．なお，本稿ではデータ分散指示文とファーストタッチ制御コードの総称をデータ分散実施コードと呼ぶ．さらにDSM並列化コンパイラが，このソースプログラムを入力し，DSM並列オブジェクトプログラムを出力してDSM並列実行を実現する．

現在，DSM並列化コンパイラとしてはSGI/Origin2000に搭載されたMIPSPro Fortran77コンパイラを用いている．これにともない，トランスレータが挿入する処理分散指示文としてはOpenMP指示文を用い，データ分散指示文としてはOrigin2000向けのSGI指示文を用いている．

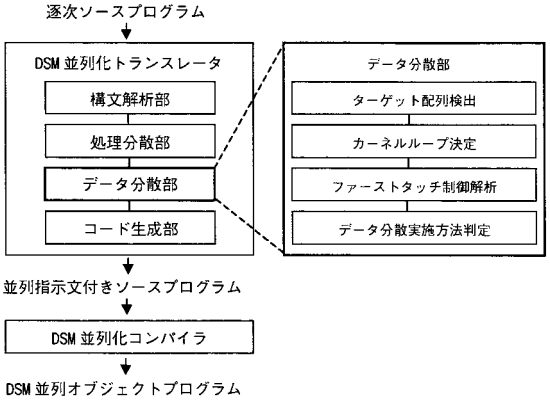

図4 システム構成
Fig. 4 Block diagram of the system.

データ分散部では，まず処理分散部で決定した並列化実施ループ内に参照を有するターゲット配列を検出する．次にターゲット配列ごとの各並列化実施ループ向けデータ分散形状，およびカーネルループを決定し，データ再分散解析を行う．最後にファーストタッチ制御コード再現用情報の解析を行い，データ分散実施方法の判定を行う．

3.2 方式説明

一般にプログラム中の部分ごとに最適なデータ分散は異なる．本研究で提案する自動データ分散方法は，以下の方針で設計した．まず，プログラム全体でデータ分散を固定した場合に最適となるデータ分散（静的データ分散）を決定し，初期データ分散とする．次に，データ再分散解析を行い，プログラム中からデータ再分散の必要があり，再分散コストに見合う部分を検出し，その部分に対してデータ再分散（動的データ分散）を実施する．

図5にDSM向け自動データ分散方法のアルゴリズムを示す．本方法では，主に以下に示す6つの処理を行う．

- ターゲット配列の検出
- 各並列化実施ループ向けデータ分散形状の決定
- カーネルループの決定
- 再分散解析
- ファーストタッチ制御向け解析
- データ分散実施方法の判定

各処理の詳細を以下に示す．

(1) ターゲット配列の検出

まず，プログラム中の全ループの中から並列化実施ループ決定方法[18]に従って決定した手続き間並列化実施ループを検出する．次に，並列化実施ループのループ本体に参照を有する配列の中から，以下の条件を満たすものをターゲット配列として検出する．

- 参照における添字式に並列化実施ループのループ制御変数を含む．
- ループ制御変数は，参照において1つの次元のみ，かつ1回のみ現れる．
- 参照における添字式が $A * i + B$ である（A，Bは定数，i はループ制御変数）．

(2) 各並列化実施ループ向けデータ分散形状の決定

ターゲット配列ごとに当配列の参照をループ本体に

```
1:  for (各ループ) do
2:      ・手続き間並列化実施ループの検出
3:      ・ターゲット配列の検出
4:  endfor
5:  for (各ターゲット配列) do
6:      ・並列化実施ループ向けデータ分散形状の決定
7:      ・手続き内カーネルループの決定
8:      if (ターゲット配列が引数配列) then
9:          ・手続き間カーネルループの決定
10:     endif
11:     ・データ再分散解析
12:     ・カーネルループ向け参照パターンの検出
13:     ・カーネルループ向け代表参照パターンの決定
14:     ・ファーストタッチ制御コード再現用情報の検出
15:     ・データ分散実施方法の判定
16: endfor
```

図 5 DSM 向け自動データ分散方法のアルゴリズム
Fig. 5 Algorithm for automatic data distribution method for distributed shared memory.

含む並列化実施ループを検出し，各ループ向けデータ分散形状を決定する．ループ向けデータ分散形状は，ループ本体の全参照において並列化実施ループのループ制御変数が最も多く現れる次元を block 分散した形状とする．

(3) カーネルループの決定

ターゲット配列ごとのカーネルループを決定する．まず，データ分散形状ごとに並列化実施ループをグループ化し，各並列化実施ループのコストを見積もってコストの合計が最大となるグループを決定する．なお，ループ本体の演算数とループ長の積をループのコストと呼ぶ．このグループの中でコスト最大の並列化実施ループを当ターゲット配列の手続き内カーネルループとする．次に，ターゲット配列が引数の場合，ターゲット配列が渡る各手続きにおける手続き内カーネルループを検出する．データ分散形状ごとに手続き内カーネルループをグループ分けし，コストの合計が最大となるグループを検出する．このグループの中でコスト最大の手続き内カーネルループを手続き間カーネルループとする．

(4) 再分散解析

まず，手続きローカル配列であるターゲット配列に対して以下の処理を行う．手続き内カーネルループ以外の並列化実施ループのうち，(2) で求めたデータ分散形状が手続き内カーネルループと異なり，データ再分散により高速化される時間がデータ再分散しきい値以上のループを検出し，データ再分散対象ループとする．ループのデータ再分散により高速化される時間は以下の式で求める．

$$N * \left(1 - \frac{1}{P}\right) * T \quad (1)$$

ここで，N はループ中のターゲット配列の全参照回数，P はスレッド数，T はリモート参照とローカル参照のアクセス時間の差分である．また，データ再分散しきい値とは，対象となる配列データを再分散させるために費やす時間を指す．その時間は，システム依存のパラメータ，スレッド数，配列サイズによって決定する．次に，引数であるターゲット配列に対して以下の処理を行う．手続き内カーネルループのうち，手続き間カーネルループとデータ分散形状が異なり，データ再分散により高速化される時間がデータ再分散しきい値以上のループを検出し，このループを含む手続きをデータ再分散対象手続きとする．

(5) ファーストタッチ制御向け解析

まず，各ターゲット配列に対し以下の解析を行う．

- 配列の参照における全次元の添字式の集合を参照パターンと定義し，ターゲット配列のカーネルループにおける全参照パターンを検出する．
- 全参照パターンのうち，最も出現回数の多い参照パターンを検出し，代表参照パターンとする．
- 代表参照パターンの添字式に含まれるすべてのループ制御変数の上限，下限，増分，およびループネスト順序を検出し，ループ再現用情報として保持する．

ターゲット配列が引数の場合，ターゲット配列が現れる最親手続きからカーネルループを含む手続きまでの手続き呼び出し経路を検出し，

- 手続き呼び出し順序
- 引数並び
- 各手続きにおける宣言形状

を同じく保持する．コールグラフが複雑な場合は，呼び出し回数が最大となる経路を検出し，その経路に関する上記情報を保持する．保持された情報は，ファーストタッチ制御コード生成において，ループを再現するために利用される．なお，以上はパラメータが定数の場合の方法である．変数の場合は，2 章で示した拡張方法を適用することにより実現可能となる．

(6) データ分散実施方法の判定

各ターゲット配列のデータ分散実施方法を判定する．まず，ターゲット配列が以下の条件を満たす場合ファーストタッチ制御方法を選択する．

- カーネルループにおいて部分配列参照が存在する．
- 引数配列であり，手続きごとに宣言形状が異なる．

その他の場合，指示文によるデータ分散方法を選択する．

まず，宣言された全要素数におけるカーネルループにおいて参照される要素数の割合を**参照要素率**と定義し，ターゲット配列が手続きローカル配列で参照要素率が 50%未満の場合，ファーストタッチ制御方法を選択する．50%以上の場合，指示文によるデータ分散方法を選択する．また，ターゲット配列が引数配列で参照要素率が 50%未満の場合，ファーストタッチ制御方法を選択する．50%以上の場合，引数配列が引き渡される全手続きにおける宣言形状を検出する．全手続きで宣言形状が一致している場合は指示文によるデータ分散方法を選択し，その他の場合はファーストタッチ制御方法を選択する．なお，2 つの方法の適用境界に参照要素率 50%を用いた理由は，次の 4.1 節で述べる．

4. 評　　価

本研究で提案する DSM 向け自動データ分散方法の効果を評価するため，例題プログラムと NPB2.3serial/FT に人手で本方法を適用してプログラム変換し，SGI/Origin2000 を用いて測定した．本章では，その結果を述べる．

4.1 カーネルループで部分配列が参照される場合

カーネルループにおいてターゲット配列の一部のみが参照されるプログラムの性能を評価した．**図 6** に示すカーネルループに対し，**表 1** のように外側ループ j の繰返し範囲を変化させて参照要素率を変え性能向上比を測定した．

本評価では，明示的にデータ分散形状を指定することのできる従来のデータ分散指示文による方法と本研究で提案するファーストタッチ制御方法を比較する．このカーネルループ向けのデータ分散指示文は，“c$distribute a(*,block)”となる．これに対し，ファーストタッチ制御方法では，**図 7** に示すようなファーストタッチ制御ループを生成する．

2 つの方法を比較するため，上記データ分散指示文とファーストタッチ制御ループをそれぞれプログラムの先頭に挿入した並列プログラムを生成し，カーネルループの性能を測定した．結果を**図 8** に示す．なお，データサイズは 32768 * 1024 とし，その他の測定条件は**表 2** に示すとおりとした．

ファーストタッチ制御方法を用いた場合，データ分散指示文を用いた場合に比べてファーストタッチ制御

```
1:   real a(n1,n2)
10:  do iter = 1, 100
11:  !$omp parallel do
12:     do j = 1, n2
13:        do i = 2, n1
14:           a(i,j) = a(i-1,j)
15:        enddo
16:     enddo
17:  enddo
```

図 6 カーネルループ
Fig. 6 Kernel loop.

表 1 参照要素率
Table 1 Reference element ratios.

参照要素率	ループ繰返し範囲（12 行目）
100%	do j = 1, n2 * 4/4
75%	do j = 1, n2 * 3/4
50%	do j = 1, n2 * 2/4
25%	do j = 1, n2 * 1/4

```
21:  !$omp parallel do
22:  do j = 1, n2
23:     do i = 2, n1
24:        dummy(i,j) = a(i,j)
25:     enddo
26:  enddo
```

図 7 ファーストタッチ制御ループ
Fig. 7 First touch control loop.

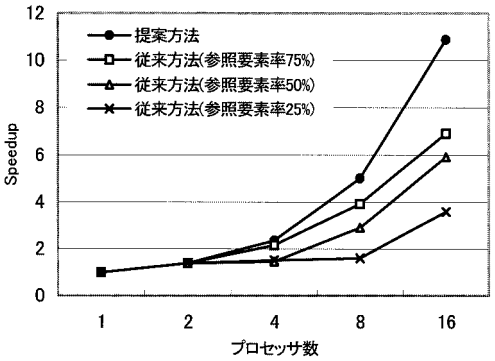

図 8 カーネルループの性能向上比
Fig. 8 Speedup for the kernel loop.

表 2 測定条件
Table 2 Measurement condition.

マシン	SGI/Origin2000
ノード	16 ノード（2 プロセッサ/ノード）
CPU	MIPS RISC R10000（195 MHz）
キャッシュ	L1: 32 KB, L2: 4 MB
メモリ	11 GB（11264 MB）
OS	IRIX6.5
コンパイラ	MIPSPro Fortran77 （Version7.2.1）
コンパイルオプション	-mp -Ofast = IP27 -OPT: IEEE_arith = 3

コードの実行時間がオーバヘッドとして加算される．しかし，一般的にカーネル部分は，繰り返し実行されるなどプログラム全体における実行比率が高く，ファーストタッチ制御コードで費やされる時間は，たかだかターゲット配列の全要素参照に費やす時間であるため，前者に比べて十分小さいと考えられる．この観点から本例でもカーネルループの外側にループ長 100 のループを置き測定を行った．したがって，本例の場合，ファーストタッチ制御方法を用いたことによるオーバヘッド，すなわちファーストタッチ制御コードの実行時間は，カーネルループの約 1%に相当する．

評価結果を分析する．ファーストタッチ制御方法は，参照要素率の変化に関係なくスケーラビリティがほぼ一定，かつ良好であった．一方，データ分散指示文による方法では，参照要素率 100%の場合は，データローカリティに関してファーストタッチ制御方法と同じ状態となるので性能の差はほとんどないものの，参照要素率を減少させると徐々にスケーラビリティが悪化した．2 章で説明したように，データローカリティの悪化とメモリアクセス集中が原因と考えられる．測定に用いた SGI/Origin2000 は，1 ノード 2 プロセッサ構成であるため 4 プロセッサ以上の測定においてデータローカリティに影響が現れる．したがって，本自動データ分散方法では，4 プロセッサ以上でファーストタッチ制御方法と性能差が大きくなる参照要素率 50%を 2 つの方法の適用境界として用いた．

4.2 手続き間で宣言形状が異なる場合

次に，引数であるターゲット配列の宣言形状が手続き間で異なる場合について評価を行った．評価には，NPB2.3serial/FT を用いた．NPB2.3serial は，様々な並列機向け並列プログラムのベースとして用いるための逐次プログラムであり，その中の FT は 3 次元 FFT に関するプログラムである．まず，**図 9** に FT のカーネルループを示す．

このループは，ft(main) - fft - cffts1 の順で呼び出された手続き cffts1 の中にある．ターゲット配列は下線部で示した x，xout の 3 次元配列であるが，上位の手続きにおいては，各々

- ft(main) 手続きでは u1，u2
- fft 手続きでは x1，x2

という別名で 1 次元配列として宣言されている．

並列化実施ループ決定方法[18] に従い，このカーネルループを最外ループ (do k) で並列化する．したがって，ターゲット配列 x，xout を 3 次元目で block 分散すればデータローカリティが最も高くなるが，上位手続きでの宣言形状が異なるので，前述のとおりデータ分散

```
1:   complex*16 x(nx, ny, nz), xout(nx, ny, nz)
2:   complex*16 y(nblock, nz, 2)
3:   do k= 1, nz
4:      do jj= 0, ny - nblock, nblock
5:         do j= 1, nblock
6:            do i= 1, nx
7:               y(j, i, 1)= x(i, j+jj, k)
8:            enddo
9:         enddo
10:        call cfftz(y)
11:        do j= 1, nblock
12:           do i=1, nx
13:              xout(i, j+jj, k)= y(j, i, 1)
14:           enddo
15:        enddo
16:     enddo
17:  enddo
```

図 9 NPB2.3serial/FT のカーネルループ
Fig. 9 Kernel loop of NPB2.3serial/FT.

```
1:   subroutine ftc_fft(x1, x2)
2:   complex*16 x1(256*256*128), x2(256*256*128)
3:   call ftc_cffts1(x1, x2)
4:   return
5:   end
6:   subroutine ftc_cffts1(x, xout)
7:   complex*16 x(256,256,128), xout(256,256,128)
8:   !$omp parallel do
9:   do k= 1, 128
10:     do jj= 0, 240, 16
11:        do j= 1, 16
12:           do i= 1, 256
13:              x(i, j+jj, k)= 0
14:              xout(j, j+jj, k)= 0
15:           enddo
16:        enddo
17:     enddo
18:  enddo
19:  return
20:  end
```

図 10 ファーストタッチ制御コード
Fig. 10 First touch control code.

指示文でこれを簡単に実現する手段はない．そこで，本方法では，**図 10** に示すようなファーストタッチ制御コードを生成し，手続き呼び出し文 “call ftc_fft(u1, u2)”を ft(main) 手続きの実行文の先頭に挿入し，OS の行うファーストタッチ方式データ分散に従ってデータ分散させる．これにより，ターゲット配列 x，xout を 3 次元目で block 分散させた形状のデータ分散が実現でき，データローカリティを向上させることができる．なお，配列宣言寸法，ループ制御変数の上限，下限，増分は手続き間定数伝播により定数に置換されたものを利用している．

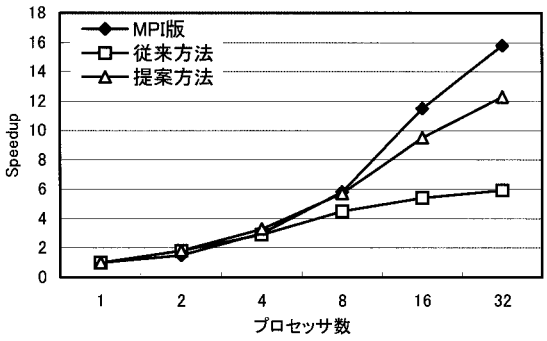

図 11 NPB2.3serial/FT（classA）の性能向上比
Fig. 11 Speedup for NPB2.3serial/FT (classA).

FT に本自動データ分散方法を適用した場合と，適用しない場合の性能向上比を**図 11** に示す．なお，測定条件は表 2 に示すとおりである．

評価では，以下に示す 3 つのバージョンの測定を行った．

(1) MPI 版プログラム（NPB2.3）
(2) 従来方法：OS のファーストタッチ方式データ分散のみを利用したプログラム
(3) 提案方法：コンパイラによるファーストタッチ制御データ分散と OS のファーストタッチ方式データ分散を協調させて利用したプログラム

従来方法では，プロセッサ数の増大とともにデータローカリティ低下の影響が大きくなり性能の伸びを妨げた．一方，提案方法では，ファーストタッチ制御コードのオーバヘッドが 16 プロセッサ時で約 1.5%発生するものの，高いデータローカリティによってすべてのプロセッサ数の範囲で良好な並列性能を示し，16 プロセッサ時では従来方法に比べて 76%の性能向上を示した．

なお，16 プロセッサ以上の範囲における MPI 版との性能差は，データ再分散によるデータローカリティの差の影響が大きいと考えられる．FT はデータ再分散を実施するとデータローカリティが向上することが分かっているが，SGI/Origin2000 では OS が行うデータ再分散が低速であるため，提案方法によるプログラムでは再分散を行っていない．実プログラムにおいて，再分散によりデータローカリティが大幅に改善されるプログラムは多い．したがって，DSM にとってデータ再分散機能の性能向上は重要であると考えられる．

本研究の最終的な目的は，DSM の容易，かつ効率的な利用環境をユーザに提供するために高性能な自動並列化コンパイラを開発することにある．以上の評価結果から，自動並列化コンパイラを用いて DSM を利用する場合，コンパイラによる自動データ分散機能を抜きにしては，SMP で良い性能を得ることができたプロセッサ数の範囲でしか通用しないことが明らかである．よって，自動並列化コンパイラを用いて DSM が SMP より大きなプロセッサ数で十分な並列性能を得るためには，高性能な自動データ分散機能が必須であると考えられる．

5. ま と め

本稿では，DSM 向け自動データ分散方法について述べた．この中で，従来のデータ分散方法では最適なデータ分散が実現できないいくつかの場合にも適用可能なファーストタッチ制御方法を提案した．また，ファーストタッチ制御方法，手続き間解析，データ再分散解析などにより動的，静的に DSM 向けの最適データ分散を実現する自動データ分散方法を提案した．評価として，ベンチマークプログラム NPB2.3serial の FT を人手変換して本自動データ分散方法によるデータ分散実施コードを挿入し，SGI/Origin2000 上で性能測定を行った．その結果，16 プロセッサ時で本自動データ分散を行わない場合に比べて約 76%性能が向上することを確認した．

本研究は，ファーストタッチ制御方法の基本的な効果の早期検証を目的とする一面があったため，3.2 節の自動データ分散方法の妥当性やファーストタッチ制御方法の適用条件についてなど厳密な比較，検討ができていない部分が残っている．今後は，これら未検討部分への対応，および本自動データ分散方法の早期実装，評価を行い，多くのベンチマークへ適用してその有効性を確認したい．また，そこから得た情報を基に機能拡張，および性能向上に取り組んでいきたい．

謝辞 本研究を進めるにあたり貴重なご助言，ご討論をいただいた新情報日立研究室の飯塚孝好氏および同研究室の諸氏に感謝いたします．

参 考 文 献

1) Chandra, R., Chen, D., Cox, R., Maydan, D.E., Nedeljkovic, N. and Anderson, J.: Data Distribution Support on Distributesd Shared Memory Multiprocessors, *Proc. PLDI '97*, pp.334–345 (1997).
2) Nguyan, T.N. and Li, Z.: Interprocedural Analysis for Loop Scheduling and Data Allocation, *Parallel Computing*, Vol.24, No.3-4, pp.477–504 (1998).
3) Ayguade, E., Garcia, J. and Kremer, U.: Tools and techniques for automatic data layout: A case study, *Parallel Computing*, Vol.24, No.3-4, pp.557–578 (1998).

4) Kennedy, K. and Kremer, U.: Automatic Data Layout for High Performance Fortran, *Proc. Supercomputing '95* (1995).
5) Gupta, M. and Banerjee, P.: PARADIGM: A Compiler for Automatic Data Distribution on Multicomputers, *Proc. ICS '93*, pp.87–96 (1993).
6) Hall, M.W., Hiranandani, S., Kennedy, K. and Tseng, C.W.: Interprocedural Compilation of Fortran D for MIMD Distributed-Memory Machines, *Proc. Supercomputing '92*, pp.522–534 (1992).
7) Hiranandani, S., Kennedy, K. and Tseng, C.W.: Evaluating Compiler Optimizations for Fortran D, *J. of Parallel and Distributed Computing*, Vol.21, pp.27–45 (1994).
8) Tseng, C.W., Anderson, J.M., Amarasinghe, S.P. and Lam, M.S.: Unified Compilation Techniques for Shared and Distributed Address Space Machines, *Proc. ICS '95*, pp.67–76 (1995).
9) Laudon, J. and Lenoski, D.: The SGI Origin: A ccNUMA Highly Scalable Server, *Proc. ISCA '97*, pp.241–251 (1997).
10) Abandah, G.A. and Davidson, E.S.: Effects of Architectural and Technological Advances on the HP/Convex Exemplar's Memory and Communication Performance, *Proc. ISCA '98*, pp.318–329 (1998).
11) Soundararajan, V., Heinrich, M., Verghese, B., Gharachorloo, K., Gupta, A. and Hennessy, J.: Flexible Use of Memory for Replication/Migration in Cache-Coherent DSM Multiprocessors, *Proc. ISCA '98*, pp.342–355 (1998).
12) High Performance Fortran Forum: *High Performance Fortran Language Specification Version 2.0* (1997).
13) Sato, M., Hirooka, T., Wada, K. and Yamamoto, F.: Program Partitioning Optimizations in an HPF Prototype Compiler, *Proc. COMPSAC '96*, pp.124–131 (1996).
14) SGI MIPSpro Fortran77 Programmer's Guide, Silicon Graphics Inc.
15) Inagaki, T., Niwa, J., Matsumoto, T. and Hiraki, K.: Supporting Software Distributed Shared Memory with an Optimizing Compiler, *Proc. ICPP '98*, pp.225–234 (1998).
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, pp.25–30 (1996).
17) 松浦健一郎，村井　均，末廣謙二，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，JSPP '99 論文集，pp.87–94 (1999).
18) 飯塚孝好，佐藤茂久，蓮見勝久，菊池純男：手続き間並列化コンパイラ WPP の試作—現状と今後の課題，第 56 回情報処理学会全国大会論文集（1），pp.280–281 (1998).
19) 廣岡孝志，太田　寛，菊池純男：配列リシェイプを用いた分散共有メモリ向けデータ再分散の最適化，第 57 回情報処理学会全国大会論文集（1），pp.289–290 (1998).
20) 廣岡孝志，太田　寛，菊池純男：分散共有メモリ向け自動データ分散方法の提案，情報処理学会研究報告，HPC, Vol.99, No.66, pp.101–106 (1999).

(平成 11 年 8 月 30 日受付)
(平成 12 年 3 月 2 日採録)

廣岡　孝志（正会員）

1966 年生．1985 年愛媛県立松山工業高等学校卒業．同年，(株）日立製作所入社．中央研究所を経て，現在，同社システム開発研究所に勤務．並列化コンパイラの研究に従事．並列処理ソフトウェア全般に興味を持つ．

太田　　寛（正会員）

1962 年生．1987 年東京大学大学院理学系研究科地球物理学専門課程修了．同年，(株）日立製作所入社．現在，同社システム開発研究所主任研究員．入社以来，論理型言語の研究を経て，並列化コンパイラの研究に従事．並列処理ソフトウェア全般，並列アーキテクチャに興味を持つ．電子情報通信学会，ACM，IEEE 各会員．

菊池　純男（正会員）

1952 年生．1978 年東京工業大学電気工学科修士課程修了．同年，(株）日立製作所入社．中央研究所にて図形処理技術の研究開発を経て，コンパイラ最適化技術，並列化技術の研究開発に従事．現在，同社システム開発研究所第 3 部部長．コンピュータアーキテクチャ，システムアーキテクチャ，コンパイラ技術に関心を持つ．ACM，IEEE 各会員．