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

廣岡孝志[†] 太田寛[††]

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方法の実装を行った．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．これに手続き間解析機能を搭載し，プログラム全体の解析結果に基づくデータローカリティ最適化を実現した．SGI/Origin2000 を用いた評価の結果，ベンチマークプログラム NPB2.3serial/FT，SP，CG，SPECfp95/tomcatv の 4 題について，本分散方法を行わない場合に比べて 16 プロセッサ時で平均 35.3%性能が向上することを確認した．さらに，CG 中の間接参照配列に対して人手でファーストタッチ制御方法を適用することにより，本分散方法を行わない場合に比べて 6.0 倍，MPI プログラムに比べて 1.2 倍に性能が向上することを確認した．

Implementation and Evaluation of Interprocedural Automatic Data Distribution Method for Distributed Shared Memory

Takashi Hirooka[†] and Hiroshi Ohta[††]

We implemented an interprocedural automatic data distribution method 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. In addition we implemented interprocedural analysis which improves data locality of the whole program. We evaluated NPB2.3serial/FT, SP, CG and SPECfp95/tomcatv on SGI/Origin2000. These benchmarks ran faster by 35.5% (average) than those without our method in the case of 16 processors. Furthermore, we applied FTC to indirect array references of CG by hand and it ran faster by 6.0 times than that without our method and by 1.2 times than MPI program.

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．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，前報告[18] では最適なデータ分散形状をコンパイラで自動的に決定する方法を提案した．本報告では，それを実装，評価した内容について述べる．

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[1]，OS が行うファーストタッチ方式データ分散[1] などが研究されてきた．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部分配列参照（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，前報告[18] 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方法を提案した．本報告では，このファーストタッチ制御方法を適用した自動データ分散方法の実装，およびベン

† 株式会社日立製作所システム開発研究所
Systems Development Laboratory, Hitachi Ltd.
†† 株式会社日立製作所情報コンピュータグループ
Information & Computer Systems, Hitachi Ltd.

チマークプログラム NPB2.3serial[19]/FT，SP，CG，SPECfp95[20]/tomcatv を用いた性能評価について述べる．本研究で提案，実装した自動データ分散方法は，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られる点に特徴がある．加えて，手続き間解析機能の搭載によりプログラム全体の解析結果に基づくデータローカリティ最適化を実現した．

本論文の構成は以下のとおりである．2 章では，分散共有メモリ機構について説明した後，実装した DSM 自動並列化コンパイラの構成を述べる．3 章では，前報告[18] で提案した DSM 向け手続き間自動データ分散方法の概要を述べる．4 章で評価方法を示し，5 章で評価結果を分析する．6 章で関連研究を紹介し，7 章で結論を述べる．

2. システム構成

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，各ノードの物理メモリに割り付ける方法がある．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 もこれを採用している．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，自ノードに割り付けられたデータの参照の割合，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]．

(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．

上記従来方法では，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や，手続きごとに引数配列の宣言形状が異なる場合に，最適なデータ分散が実現できないという問題点があった．前報告[18] では，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．**図 1** を用いて簡単に説明する．

図 1 (a) に示した例題プログラム 1 の場合，従来方法では図 1 (b) に示すようなデータ分散指示文をプログラムの先頭に挿入していた．提案方法では，コンパイラが図 1 (c) に示すような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（以後，この一連のコードをファーストタッチ制御コードと呼ぶ）．なお，本論文では，上記ファーストタッチ制御コードとデータ分散指示文の総称をデータ分散実施コードと呼ぶ．

次に，実装した DSM 向け手続き間自動データ分散方法のシステム構成を**図 2** に示す．本自動データ分散技術は，既存の SMP 向け自動並列化コンパイラ WPP[17] にデータ分散部を挿入することで実現した．以後，本コンパイラを DSM 自動並列化コンパイラと呼ぶ．DSM 自動並列化コンパイラは，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（OpenMP 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．さらに，SGI が DSM 向けに独自にデータ分散指示文をサポートした OpenMP コンパイラが，このソースプログラムを入力として DSM 並列オブジェクトプログラムを出力し，並列実行を実現する．DSM 型並列計算機としては Origin2000 を用い，OpenMP コンパイラとしては Origin2000 に搭載された MIPSpro Fortran90 コンパイラを用いた．

データ分散部では，まず処理分散部[17] で決定した並列化ループを検出し，このループ本体に参照を有する配列（以後，ターゲット配列と呼ぶ）を検出する．次にターゲット配列ごとに各並列化ループ向けデータ分散形状を決定し，プログラム中で最も実行頻度が高いループ，すなわちカーネルループ，およびターゲット配列のデータ分散形状を決定する．また，このときデータ再分散解析を行い，ターゲット配列のデータ再分散形状を決定する．次に提案方法の特徴技術の 1 つであるファーストタッチ制御データ分散を適用するために必要な解析を行い，最後にデータ分散実施手段の決定を行う．

3. DSM 向け手続き間自動データ分散方法

3.1 概　　要

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，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．したがって，手続きに跨って参照される配列のデータ分散について考えると，その配列が初め

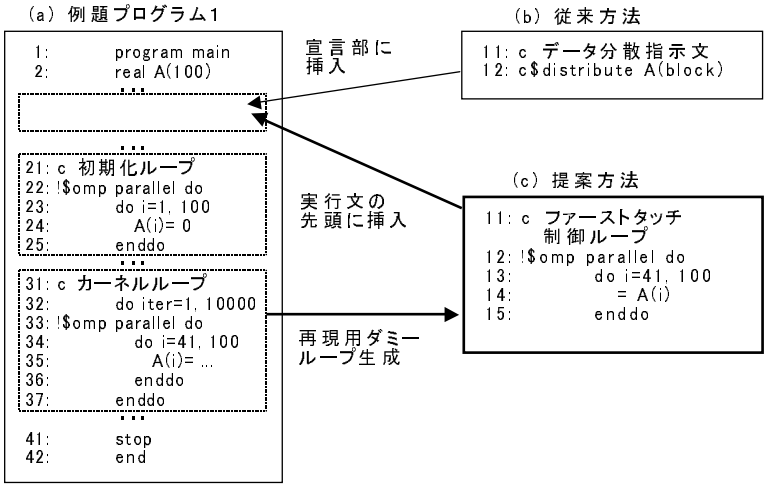

図 1 ファーストタッチ制御方法
Fig. 1 First touch control method.

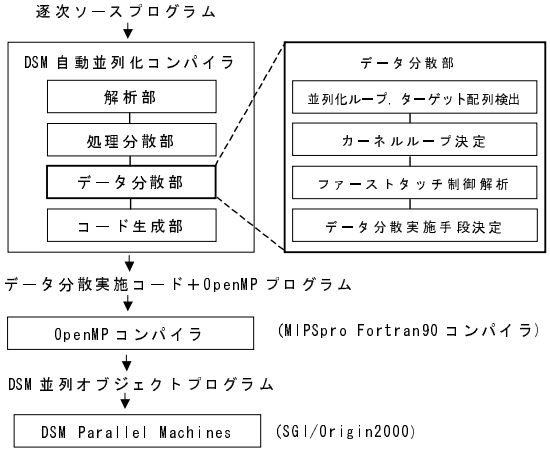

図 2 DSM 自動並列化コンパイラの構成
Fig. 2 Structure of DSM automatic parallelizing compiler.

```
1:   do i = 1, 100                    ;L1
2:      do j = 1, 100                 ;L2
3:         A(i,j) = …
4:      enddo
5:   enddo
     …
11:  do iter = 1, 10                  ;L3
12:     do i = 2, 50                  ;L4
13:        do j = 2, 99               ;L5
14:           B(j,i) = A(j+1,i+1)
15:                  + A(j+1,i-1)
16:                  + A(j-1,i-1)
17:        enddo
18:     enddo
19:  enddo
```

図 3 例題プログラム 2
Fig. 3 Example program2.

て参照される手続きで決定したデータ分散形状をプログラム全体で利用し続けるか，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．

本章では，前報告[18]で提案した自動データ分散方法の概要を示し，具体的にデータ分散形状を決定していく過程を説明する．本方法では，以下のステップを経てデータ分散形状，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
(2) 各並列化ループ向けデータ分散形状の決定
(3) 手続き内カーネルループの決定
(4) 手続き間カーネルループの決定
(5) データ再分散解析
(6) ファーストタッチ制御向け情報の検出
(7) データ分散実施手段の決定

次節で，これらを詳細に説明する．

3.2 データ分散形状，データ分散実施手段の決定

図 3 の例題プログラム 2 が，本 DSM 自動並列化コンパイラの入力となった場合の処理について説明する．図 2 の処理分散部[17]では，ループ運搬フロー依存のないループを検出し，その中からしきい値以上のループ粒度を有する最外側ループを並列化ループと決

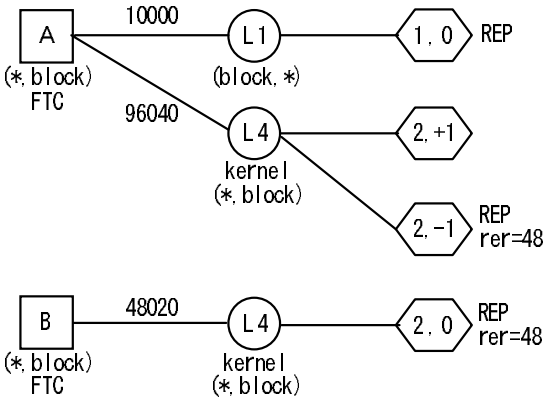

図 4 データ分散グラフ
Fig. 4 Data distribution graph.

定する．本例では，5 つのループのうち，L1 と L4 が並列化ループとして検出されたと仮定する．

本方法では，入力プログラムに対し，**図 4** に示すようなデータ分散グラフを生成し解析に用いる．グラフは，配列ノード，ループノード，参照パターンノードから構成され，エッジはノード間の関係の有無を表す．なお，本グラフは配列ごとに生成される．図 3 と図 4 を用いて解析の過程を説明する．

(1) 並列化ループ，およびターゲット配列の検出

並列化ループ (L1,L4) を検出し，並列化ループのループ本体で参照される配列 (A,B) をターゲット配列とする．グラフでは，配列ノード A と B を生成する．

(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．なお，block 分散とはデータをプロセッサ数分に短冊状に切り分ける方法を指す．

グラフでは，まず初めに配列ノード A に対し，ループノード L1 と L4 を接続し，配列ノード B に対し，ループノード L4 を接続する．そして，各々のループノードの属性としてデータ分散形状を設定する．次に，各ループにおけるターゲット配列の参照パターンを検出し，参照パターンノードを接続する．参照パターンノード中に示した情報の 1 要素目は並列化ループのループ制御変数を含む次元，2 要素目は並列化ループのループ制御変数単独式からの距離を示す（たとえば，i+1 の場合，距離は 1）．次に，各配列，各ループごとに全参照パターンの中から以下の優先順位で代表参照パターンを決定する．

- 参照回数が最大の参照パターン
- 左辺の参照パターン
- ループ制御変数単独式との距離（絶対値）が最小の参照パターン

ここで，距離が最小のものを選択するのは，ループ制御変数単独式に近い方がカーネルループ以外の参照において有利になる可能性が高いと考えたためである．決定した代表参照パターンを示すフラグは，参照パターンノードの属性として登録する（図 4 の REP）．次に，代表参照パターンの添字式と並列化ループから各ループ向けのデータ分散形状を決定し，ループノードの属性として登録する．

(3) 手続き内カーネルループの決定

各手続きごとに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ループをグループ化し，ループ本体の演算数とループ長の積をループ演算量と定義して各並列化ループの演算量を見積もり，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演算量最大の並列化ループをターゲット配列の手続き内カーネルループとする．このカーネルループ向けのデータ分散形状（(2) で決定したもの）を，ターゲット配列の手続き内データ分散形状とする．グラフでは，配列ノードとループノード間のエッジの属性として代表参照パターンの全ループ繰返しで参照される回数を登録し，この参照回数が最大のループを手続き内カーネルループと決定する（図 4 の kernel）．

(4) 手続き間カーネルループの決定

ターゲット配列が引数の場合，引き渡され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．次に，データ分散形状ごとにプログラム全体で手続き内カーネルループをグループ分けし，演算量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演算量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．

(5) データ再分散解析

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．

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

```
1:   !$omp parallel do
2:      do i = 2, 50
3:         do j = 2, 99
4:            B(j,i) = 0
5:            A(j+1,i-1) = 0
6:         enddo
7:      enddo
```

図 5 データ分散実施コード
Fig. 5 Data distribution code.

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．

(6) 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の代表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．

(7) 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，ファーストタッチ制御方法[18] 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．

具体的には，まず宣言された全要素数に対するカーネルループ本体で参照される要素数の割合を参照要素率と定義し，代表参照パターンの参照要素率を解析する．参照要素率（図 4 の rer）が 50%以下の場合[18]，データ分散実施手段として，ファーストタッチ制御による方法（図 4 の FTC）を選択し，その他の場合，データ分散指示文による方法（DIR）を選択する．決定したデータ分散実施手段は，配列ノードの属性として登録する．グラフでは，カーネルループの代表参照パターンの参照要素率から，ターゲット配列 A，B ともに FTC が選択された．以上の解析結果に基づき**図 5** に示すようなデータ分散実施コードが生成されプログラム中の main 手続きの実行文の先頭に挿入される．また，ターゲット配列が引数の場合は，ターゲット配列の参照を有する手続きのうち，コールグラフ上最も上位の手続きの実行文の先頭に挿入する．なお，現在 COMMON 変数，および MODULE 変数には未対応である．

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 (11264 MB)
OS	IRIX6.5.4
コンパイラ	MIPSPro Fortran90 (Version7.3)
コンパイルオプション	-mp -64 -Ofast=IP27 -OPT:IEEE_arithmetic=3

4. 評価方法

4.1 測定環境

性能測定は，Origin2000 上で行った．測定条件を**表 1** に示す．Origin2000 は，1 ノード 2 プロセッサ構成であるため，以下で示す性能グラフでは 4 プロセッサ以上でデータローカリティの影響が表れる．

4.2 DSM 自動並列化コンパイラの評価

ベンチマークプログラムとして，SPEC Benchmark の SPECfp95/tomcatv，および NASA 提供の NPB2.3serial/FT，SP，CG の 4 題を用いた．DSM 自動並列化コンパイラにおける自動データ分散機能の評価として，各ベンチマークの以下の項目を明らかにする．

- 検出したターゲット配列
- 決定したデータ分散形状
- 決定したデータ分散実施手段
- スケーラビリティ

5 章でベンチマークごとに測定結果を示し，分析して本自動データ分散機能の効果を吟味する．

4.3 ファーストタッチ制御コードのオーバヘッド

データ分散実施方法としてファーストタッチ制御方法が選択された場合，ファーストタッチ制御コードによるオーバヘッドが生じる．しかし，このオーバヘッドは，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り，カーネルループの実行時間に比べて十分小さいと考えられる．本方法の適用対象となった FT を用い，各プロセッサ数におけるオーバヘッドの実行時間，および全体に占める割合を測定し，これを検証する．

4.4 データ再分散の評価

DSM が幅広い実用プログラムで十分なスケーラビリティを得るためには，データ再分散の高速化が重要

表 2 自動データ分散機能が決定した主要配列のデータ分散形状
Table 2 Data distribution shape of key arrays which our automatic data distribution function determined.

Program	Key Arrays	Data Size		Data Distribution	Method
		classA	classB		
FT	U1, INDEXMAP	256x256x128	512x256x256	(*,*,block)	FTC
	U0, U2	256x256x128	512x256x256	(*,block,*)	
SP	LHS	64x64x63x15	102x102x101x15	(*,*,block,*)	DIR
	FORCING, RHS, U	64x64x63x5	102x102x101x5	(*,*,block,*)	
	SQUARE, SPEED, RHO_I AINV, QS, WS, VS, US	64x64x63	102x102x101	(*,*,block)	
CG	COLIDX	2198000	15825000	(block)	DIR
	ROWSTR, V, X, Z	14001	75001	(block)	
tomcatv	AA, DD, X, Y, RX, RY	513x513		(*,block)	DIR
	D	513x513		(block,*)	

と予想される．データ再分散により，データローカリティが向上する FT を用いてデータ再分散の効果を検証する．

4.5 間接参照配列をターゲットに含めた場合の評価

現在の実装では，間接参照を有する配列をターゲット配列の対象から除いている．分析の結果，CG の主要配列の 1 つに間接参照を有する配列が存在することが分かったため，これを自動データ分散の対象に含めた場合の効果を検証する．また，間接参照配列に対するファーストタッチ制御方法の効果を評価するため，手動でファーストタッチ制御コードを挿入し，スケーラビリティを測定する．

5. 評価結果

5.1 DSM 自動並列化コンパイラの評価

各ベンチマークを入力とした際の本自動データ分散方法による解析結果を**表 2** にまとめる．表には，各ベンチマークの主要配列のうち，自動データ分散機能がターゲット配列として検出したものについて，そのデータサイズ，および決定したデータ分散形状とデータ分散実施手段を示す．次に，各ベンチマークの性能測定結果を**図 6** の (a) から (d) に示す．グラフは，横軸にプロセッサ数，縦軸に性能向上比を示し，本方法による自動データ分散を行った場合（dist）と行わない場合（no dist）の性能を比較したものである．なお，no dist では，OS が行うファーストタッチ方式データ分散に従う．

(1) FT

FT は，カーネルループと主要配列が比較的限定されている．本方法により，主要配列の多くが自動データ分散のターゲットとして検出され，各配列に対し，表 2 に示すようなデータ分散形状が求められた．データ分散実施手段としては，主要配列の宣言形状が手続きごとに異なることから，4 題のベンチマークで唯一ファーストタッチ制御方法（FTC）が選択された．ターゲット配列の参照を有する手続きのうち，コールグラフ上最も上位のものを最親手続きと定義すると，指示文による方法（DIR）では，カーネルループを含む手続きと最親手続きでターゲット配列の宣言形状が異なるとき，カーネルループにおいて最適となるデータ分散形状を最親手続きで指示することが困難な場合がある．よって，ファーストタッチ制御方法を適用する．

性能向上比を図 6 (a) に示す．各クラスとも，コンパイラによる自動データ分散を行わない場合，プロセッサ数の増大とともにデータローカリティが低下するため，十分なスケーラビリティを得ることができない．一方，自動データ分散機能を適用した場合，最適なデータ分散が実施され，データローカリティが改善されて大幅な性能向上を実現した．class B，16 プロセッサ時で 89%性能が向上した．また，class A と class B の比較から，データサイズが増大すると性能差が拡大する傾向にある．なお，このグラフにはファーストタッチ制御コードによるオーバヘッドも含まれている．

FT の場合，データ再分散により大きくデータローカリティが改善される部分がある．しかし，本方法ではコスト計算の結果，データ再分散は実施されなかった．これは，Origin2000 のデータ再分散コストが大きいためである．データ再分散のコストが小さくなり，データ再分散を実施してデータローカリティの改善が実現できれば，さらにスケーラビリティが向上すると考えられる．5.3 節でデータ再分散の効果を評価する．

(2) SP

SP では表 2 に示すような主要配列を検出し，表記

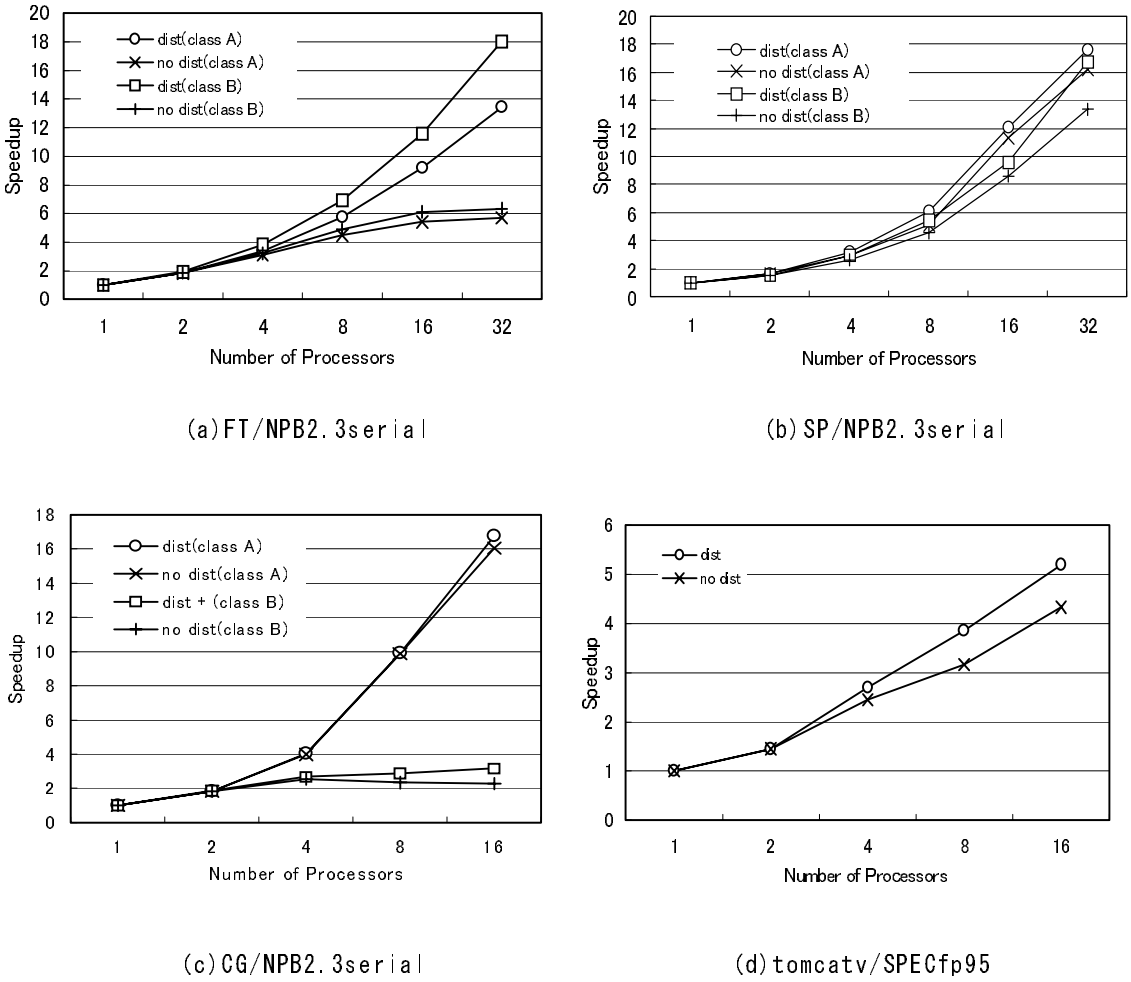

図 6 ベンチマークプログラムの性能向上比
Fig. 6 Speedup of benchmark programs.

のようなデータ分散形状を求めることができた．データ分散実施手段は，指示文による方法（DIR）が選択された．

性能向上比を図 6 (b) に示す．SP は，コンパイラによる自動データ分散を実施しない場合でも比較的良好な並列性能を示していた．しかし，本自動データ分散機能により，さらに性能を向上させることができ，class B，16 プロセッサ時で 12%の性能向上を実現した．また，データサイズが増大するとプロセッサ数の増加とともに自動データ分散機能による性能向上の割合が拡大する傾向がある．

(3) CG

CG は，カーネルループと主要配列が限定されている．また，処理分散の並列化効率が高く，データローカリティに問題がなければ良好なスケーラビリティを得ることができると予想される．本方法により，主要配列 3 つのうち 2 つがターゲットとして検出された．

性能向上比を図 6 (c) に示す．CG は，class A では主要配列のデータが各プロセッサのキャッシュに載りきるため，コンパイラによる自動データ分散機能の有無に関係なく良好な並列性能を示した．しかし，class B 以降の大きなデータサイズでは，リモート参照の割合が増大して，大きくスケーラビリティを悪化させている．コンパイラによる自動データ分散を適用した場合，少しスケーラビリティが改善し，class B，16 プロセッサ時で 39%性能が向上するが，十分な並列性能を得ることはできなかった．

分析の結果，CG では残る 1 つの主要配列が間接参照を有するため，ターゲット配列の対象から除外されていたことが分かった．また，この配列のデータローカリティが性能に大きく影響を及ぼすことが分かったため，この配列をターゲット配列に含めた場合の性能

を測定した．結果を 5.4 節で説明する．

(4) tomcatv

性能向上比を図 6 (d) に示す．tomcatv でも，自動データ分散機能により，データローカリティが改善され，16 プロセッサ時で 20%性能が向上した．

(5) 全体について

本自動データ分散方法では，処理分散解析により決定した並列化ループを用いてデータ分散を行う．処理分散の良し悪しは考慮しない．現状の処理分散でデータローカリティの影響により，十分な並列性能を出せていない場合に効果を発揮する．その意味で測定した 4 題のベンチマークは，各々この条件を満たし性能向上を得ることができた．また，本評価は OS が行うファーストタッチ方式データ分散とコンパイラによる自動データ分散の性能差を比較したものである．したがって，前者の場合，初期化ループの並列化が重要となる．たとえば，

- 初期化ループが，データ依存の存在や依存解析精度の問題のために並列化不能である．
- ターゲット配列の宣言形状が，カーネルループを含む手続きと異なる．
- 初期化ループの構造や並列化ネスト位置が，カーネルループと異なる．

といった場合，低いデータローカリティの影響でスケーラビリティを著しく悪化させるケースがある．FT，CG がこれに該当し，本自動データ分散方法の効果が大きいタイプと考えられる．

5.2 ファーストタッチ制御コードのオーバヘッド

データ分散実施手段としてファーストタッチ制御方法が選択された FT，class A を用いてファーストタッチ制御コードのオーバヘッドを測定した．**表 3** に，プロセッサ数ごとの測定結果を示す．noopt_OH に，現状コンパイラのファーストタッチ制御コードの実行時間 (msec) を示し，noopt_OH/KER に，プログラム中のカーネル部分に対する比率 (%) を示す．各プロセッサ数において，おおむね数%で推移している．これはデータローカリティ改善による性能向上の効果に比べて十分小さいと考えられる．

ここで，さらにオーバヘッドを削減する最適化を考える．ファーストタッチ制御ループでは，各ページを最適なノードに割り付けるため，ターゲット配列の全要素を参照させている．理論的には，1 ページにつき 1 要素参照させればよいはずなので，ファーストタッチ制御ループの繰返し範囲を 1 ページにつき 1 回に変更する．Origin2000 の場合，デフォルトのページサイズは 16 キロバイトなので，単純に計算しても 1 要素のデータ長が 8 バイト場合，参照回数が 2000 分の 1 に削減できる．手動でこの最適化を施したファーストタッチ制御コードの実行時間を表 3 の opt_OH(msec) に示し，カーネル部分に対する比率 (%) を opt_OH/KER に示す．元々ループ本体の演算量が少なく，スレッド起動オーバヘッドが目立つ構造であったので 2000 分の 1 にはならないが，本最適化により，数十分の 1 から 100 分の 1 程度にオーバヘッドを削減することができた．これで，オーバヘッドの問題は無視できるほど十分小さくできると考えられる．

表 3 FT（class A）の FTC オーバヘッド
Table 3 FTC overhead for FT (class A).

プロセッサ数	1	2	4	8	16	32
noopt_OH (msec)	2522	1758	1256	757	249	209
noopt_OH /KER (%)	2.11	2.71	3.50	3.62	1.93	2.35
opt_OH (msec)	26	19	9	5	5	16
opt_OH /KER (%)	0.02	0.03	0.03	0.02	0.04	0.18

OH：ファーストタッチ制御コードの実行時間
KER：カーネル部分の実行時間

5.3 データ再分散の評価

FT は，データ再分散を行うことによりデータローカリティをさらに向上させることができる．そこで，DSM 自動並列化コンパイラの解析結果ではデータ再分散が適用されなかったが，FT でデータ再分散を実施した場合の効果を評価するため，ソースプログラムに手動で以下の修正を施し測定を行った．

- 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に最適となる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，実行文の先頭に挿入する．
- データ再分散区間の直前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．
- データ再分散区間内のターゲット配列をクローン配列にリネイムする．
- データ再分散区間の直後で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．

図 7 にデータ再分散を実施した場合の実行時間の比較を示す．縦軸に経過時間，横軸にプロセッサ数を示し，no redist にデータ再分散を行わない場合，redist にデータ再分散を行った場合の性能を示す．

グラフに示すように，すべてのプロセッサ数の範囲でデータ再分散を実施することにより性能が劣化した．データ再分散のオーバヘッドがデータローカリティ改善の効果を上回ったためと考えられる．幅広い実用プ

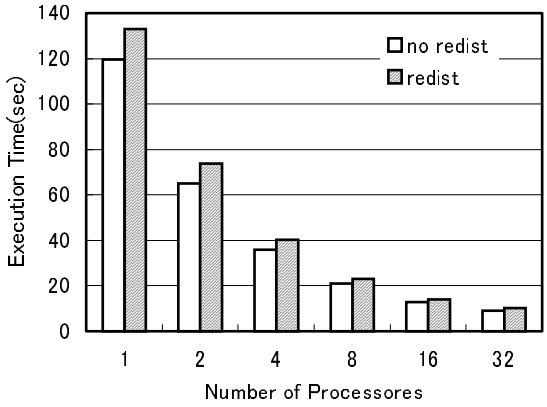

図 7 FT（class A）の実行時間
Fig. 7 Execution time of FT (class A).

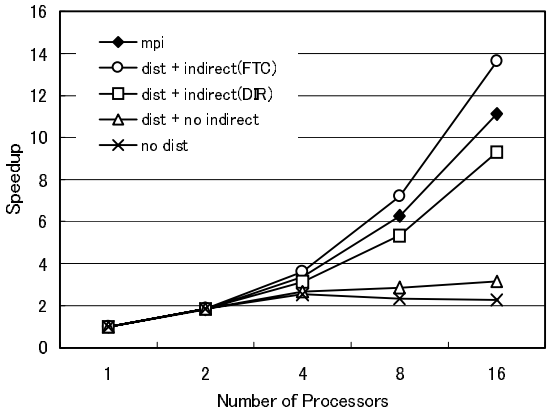

図 8 CG（class B）の性能向上比
Fig. 8 Speedup of CG (class B).

ログラムで十分な並列性能を得るためにはデータ再分散が必要であるため，データ再分散の高速化は DSM の課題の 1 つであると考えられる．

5.4 間接参照配列をターゲットに含めた場合の評価

(1) データ分散指示文の適用

CG の場合，ソースプログラムの分析の結果，主要な配列の中に間接参照を有する配列を含むことが分かった．しかし，現在の実装では，これを自動データ分散の対象外としていた．間接参照配列を対象に含めた場合の効果を評価するため，DSM 自動並列化コンパイラが出力したソースプログラムに手動でデータ分散指示文（1 次元目 block 分散）を挿入し，スケーラビリティを測定した．結果を**図 8** に示す．

コンパイラによる自動データ分散を行わない場合（no dist），およびコンパイラによって間接参照配列を対象から除いてデータ分散を行った場合（dist＋no indirect）に比べて，間接参照配列を対象に含めた場合〔dist＋indirect（DIR)〕は，飛躍的にスケーラビリティを改善し，16 プロセッサ時で no dist に比べて 4.1 倍，dist＋no indirect に比べて 2.9 倍に性能を向上させることができた．また，MPI プログラムとの比較においても，これに迫るスケーラビリティを実現することが分かった．

(2) ファーストタッチ制御方法の適用

次に，間接参照のように複雑なアクセスパターンになる可能性がある場合，最適なデータ分散形状自体が複雑になる可能性があると考え，ファーストタッチ制御方法を適用して評価を行った．具体的には，ソースプログラムに対し，以下の人手変換を施した．

- ターゲット配列のクローン配列を生成し，パラメータ確定位置より前のターゲット配列をクローン配列にリネイムする．
- 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．
- ファーストタッチ制御ループの直後で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．

CG のソースプログラム中の一部を切り出し，簡素化した変換例を**図 9** に示す．本例は，配列 a がターゲット配列である場合の変換例である．CG では，ターゲット配列 a のカーネルループでループ繰返し範囲に配列 rowstr が用いられる．したがって，配列 rowstr の各要素の値が決定するまでファーストタッチ制御ループを実行することができない．配列 rowstr の値は，手続き sparse の 15 行目までに確定する．よって，まず 3 行目で配列 a と同じ宣言形状のクローン配列 a_ftc を宣言する．次に，15 行目以前の配列 a の参照をすべてクローン配列に置換する（10 行目）．次に，15 行目の直後にカーネルループの参照パターンを再現するファーストタッチ制御ループを挿入する（17 行目から 24 行目まで）．続いて，そ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ループを挿入する（25 行目から 30 行目）．このプログラムを OS の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並列実行し，性能を測定した．結果を図 8 の dist＋indirect（FTC）に示す．

グラフに示すように，データローカリティの改善により，大幅にスケーラビリティが向上し，MPI プログラムを上回る性能が得られた．CG，class B，16 プロセッサ時で，コンパイラによる自動データ分散を行わない場合に比べて 6.0 倍，指示文版に比べて 1.5 倍，MPI プログラムに比べて 1.2 倍の性能向上を示した．

特に，MPI プログラムを上回るスケーラビリティを得たことは興味深い．MPI プログラムにおいて，ある程度のプログラム開発工数で実現可能なデータ分散形

```
1:      subroutine sparse( a, .....  )
           ...
2:    c FTC double precision a(1)
3:      double precision a(1),a_ftc(15825000)
           ...
4:      do j = 1, nrows
              ...
5:         do k = 1, nzrow
                 ...
6:            xi = x(i)
7:            if (xi .ne.  0.D0) then
8:               nza = nza + 1
9:    c FTC       a(nza) = xi
10:              a_ftc(nza) = xi
                    ...
11:           endif
12:        enddo
13:        jajp1 = rowstr(j+1)
14:        rowstr(j+1) = nza + rowstr(1)
15:     enddo
16:   c FTC code start
17:   !$omp parallel
18:   !$omp do schedule(static)
19:      do j=1,75000
20:         do k=rowstr(j),rowstr(j+1)-1
21:            a(k)=0
22:         enddo
23:      enddo
24:   !$omp end do nowait
25:   !$omp do schedule(static)
26:      do j=1,15825000
27:         a(j)= a_ftc(j)
28:      enddo
29:   !$omp end do nowait
30:   !$omp end parallel
31:   c FTC code end
           ...
32:      return
```

図 9 CG のファーストタッチ制御コード
Fig. 9 First touch control code for CG.

状は，比較的単純な block 分散，cyclic 分散，block-cyclic 分散ぐらいまでである．最適データ分散が上記のような場合は，MPI プログラムで良いスケーラビリティが得られてきた．また，データ分散指示文を用いた DSM 並列プログラムでも同様の傾向が見られた．本評価でも，FT や SP では MPI プログラムの性能の方が 16 プロセッサ時で 6%から 77%良かった．ところが，カーネルループ中にターゲット配列の間接参照が存在する場合など，最適データ分散が複雑な形状になる可能性がある場合は，ファーストタッチ制御方法を用いた DSM 向け並列プログラムの方が，はるかに少ないプログラム開発工数で正確なデータ分散を実現でき，今まで最もスケーラビリティが良いと思われてきた MPI プログラムを上回る性能が得られることを示した．これにより，ファーストタッチ制御方法の有効性の 1 つを示すことができたと考える．また，分散メモリ型並列計算機では得ることが困難であったスケーラビリティを DSM の特徴を生かして得たことは，DSM 自体の優位性の 1 つを示したことになると考えられる．

表 4 CG（class B）の FTC オーバヘッド
Table 4 FTC overhead for CG (class B).

プロセッサ数	1	2	4	8	16	32
OH (msec)	2704	2080	1116	718	536	436
OH /KER (%)	0.22	0.32	0.33	0.43	0.60	0.94

OH：ファーストタッチ制御コードの実行時間
KER：カーネル部分の実行時間

表 4 にファーストタッチ制御コードのオーバヘッドを示す．カーネル部分の実行時間に対して平均して 0.5%前後であり，比率として十分小さいと考えられる．今後は，適用条件に関する詳しい分析を行い，コンパイラへの実装を検討していく予定である．

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[3]，Gupta ら[4]，辰巳ら[15]，松浦ら[16] 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．辰巳ら[15] 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[16] は，手続きに跨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[4] 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跨る配列のデータ分散を自動決定することができる．

7. まとめ

本論文では，DSM 向け手続き間自動データ分散方法の実装と評価について述べた．この中で，OS によ

るファーストタッチ方式データ分散とそのコンパイラ制御の組合せにより，従来より正確なデータ分散が実現できることを示し，その有効性を検証した．また，上記の方法と従来利用されてきたデータ分散指示文を併用し，手続き間解析機能を搭載した自動データ分散方法を実装したことにより，従来のデータ分散方法では困難であった部分配列参照などでの最適なデータ分散，およびプログラム全体で最適なデータ分散が実現可能となった．

Origin2000を用いた評価の結果，本方法によるデータ分散指示文とファーストタッチ制御方法の併用で，ベンチマークプログラムNPB2.3serial/FT，SP，CG，SPECfp95/tomcatvの4題で，コンパイラによる自動データ分散を行わない場合に比べて16プロセッサ時に平均35.3%性能が向上した．ファーストタッチ制御コードのオーバヘッドは，FTで平均数%以下であり，データローカリティ改善の効果に比べて十分小さい．さらに，ファーストタッチ制御コードのオーバヘッドを現在の数十分の1から100分の1に削減できる最適化方法を示した．次に，FTを用いてデータ再分散の効果を評価し，Origin2000上では効果がないことを確認した．最後に，CGの間接参照を有する配列に対し，ファーストタッチ制御方法を手動で適用した場合，データローカリティの改善により，大幅にスケーラビリティが向上し，MPIプログラムを上回る性能が得られた．CGのclass B，16プロセッサ時で，コンパイラによる自動データ分散を行わない場合に比べて6.0倍，指示文版に比べて1.5倍，MPIプログラムに比べて1.2倍の性能向上である．

前報告[18]と本報告で，本方法の提案から実装，評価までをまとめた．今後は，自動データ分散方法全体としては，

- 間接参照配列対応の実装
- より多くのベンチマークによる評価を実施して有効性の検証を行い，性能向上，機能拡張へ反映させること

ファーストタッチ制御方法については，

- 有効なプログラムパターンの摘出
- プログラムパターンごとの適用条件の検討
- コンパイラへの実装範囲の拡大

を行う予定である．

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日立製作所システム開発研究所第3部305研究ユニット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) Kennedy, K. and Kremer, U.: Automatic Data Layout for High Performance Fortran, *Proc. Supercomputing'95* (1995).
4) Gupta, M. and Banerjee, P.: PARADIGM: A Compiler for Automatic Data Distribution on Multicomputers, *Proc. ICS'93*, pp.87–96 (1993).
5) Kandemir, M., Choudhary, A., Ramanujam, J. and Banerjee, P.: A Framework for Interprocedural Locality Optimization Using Both Loop and Data Layout Transformations, *Proc. ICPP'99*, pp.95–102 (1999).
6) 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).
7) 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).
8) Laudon, J. and Lenoski, D.: The SGI Origin: A ccNUMA Highly Scalable Server, *Proc. ISCA'97*, pp.241–251 (1997).
9) 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).
10) High Performance Fortran Forum: *High Performance Fortran Language Specification Version 2.0* (1997).
11) Sato, M., Hirooka, T., Wada, K. and Yamamoto, F.: Program Partitioning Optimizations in an HPF Prototype Compiler, *Proc. COMPSAC'96*, pp.124–131 (1996).
12) SGI MIPSpro Fortran77 Programmer's Guide, Silicon Graphics Inc
13) 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).
14) 原田 浩，石川 裕，堀 敦史，手塚宏史，住元真司，高橋俊行：ソフトウェア分散共有メモリ

SCASH におけるページ管理ノードの動的再配置機構の実装と評価，情報処理学会研究報告，99-HPC-77, pp.89–94 (1999).
15) 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).
16) 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).
17) 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).
18) 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).
19) The NAS Parallel Benchmarks. http://www.nas.nasa.gov/Software/NPB/
20) SPEC Benchmarks. http://www.specbench.org/

(平成 12 年 8 月 22 日受付)
(平成 13 年 3 月 9 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．同社システム開発研究所を経て，現在，同社情報コンピュータグループ事業企画本部主任技師．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．