ファーストタッチ制御を用いた間接参照配列向け最適化方法

廣岡孝志[†,††]

我々は，分散共有メモリ向けコンパイラにおける手続き間自動データ分散技術の実装を進めている．データ分散方法としては,「ファーストタッチ制御（FTC）方法」とデータ分散指示文を併用する．FTC 方法の特徴は，コンパイラが OS のファーストタッチ方式データ分散を制御することで，複雑なデータ分散に適確に対応できることである．これらの併用により，従来のデータ分散方法が不得手とするプログラムパターンに対し，最適なデータ分散が実現可能となる．今回，この FTC 方法を間接参照配列に適用する最適化方法を設計し，ベンチマークプログラム NPB2.3serial/CG（class B）を用いた評価を行った．その結果，本拡張を適用する前のデータ分散指示文を用いた自動データ分散に比べて 1.3 倍，OS のファーストタッチ方式データ分散に比べて 6.5 倍に性能が向上することを確認した（32 プロセッサ時）．

Optimization for Indirect Array References Using First Touch Control

Takashi Hirooka[†,††]

We are implementing an interprocedural automatic data distribution technique for our distributed shared memory compiler. This method combines the “First Touch Control (FTC)” with data distribution directives. The characteristics of FTC is that our compiler controls first touch data distribution of the operating system and accurately determines complex data distributions. By this combined method, we can achieve appropriate data distributions for program patterns which conventional data distribution methods can’t treat properly. This time we designed the optimization technique which applies this FTC method to indirect array references. In addition, by preliminary evaluation by the benchmark CG(class B) of NPB2.3serial, it runs 1.3 times faster than CG with the compiler ’s data distribution by directive, 6.5 times faster than CG without the compiler ’s data distribution(on 32 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．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．特に，物理分散メモリを有する DSM ではデータローカリティが性能を左右する重要な要素となる．そこで，本研究ではデータローカリティの最適化に着目した．並列プログラミングが容易であるという DSM の利点を損なうことなく十分なスケーラビリティを得るためには，自動的に最適なデータ分散を決定する機能をシステム側から提供することが必要となる．我々は，有効かつ適用範囲の広い自動データ分散機能の実現のためには，ハードウェアや OS の工夫だけでは限界があり，コンパイラの解析情報を利用したレベルの自動データ分散が必須であると考えた．そこで，本研究では最適なデータ分散をコンパイラで自動的に決定する方法を提案し，実装を進めてきた[7]．

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ユーザ指示，OS が行うファーストタッチ方式データ分散などが実用化されてきた．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現れる部分配列参照（実行比率が高い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を指す）などのプログラムパターンに対して，容易に最適なデータ分散を実現できない場合があった．この問題を解決するため，本研究では，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で制御するファーストタッチ制御

† 株式会社日立製作所システム開発研究所
Systems Development Laboratory, Hitachi Ltd.
†† アドバンスド並列化コンパイラ研究体
Advanced Parallelizing Compiler Project

（FTC）方法を提案し，自動データ分散方法で用いるデータ分散方法の 1 つとして利用してきた．我々の自動データ分散方法では，FTC 方法と従来のデータ分散指示文を併用することにより，従来のデータ分散指示文では適切な指示が困難な部分配列参照などに対しても最適なデータ分散を実現し，幅広いプログラムで良好なスケーラビリティを得てきた．今回，FTC 方法の適用範囲をさらに拡大するため，間接参照配列向けの拡張方式を設計した．前報告[7] では，基本的な FTC 方法を用いた自動データ分散方法の実装と評価について述べた．本報告では，間接参照配列に FTC 方法を適用する方式の設計，およびベンチマークプログラム NPB2.3serial/CG[8] を用いた事前評価について述べる．

本論文の構成は以下のとおりである．2 章では，FTC 方法の概要を説明する．3 章では，実装を進めてきた DSM 向け自動データ分散方法の基本部分について述べる．4 章では，今回検討した FTC 方法の間接参照配列向け拡張方式の設計について述べる．5 章では，評価結果を示し，分析を行う．6 章で関連研究を紹介し，7 章で結論を述べる．

2. ファーストタッチ制御方法

DSM を実現する手段の主流として，仮想メモリ空間をページ単位で切り分けて各ノードの物理メモリに割り付ける方式がある．今回，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 もこの方式を採用している．本 DSM 方式において，ページをどのノードの物理メモリに割り付けるかを決める方法をデータ分散方法という．Origin2000 には，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

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

(2) ファーストタッチ方式データ分散

OS が，各ページを最初にアクセスしたノードの物理メモリに割り付ける．

ところが，上記従来方法 (1) では，

- 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
- 手続きごとに引数配列の宣言形状が異なる場合
- カーネルループにおいて間接参照が存在する場合

などに最適なデータ分散の実現が困難であるという問題があった．また，従来方法 (2) では，

- 配列を初めて参照する初期化ループが並列化不能の場合
- 初期化ループとカーネルループのループ構造，もしくは並列化構造が異なる場合
- 手続きごとに引数配列の宣言形状が異なる場合

などに同じく最適なデータ分散の実現が困難であるという問題があった．

そこで，我々はこの問題を解決するファーストタッチ制御方法を提案した．これは，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である．FTC 方法では，カーネルループ中におけるターゲット配列の参照パターンを再現するダミーループ（以後，FTC ループと呼ぶ）をプログラムの先頭で実行させ，OS のファーストタッチ方式データ分散によってデータ割付けを行わせる．結果として，カーネルループ向けの最適データ分散を実現することが可能となる．FTC 方法の利点は，従来最適なデータ分散を実現することが困難であった上記のようなケースにも最適なデータ分散を実現することが可能となる点である．また，従来方法では困難であった実行時情報を利用した最適データ分散が可能となる点である．FTC 方法のデメリットとしては，FTC ループのオーバヘッドが考えられるが，たかだか配列の全要素参照に費やす時間となるのでカーネルループの実行時間に比べて十分小さい．

3. DSM 向け自動データ分散方法

本研究で実装を進めてきた自動データ分散方法では，2 章で示した FTC 方法と従来のデータ分散指示文を併用することにより，OS のファーストタッチ方式やデータ分散指示文といった従来方法が苦手としてきたプログラムパターンに対応しながら，幅広いプログラムで良好なスケーラビリティを得ることができる．ここでは，本論文で提案する拡張のベースとして用いた自動データ分散方法の概要を述べる．

分散メモリ機構を有する並列計算機の場合，各データは，そのデータが初めて参照されるまでには，何らかの分散形状で各ノードの物理メモリに配置されている必要がある．また，その参照においては，最初に決定したデータ分散形状をプログラム全体を通して利用し続けるか，必要な場面でデータ再分散を行うしかない．その結果，マシンの特性によっては，最初に行ったデータ分散の形状，およびデータ再分散コストが，性能に大きく影響を及ぼすことになる．そこで，本自動データ分散方法では，手続き間解析によりプログラム全体でデータ分散形状を固定させた場合に最適となる

データ分散形状を決定して初期分散させ，データ再分散解析の結果を基にコストに見合う場合にはデータ再分散を実施してプログラム全体のデータローカリティをより最適化させる手法をとる．

本方法では，以下に示す 6 つのステップを経て，各ターゲット配列のデータ分散形状，およびデータ分散実施手段を自動決定する．

(1) 並列化ループ，およびターゲット配列の検出

プログラム中の全ループの中から並列化ループ決定方法[6]，もしくは指示文に従って決定した手続き間並列化ループを検出する．次に，並列化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ループのループ制御変数を含む．
- ループ制御変数が，参照において 1 つの次元のみ，かつ 1 回のみ現れる．

(2) 各並列化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含む並列化ループを検出し，各ループ向けの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

データ分散形状ごとに並列化ループをグループ化し，各並列化ループのコストを静的に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．このグループの中でコストが最大の並列化ループをカーネルループと決定する．カーネルループ向けのデータ分散形状をターゲット配列のデータ分散形状と決定する．

(4) データ再分散解析

カーネルループ以外の並列化ループのうち，(2) で求めたデータ分散形状が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決定する．

(5) ファーストタッチ制御向け情報の検出

ターゲット配列ごとにカーネルループ中の代表的な参照パターンを検出し，添字式に含まれる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保持する．なお，カーネルループのループ構造に特に制約はない．

(6) データ分散実施手段の決定

ターゲット配列が以下の条件を満たす場合，FTC 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．

その他の場合は指示文による方法を選択する．なお，現在 EQUIVALENCE，COMMON には未対応である．

4. 間接参照配列向け最適化

これまでに，基本的な FTC 方法とこれを用いた自動データ分散方法の概要について説明してきた．本章では，今回提案する間接参照配列向けに FTC 方法を拡張する方式について述べる．

4.1 基本アイディア

従来の FTC 方法では，FTC コードの生成に必要な情報が定数のみで構成されている場合に FTC ループを実行文の先頭に挿入することで対応していた．FTC コードの生成に必要な情報（以下，パラメータと呼ぶ）とは以下を指す．

- カーネルループ中のターゲット配列の代表的な参照における添字式を構成する項
- カーネルループのループ制御変数の上限式，下限式を構成する項

しかし，パラメータに変数を含む場合，パラメータの値が確定する位置まで FTC ループを挿入することができない．間接参照は，添字式に配列を含むためパラメータに変数を含むケースにあてはまる．パラメータに変数を含む場合，以下に示すプログラム変換を行いファーストタッチ制御を実現する．

まず，変数パラメータの値が確定する位置（以下，変数パラメータ確定位置と呼ぶ）が，ターゲット配列の初期化ループより先行する場合，変数パラメータ確定位置の直後に FTC ループを挿入する．次に，初期化ループが変数パラメータ確定位置より先行する場合，以下のプログラム変換を行う．

- ターゲット配列と同じ宣言形状の配列（以下，クローン配列と呼ぶ）を宣言する．
- 初期化ループと変数パラメータ確定位置の間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．
- 変数パラメータ確定位置の直後にターゲット配列の FTC ループを挿入する．
- FTC ループの直後にクローン配列の全要素の値をターゲット配列へコピーするコードを挿入する．

4.2 方 式 説 明

まず，本方式のアルゴリズム概要を**図 1** に示す．図 1 の (1) から (6) は 3 章の各処理に対応する．14 行目までに示した既存の処理が終了した時点で，ターゲット配列，カーネルループ，FTC コード生成に必要な情

```
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                                    ┘(1)
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               ┐(2)
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                      │(3)
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                                      ┐(4)
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               ┘(5)
14:  ・ファーストタッチ制御コード再現用情報の検出
15:  if(ターゲット配列が間接参照配列である) then              ┐
16:    ・初期化ループを検出（以後、Ｉループと呼ぶ）           │
17:    ・変数パラメータの値が確定する位置を検出             │
         （以後、Ｖコードと呼ぶ）                           │
18:    if(Ｉループが Ｖコードより先行する) then             │
19:      ・FTC コード種別として“clone”を設定                │(a)
20:    else                                                 │
21:      ・FTC コード種別として“basic”を設定                │
22:    endif                                                │
23:    if((FTC コード種別が“basic”) .and.                  │
24:       (Ｉループを FTC ループ化可能)) then               │
25:      ・FTC コード種別として“i2ftc”を設定                │
26:    endif                                                │
27:  endif                                                  ┘
28:  ・データ分散実施手段の判定                             ┐(6)
29:  if(データ分散実施手段が FTC 方法である) then           ┐
30:    if(FTC コード種別が“i2ftc”) then                     │
31:      ・Ｉループの FTC ループ化変換                      │
32:    else if(FTC コード種別が“basic”)                     │
33:      ・Ｖコードの直後に FTC ループを挿入                │
34:    else if(FTC コード種別が“clone”)                     │(b)
35:      ・ターゲット配列のクローン配列を宣言               │
36:      ・Ｉループ、Ｖコード間のターゲット配列の参照を     │
           クローン配列の参照にリネイム                     │
37:      ・Ｖコードの直後に FTC ループを挿入                │
38:      ・FTC ループの直後にクローン配列の全要素の         │
           値をターゲット配列にコピーするコードを挿入       │
39:    endif                                                │
40:  else if(データ分散実施方法が指示文による方法である)   │
41:    ・データ分散指示文を挿入                             │
42:  endif                                                  ┘
43:endfor
```

図 1 アルゴリズム
Fig. 1 Algorithm.

報などが解析済みとなる．次に，15 行目以降で図 1 (a) に示した間接参照配列向け解析処理を行う．

4.2.1 間接参照配列向け解析

(1) 間接参照配列の検出

以下の条件を満たす配列を間接参照配列として検出する．

- 参照における添字式に 1 回のみ配列を含む．
- 添字配列の添字式において，1 つの次元のみ，かつ 1 回のみ並列化ループのループ制御変数を含む．
- 添字配列の添字式が A*i+B である（A，B は定数，i はループ制御変数）．

また，以下の条件を満たす場合も間接参照配列と見なし検出する．

- 参照における添字式にループ制御変数を含む．
- そのループ制御変数の上限式，もしくは下限式に配列を含む．
- 上下限式に含まれる配列の添字式に並列化ループのループ制御変数を含む．

(2) 初期化ループの検出

各ターゲット配列について実行時に最初に参照される初期化ループを検出する．ターゲット配列が手続きローカル配列の場合は，カーネルループより先行する文を検索し，ターゲット配列の定義文を含む最も先行するループを初期化ループとして検出する．なお，先行する文とは実行時に先に実行される文を指す．ターゲット配列が引数配列の場合は，まず，初期化ループ候補の検出を行う．

- カーネルループを含む手続き中のカーネルループより先行する文
- カーネルループを含む手続きを呼び出す上位手続き中の，カーネルループを含む手続きへのコールサイトより先行する文
- 上記の文から呼び出される手続きの全文

を検索対象とし，各手続きで最も先行するターゲット配列の定義文を含むループを初期化ループ候補として検出する．なお，各手続きにおいて初期化ループ候補が存在した場合は，それより後続する文は検索対象から除く．次に，実行順に手続きをたどり，最初に実行される候補を初期化ループと決定する．また，後述の変換コード種別判定において，初期化ループが条件下にある場合は，制御フロー上の分岐地点を初期化ループの位置とする．

(3) 変数パラメータ確定位置の検出

変数パラメータごとにカーネルループへ到達する定義位置を検出し，全変数パラメータの値が確定する位置を決定する．変数パラメータが手続きローカル変数の場合は，カーネルループより先行する文を検索し，変数パラメータの定義文のうち最も後続の文を変数パラメータ確定位置とする．変数パラメータが引数配列の場合は，まず，変数パラメータ確定位置候補の検出を行う．

- カーネルループを含む手続き中のカーネルループより先行する文
- カーネルループを含む手続きを呼び出す上位手続き中の，カーネルループを含む手続きへのコールサイトより先行する文
- 上記の文から呼び出される手続きの全文

を検索対象とし，各手続きで最も後続の変数パラメータの定義文を変数パラメータ確定位置候補として検出する．なお，各手続きにおいて変数パラメータ確定

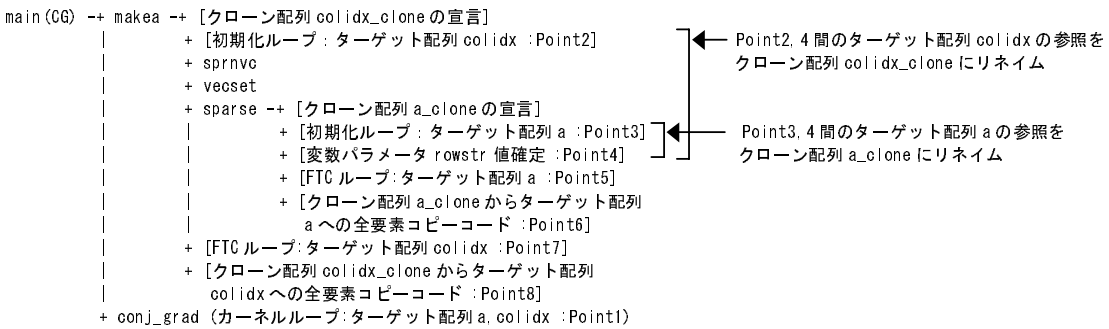

図 2 CG の変換例
Fig. 2 Example of transformation for CG.

位置候補が存在した場合は，それより先行する文を検索対象から除く．次に，実行順に手続きをたどり，最後に実行される候補を変数パラメータ確定位置と決定する．また，後述の変換コード種別判定において，変数パラメータ確定位置が条件下にある場合は，制御フロー上の結合地点を変数パラメータ確定位置とする．

(4) FTC コード種別の決定

初期化ループと変数パラメータ確定位置の実行順序関係を判定し，初期化ループが先行する場合，FTC コード種別をクローン型 “clone” とする．初期化ループが後続するループの場合，FTC コード種別を基本型 “basic” とする．また，基本型が選択された場合で，初期化ループを FTC ループ化可能と判定できた場合，FTC コード種別を初期化ループ FTC 化型 “i2ftc” とする．

4.2.2 データ分散実施コードの生成

次に，データ分散実施方法として FTC 方法が選択され，FTC コード種別が初期化ループ FTC 化型 “i2ftc” であった場合，初期化ループの FTC ループ化変換を行う．FTC コード種別が基本型 “basic” の場合，ターゲット配列の FTC ループを変数パラメータ確定位置の直後に挿入する．FTC コード種別がクローン型 “clone” の場合，以下の処理を行う．

- ターゲット配列のクローン配列を宣言する．
- 初期化ループ，変数パラメータ確定位置間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．
- 変数パラメータ確定位置の直後にターゲット配列の FTC ループを挿入する．
- FTC ループの直後にクローン配列の全要素の値をターゲット配列にコピーするコードを挿入する．

4.3 ベンチマークを用いた変換例

ここで NPB2.3serial/CG[8] を用いて，本方式を適用した場合，実際に実プログラムでどのようなプログラム変換が行われるかを説明する．まず，**図 2** に CG のコールグラフの概要を示す．CG では，2 つのターゲット配列 a，colidx が存在し，各々のカーネルループが手続き conj_grad 中に存在する．また，ターゲット配列の添字配列として rowstr が参照される．

まず，ターゲット配列 a について考える．初期化ループは，手続き sparse 中の Point3 にあり，変数パラメータ rowstr の確定位置は Point4 にある．両者のうち，初期化ループが先行するため，FTC コード種別としてクローン型が選択される．したがって，まず手続き sparse でクローン配列 a_clone を宣言する．次に，Point3 から Point4 の間のターゲット配列 a の参照をクローン配列 a_clone にリネイムする．最後に，Point4 の直後にターゲット配列 a の FTC ループを挿入し，その直後にクローン配列 a_clone からターゲット配列 a への全要素コピーコードを挿入する．ターゲット配列 colidx については，初期化ループが手続き makea に存在するため，手続き sparse へのコールサイトを変数パラメータ rowstr の確定位置とし，手続き makea においてターゲット配列 a と同様のプログラム変換を行う．

5. 評　　価

今回設計した間接参照配列向け FTC 方法の効果を検証するため，NASA 提供の標準的ベンチマークプログラム NPB2.3serial/CG[8] に人手で FTC 方法を適用してプログラム変換し，実行時間の測定を行った．また，スケーラビリティ向上の原因を明らかにするため，データローカリティに着目した CG アルゴリズムの分析を行った．

5.1 評 価 結 果

測定には，32 プロセッサ構成の Origin2000 を用いた．その他の測定環境は**表 1** に示す．評価では，以下の 3 つのデータ分散方法の性能比較を行った．

- co-DIR：本拡張適用前の性能（逐次ソースプロ

表 1 測定条件
Table 1 Measurement conditions.

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 (11,264 MB)
OS	IRIX6.5.4
コンパイラ	MIPSPro Fortran90 (Version7.3)
コンパイルオプション	-O3 -mp

グラム+OpenMP 指示文+データ分散指示文）
- hand-FTC：本拡張適用後の性能（逐次ソースプログラム+OpenMP 指示文+FTC コード）
- os-FT：OS のファーストタッチ方式データ分散による性能（逐次ソースプログラム+OpenMP 指示文）

まず，現在実装済みのコンパイラで出力することができるプログラムの性能を測定した（co-DIR）．現状では，間接参照配列に対してデータ分散指示文を挿入することができる．次に，本論文で提案した拡張を適用し，手動で間接参照配列に対する FTC コードを挿入したプログラムの性能を測定した（hand-FTC）．この hand-FTC と co-DIR の性能差が，本論文で提案する拡張の効果となる．最後に，参考のためデータ分散の制御に関わるコードが挿入されていないプログラムを OS のファーストタッチ方式データ分散に任せて実行させた場合の性能を測定した（os-FT）．なお，測定は class A，class B の 2 つのデータサイズで実施した．

(1) class A

16 プロセッサまでの実行時間の推移を**図 3** に，os-FT の 1 プロセッサ時の性能に対する各方法のスケーラビリティを**図 4** に示す．図に示すように，データ分散方法にかかわらず全ケースで同様に良好なスケーラビリティを得た．これは，データサイズが L2 キャッシュサイズ（4 MB）と比べてあまり大きくないため，データの多くがキャッシュに載ることが原因と考えられる．すべてのケースでほぼ同等のスケーラビリティを得ているものの，FTC 方法以外では性能向上のペースにばらつきがある．それに比べ，FTC 方法はつねに安定して最も良いスケーラビリティを示している．

また図 3 より，3 プロセッサから 6 プロセッサの間で比較的各方法間の性能差があることが分かる．これは，このプロセッサ数の範囲では L2 キャッシュに載るデータの割合が低いため，キャッシュミスが多発してメモリ参照が増大し，データ分散形状の影響が現れたものと考えられる．**表 2** に 1 プロセッサあたりのデータサイズの概算を示す．

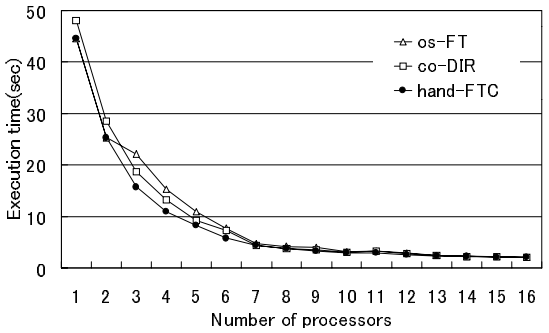

図 3 実行時間（class A）
Fig. 3 Execution time (class A).

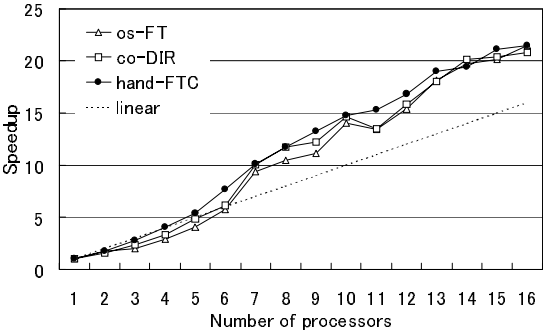

図 4 性能向上比（class A）
Fig. 4 Speedup (class A).

表 2 1 プロセッサあたりのデータサイズ（MB）
Table 2 Data size per one processor (MB).

class		A	B
ターゲット配列 (a,colidx)	要素数	2,198,000	15,825,000
	バイト数	35.2	253.2
プロセッサ数	1	70.4	506.4
	2	35.2	253.2
	4	17.6	126.6
	8	8.8	63.3
	16	4.4	31.6

CG は，データローカリティが性能に大きく影響を及ぼすターゲット配列を 2 つ有し，その他のデータのサイズが比較的小さいことから，プログラム全体で利用するデータサイズの概算は約 70 MB となる．したがって，1 プロセッサあたりのデータサイズは表 2 のようになる．表から，8 プロセッサでデータの約 50%が L2 キャッシュに載り，プロセッサ数の増大とともに 100%に近づくことが分かる．一方，3 プロセッサから 6 プロセッサの間では，データの多くがキャッシュから溢れることが分かる．それにより，メモリ参照時間の影響を受けリモートメモリ参照の割合が性能差を生じさせたと考える．os-FT のスケーラビリティを 1

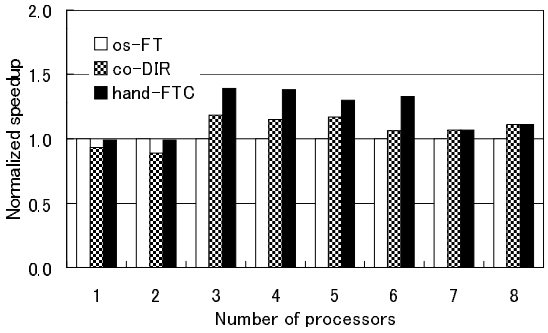

図 5 性能向上比（classA）
Fig. 5 Speedup (classA).

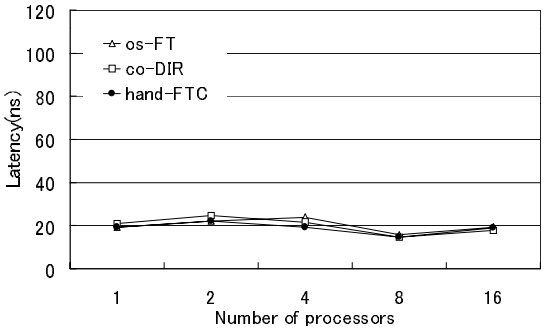

図 8 平均レイテンシ（classA）
Fig. 8 Average latency (classA).

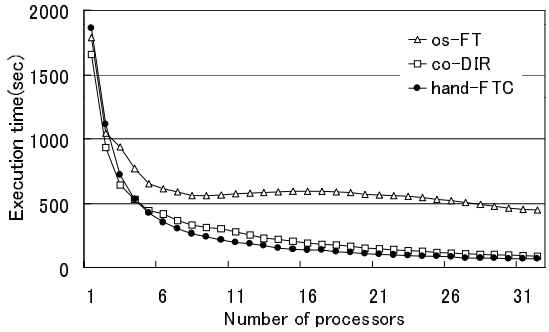

図 6 実行時間（class B）
Fig. 6 Execution time (class B).

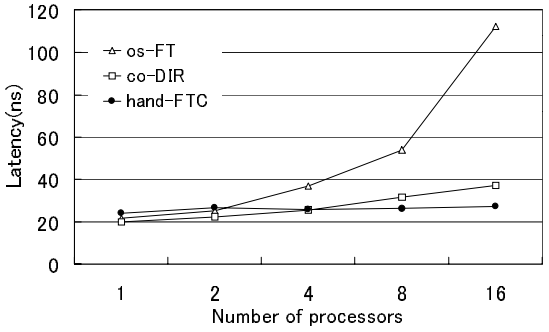

図 9 平均レイテンシ（classB）
Fig. 9 Average latency (classB).

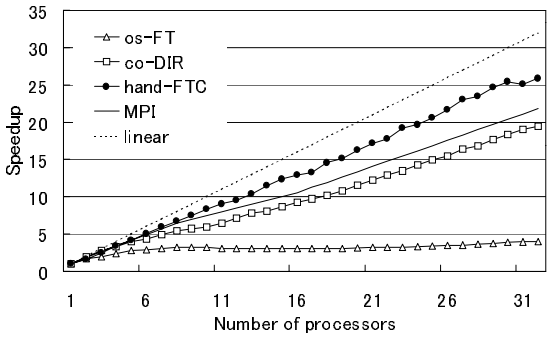

図 7 性能向上比（class B）
Fig. 7 Speedup (class B).

としたときの他の方法との比を**図 5** に示す．測定の結果，FTC 方法が 3 プロセッサから 6 プロセッサの範囲で co-DIR に比べて平均 18%，os-FT に比べて平均 35%高速であった．

(2) class B

32 プロセッサまでの実行時間の推移を**図 6** に，os-FT の 1 プロセッサ時の性能に対する各方法のスケーラビリティを**図 7** に示す．グラフに示すように，os-FT のスケーラビリティは 4 倍程度で頭打ちとなった．データ分散指示文を挿入することで（co-DIR）大きく改善することができるが，FTC 方法を適用することでさらにスケーラビリティを改善することができた．32 プロセッサまでリニアに近い順調なスケーラビリティを得，32 プロセッサ時で co-DIR に比べて 1.3 倍，os-FT に比べて 6.5 倍の性能を得た．また，参考のため MPI プログラムとの比較を行う．図 7 に示すように FTC 方法では，分散メモリ型並列計算機上では最も高速であると思われている MPI プログラムを上回るスケーラビリティを得ることができた．原因については，5.2 節で考察する．

class B の場合，表 2 に示すようにプログラム全体で利用するデータサイズの概算が約 500 MB となり，32 プロセッサまでの範囲では 1 プロセッサあたりで扱うデータのサイズが L2 キャッシュサイズよりかなり大きくなる．したがって，図 6，7 に示すようにデータ分散方法によって，大きく性能に差がついたものと考える．スケーラビリティ向上の原因がデータローカリティ改善の効果であることを検証するため，ハードウェアカウンタを用いて平均レイテンシの測定を行った．class A，class B の結果をそれぞれ**図 8**，**9** に示す．図 8 に示した class A では，データの多くが L2 キャッシュに載るため，プロセッサ数が増大しても全

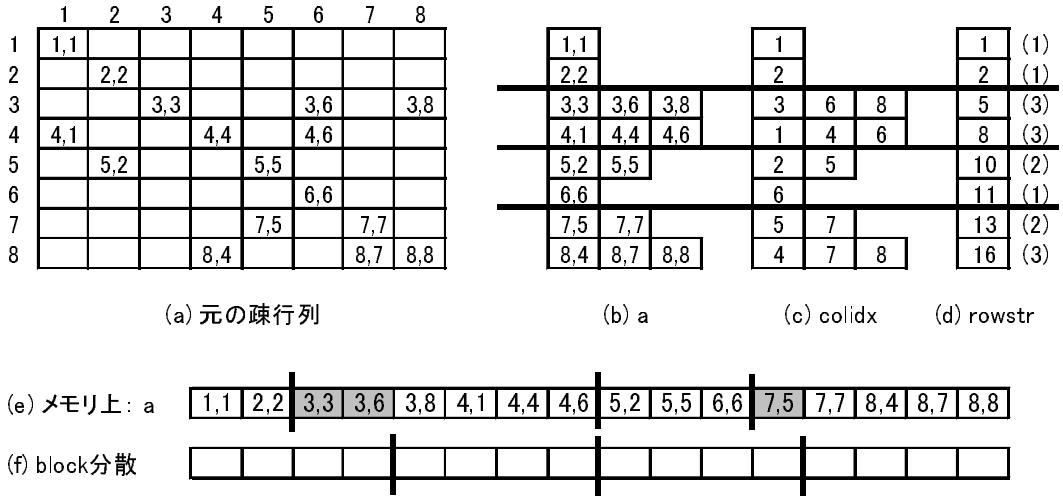

図 10 CG のターゲット配列のデータ割付け
Fig. 10 Data distribution of target array for CG.

方法でつねに平均レイテンシは 20 nsec 前後と高速に維持されている．一方，図 9 に示した class B では，os-FT についてはプロセッサ数の増加にあわせて著しく平均レイテンシが増大している．FTC 方法では，つねに 20 nsec 付近で安定しており平均レイテンシ増大のペースは鈍い．この結果から，FTC 方法によるスケーラビリティ向上の原因がデータローカリティ改善の効果であることを確認できた．

また，class B でも 100 プロセッサを超えれば L2 キャッシュにデータの多くが載りはじめ，データ分散方法にかかわらず良好なスケーラビリティを得られると予想される．しかし，プロセッサ数，キャッシュサイズといった計算機資源環境の影響を受けずにつねに安定して良好なスケーラビリティを得ることは重要である．FTC 方法を適用すれば，最適なデータ分散形状を実現することにより，これが可能となる．

5.2 考　　察

(1) データローカリティ改善の原因

間接参照配列に FTC 方法を適用した動機は，間接参照のようにアクセスパターンが複雑になる可能性がある場合，最適なデータ分散の形状自体も複雑になる可能性があり，データ分散形状が複雑になれば FTC 方法が有利であると考えたためである．今回の実験で CG に FTC 方法を適用した結果，ねらいどおり従来方法よりスケーラビリティを向上させることができた．ここで，FTC 方法がデータ分散指示文による方法や MPI プログラムより高いスケーラビリティを得ることができた理由について，CG アルゴリズムを分析して考察する．

```
1:   !$omp parallel do
2:   do j = 1, n
3:      do k = rowstr(j), rowstr(j+1)-1
4:         sum = sum + a(k) * p(colidx(k))
5:      enddo
6:      q(j) = sum
7:   enddo
```

図 11 CG のカーネルループ
Fig. 11 Kernel loop of CG.

まず，**図 11** に CG のカーネル部分の典型的なアクセスパターンを示す．データローカリティが性能に影響を及ぼす重要な配列は，a と colidx の 2 つである．図 11 のカーネルループでは，内側ループのループ制御変数 k の上限，下限に配列 rowstr が現れるため，内側ループ本体で参照される配列 a，colidx の最初と最後の参照は間接参照となる．

次に，**図 10** を用いて CG のアルゴリズムについて簡単に説明する．CG では，元の疎行列（図 10 (a)）の非零要素だけをいったん配列 a（図 10 (b)）にコピーし，主にこの配列 a とインディックス情報を持つ配列 colidx（図 10 (c)），要素数が設定された配列 rowstr（図 10 (d)）の 3 つの配列を利用して図 11 に示したアクセスパターンに従って演算が行われる．したがって，配列 a を例に説明すると，図 10 (e) に示すように 1 次元配列として (1,1)，(2,2)，(3,3)，(3,6)，(3,8) の順でアクセスされることになる．よって，メモリ上連続した領域のアクセスとなり，L1 キャッシュの利用効率の良いプログラムになっていることが分かった．さらに，カーネルループ中では，ターゲット配列の参照

は使用参照のみであったため，データがいったん L2 キャッシュに載ってしまえば，メモリ上のデータ分散形状にかかわらず良好なスケーラビリティを得られることが分かった．CG の場合，class A 以下のデータサイズがこの対象となる．ところが，L2 キャッシュにおいてキャッシュミスが多発しメモリ参照が増大した場合，最適なデータ分散形状が実現できているか否かで性能が左右される．class B 以上でこの影響が顕著となり，図 6，7 に示したように本拡張前の co-DIR および os-FT では，十分なスケーラビリティを得ることができなかった．

ここで，メモリ上のデータ分散形状が性能に影響を及ぼす class B 以上の十分大きいデータサイズについて考えてみる．カーネルループは外側ループで並列化されるため，ループスケジューリング方式を block 分散，プロセッサ数を 4 と仮定すると，3 つのターゲット配列（図 10 (b)，(c)，(d)）は，縦軸方向を太線で示したように各ノードに均等に割り付けた状態で参照されることになる．したがって，ターゲット配列 a のカーネルループ向けの最適なデータ分散形状は，図 10 (e) に示すような不規則な形状となる．なおかつ，この形状は実行時に決定する．不規則な形状を指示文で指定することは大きな工数を必要とする．また，実行時に決定する形状を指示文で指定することは困難である．仕方なく，たとえば指示文で block 分散を指定したと仮定すると，分散形状は図 10 (f) に示すようになり，カーネルループ向けのデータ分散形状である図 10 (e) との差分，色付きの部分がリモートメモリ参照となる．FTC 方法を適用すれば，100%ローカルメモリ参照となるので，このリモートメモリ参照の割合が，32 プロセッサ時で 1.3 倍という指示文版との性能差につながったと考える．また，MPI プログラムについては，元々複雑なデータ分散形状をベースにしたコーディングは工数上困難であり，あまり行われない．ソース分析の結果，MPI 版のデータ分散形状は block 分散であることが分かり，MPI 版でも指示文版と同様の理由によるデータローカリティの差が生じたと考えられる．

(2) 検証

以上の考察を検証するため，データ要素の割付け先ノードを問い合わせるシステムコールを用いてローカルメモリ参照の割合を測定した．結果を**表 3** に示す．OS のファーストタッチ方式データ分散（os-FT）では，初期化ループが並列化できないため，ローカルメモリ参照の割合がプロセッサ数分の 1 しか得られない．データ分散指示文の場合（co-DIR）は，大きくデータローカリティを改善するが，説明してきたような理由により，プロセッサ数の増大とともに図 10 (e) に示したズレの部分が増大して十分なデータローカリティを得ることができない．FTC 方法によれば，プロセッサ数が増えても一定して 100%近くのローカルメモリ参照率を得られることが確認できた．

表 3 ローカルメモリ参照率（%）
Table 3 Rate of local memory reference (%).

Data Distribution Method	Number of processors				
	1	2	4	8	16
os-FT	100	50	25	12	5
co-DIR	100	90	73	42	20
hand-FTC	100	99	98	98	98

(1processor/node)

本節では，FTC 方法が他の方法に比べて良いデータローカリティを実現する理由を示した．分析の結果，CG は FTC 方法に適したアルゴリズムであったことが分かった．しかし，一般の間接参照では，メモリ上に隣接するデータを異なるプロセッサが参照する事態が多発し，十分なデータローカリティを得られない場合が生じることも予想される．したがって，間接参照向け最適化の汎用性を向上させていくためにも，様々なプログラムパターンを FTC 方法に適した方向に誘導していく最適化方法の開発が必要であると考える．1 つの方法は，連続したアクセスデータをメモリ上に隣接した領域に配置する方法を探ることである．今後は，バッファ配列を利用した最適化方法，およびプロファイル情報の利用などの検討を進める予定である．

6. 関連研究

データ分散形状を自動的に決定する方法は，これまでにも多数行われている．最適なデータ分散形状を決定する問題は NP 完全であるため，Kennedy ら[2]，Gupta ら[3]，辰巳ら[4]，松浦ら[5] などが，近似解を求める様々なヒューリスティクを提案した．辰巳ら[4] は，単独のループを対象として，配列間の相対的な配置関係から分散次元とブロックサイズを決定する方法を示した．松浦ら[5] は，手続きにまたがる複数ループを対象として，手続き間の配列アクセス情報を基に通信を最小化したデータ分散を決定する方法を示した．Gupta ら[3] は，CC-NUMA 向けの手続き間を対象としたループ，およびデータの分散方法を示した．いずれの方法も指示文を用いたデータ分散方法を想定しており，

- 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
- 手続きごとに引数配列の宣言形状が異なる場合

- カーネルループに間接参照が存在する場合

などに最適なデータ分散を実現する方法については論じられていない．提案方法では，これらが正確に実現可能となり，複数ループ，複数手続きに参照のまたがる配列のデータ分散を自動決定することができる．

また，ファーストタッチ制御方法に関連して Bircsak ら[9] が，直後のループ向けデータ分散を指示する next-touch 指示文を提案している．この指示文を利用することにより本論文で示した内容と同様の効果を得る場合がある．しかし，以下の点が提案方法と大きく異なる．

- 提案方法は，コンパイラで自動化している．
- next-touch 指示文を利用した場合，指示文挿入位置より以前に対象配列の参照がある場合はつねにデータ再分散が発生する．しかし，提案方法では，初期化ループが変数パラメータ確定位置より後で実行される場合やパラメータが定数で構成される場合，データ再分散に相当する処理（クローン配列からターゲット配列への全要素コピー）は発生しない．
- 提案方法では，next-touch 指示文をサポートしたノードコンパイラが不要である．

7. まとめ

本論文では，間接参照配列向け FTC 方法の設計と評価について述べた．Origin2000 を用いた評価の結果，今回設計した機能により，ベンチマークプログラム NPB2.3 serial/CG で，class B の場合，32 プロセッサ時，データ分散指示文による方法に比べて 1.3 倍に性能が向上した．また，OS のファーストタッチ方式データ分散による方法に比べて 6.5 倍に性能が向上した．class A の場合，3 プロセッサから 6 プロセッサの間でデータ分散指示文による方法に比べて平均 18%，OS のファーストタッチ方式データ分散による方法に比べて平均 35%性能が向上した．平均レイテンシを測定し，FTC 方法適用によるスケーラビリティ向上の原因は，データローカリティ改善の効果であることを確認した．次に，CG で FTC 方法が他の方法より高いデータローカリティを実現する理由を明らかにし，ローカルメモリ参照率を測定して検証した．

プロセッサ数やキャッシュサイズといった計算機資源環境による性能のばらつきを抑え，つねに安定して良好なスケーラビリティを得ることは重要である．それにはデータローカリティ最適化が重要であることを示した．FTC 方法を適用すれば，データローカリティの改善を実現し，この点に貢献できることを示した．また，データローカリティの向上には，各プログラムのアルゴリズムまで考慮に入れたコンパイラレベルの最適化が重要であることを示した．

今後の課題としては，

- 間接参照配列向け FTC 方法の実装
- 間接参照配列向け最適化の一般化，および拡張
- より多くのベンチマークによる検証

などがあげられる．

謝辞 本研究は，新情報処理開発機構（RWCP）における成果を基に行ったものである．また，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アドバンスト並列化コンパイラ研究体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) Kennedy, K. and Kremer, U.: Automatic Data Layout for High Performance Fortran, *Proc. Supercomputing'95* (1995).
3) Gupta, M. and Banerjee, P.: PARADIGM: A Compiler for Automatic Data Distribution on Multicomputers, *Proc. ICS'93*, pp.87–96 (1993).
4) 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).
5) 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，情報処理学会論文誌，Vol.41, No.5, pp.1420–1429 (2000).
6) 青木雄一郎，佐藤真琴，飯塚孝好，佐藤茂久，菊池純男：手続き間自動並列化コンパイラ WPP の試作—実機性能評価，情報処理学会研究報告，98-ARC-130, pp.43–48 (1998).
7) 廣岡孝志，太田 寛，菊池純男：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法，情報処理学会論文誌，Vol.41, No.5, pp.1430–1438 (2000).
8) The NAS Parallel Benchmarks. http://www.nas.nasa.gov/Software/NPB/
9) Bircsak, J., Craig, P., Crowell, R., Cvetanovic, Z., Harris, J., Nelson, C.A. and Offner, C.D.: Extending OpenMP For NUMA Machines, *Proc. Supercomputing'2000* (2000).

(平成 14 年 1 月 15 日受付)
(平成 14 年 5 月 17 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．