計算機アーキテクチャ 142−31
ハイパフォーマンス
コンピューティング 85−31
（2001. 3. 9）

ソフトウエア分散共有メモリ上のOpenMP Omni/SCASHにおけるNPBの最適化と性能評価

長谷川 篤史† 佐藤三久††
石川 裕†† 原田 浩††

我々は、ソフトウェア分散共有メモリSCASHを用いて、PCクラスタ用のOpenMPコンパイラOmni/SCASHを開発した。分散共有メモリ上で効率良くメモリアクセスを行なうため、ホームノードを指定するmapping指示文と、データの配置に合わせてスケジューリングを行うaffinityスケジューリングをOpenMPの指示文に拡張した。逐次版NPBをOpenMPで並列化したものを基に、Omni/SCASHで拡張したDSM環境向けの指示文を用いて最適化し、32ノードのPCクラスタで実行した結果、クラスAのSP,BTで7.6〜16.3倍の性能向上が得られた。

Optimization and Performance Evaluation of NPB on Omni OpenMP Compiler for SCASH, Software Distributed Memory System.

Atsushi Hasegawa,† Mitsuhisa Sato,†† Yutaka Ishikawa†† and Hiroshi Harada††

We have developed a Omni/SCASH, an OpenMP compiler for the SCASH software distributed shared memory system on a PC cluster system. Extended directives, mapping directive and affinity scheduling, are initroduced in OpenMP directives for a memory access optimization on the DSM environment. A mapping directive specifies the home nodes of each shared data. Affinity scheduling is a loop scheduling policy which schedules iterations onto threads associated with the data mapping. The SP and BT benchmark programs in the NAS Parallel benchmark are parallelized and optimized using Omni OpenMP directives. The evaluation results show that the execution of parallel SP and BT are 7.6 - 16.4 times faster than that of those serial versions.

1. はじめに

近年のマイクロプロセッサの高性能化とクラスタ技術の進歩により、従来、高性能な並列処理環境に、安価なPCを高速なネットワークで結合したクラスタシステムが利用されるようになって来ている。一般的に、これらのクラスタシステムは分散メモリ環境で構築されており、MPIなどのメッセージ通信ライブラリを用いてプログラムを記述する必要がある。並列化するためには、プログラムを大幅に書き換える必要が生じ、かなりの労力を必要とする困難な作業となっている。

これに対して、共有メモリ型並列計算機向け並列化インターフェイスとしてOpenMPが提案されている[1]。OpenMPでは、データ並列を利用して、並列化可能なループや同期位置、排他処理などをコンパイラに対して指示する事により、プログラムの並列化を行う。OpenMPは、プログラムの並列化が容易であることから、並列プログラムの標準インターフェイスとして普及しつつある。しかしながら、分散メモリ環境における並列プログラムの開発は、事実上、MPIのみに限定されている。

RWCPでは、OpenMPを並列化インターフェースとした並列計算機環境を構築する事を目的として、OpenMPコンパイラOmni[2]を開発している。Omniは、ソフトウェア分散共有メモリ(Software Distributed Shared Memory, SDSM) SCASH[4][5]を用いて、共有メモリ向けにOpenMPで記述されたプログラムを、クラスタ上で実行させることができる。SCASHは、並列オペレーティングシステムSCore[3]上に提供されている、ページ管理機構を利用したSDSMである。我々は、このSCASHを用いたOpenMPコンパイラ

† 株式会社NEC情報システムズ
NEC Informatec Systems, Ltd.
†† 技術研究組合 新情報処理開発機構
Real World Computing Partnership (RWCP)

Omni/SCASH の性能評価として、逐次版 NAS Parallel ベンチマーク 2.3(NPB) の BT と SP を OpenMP で並列化し、32 ノードの PC クラスタ上で性能を測定した。この結果、クラス A の SP,BT で、それぞれ、7.6 倍～16.3 倍、性能が向上することが判った。

また、佐藤ら[6] により、分散メモリ型並列計算機では、ホームノードの割り当てが、計算性能に大きな影響を与えることが判っている。Omni/SCASH では、ホームノードの割り当ての指定やホームノードの割り当てに対応したループスケジューリングを行うための指示文を拡張している。これらの指示文を使用して BT,SP を最適化し、性能を測定した結果、2～8 台程度の小規模な環境において、より性能が向上する事が判った。本稿では、Omni/SCASH を用いて行った、OpenMP による NPB の並列化と、Omni/SCASH に拡張した指示文を使用した最適化の手法に関して述べた後に、それらの測定結果と評価に関して述べる。

以下、2 章で、Omni/SCASH のメモリ割り当てとループスケジューリングに関する実装方法を述べる。次の 3 章で、SP,BT の並列化と拡張した機能を用いた最適化方法に関して述べる。4 章で、評価結果に関して述べた後に、5 章で、結果に対する考察を述べる。最後の 6 章で、まとめと今後の課題に関して述べる。

2. Omni/SCASH コンパイラ

2.1 メモリ割り当て

SCASH が提供するメモリモデルでは、プログラムやデータの各セグメントは、ノード毎に独立した分散メモリ上に割り当てられる。共有メモリは専用のアロケータを通してのみ提供され、実行時に確保する必要がある。

OpenMP では、静的に宣言された大域変数は、特別な指定がない限り共有される。このため、Omni/SCASH では、以下のようにコードを変換し、全ての共有変数を実行時に割り当てる。

- 全ての静的な大域変数の宣言を、実行時に割り当てられる共有メモリ領域へのポインタ変数の宣言に変換する。
- 大域変数への参照を、このポインタ変数を用いた間接参照に変換する。
- コンパイル単位で宣言された変数を実行時に割り当てるための初期化関数を生成する。

初期化関数は ctor セクションに置かれ、リンカにより、プログラム起動時に実行されるようにリンクされる。初期化関数は、共有変数に関するテーブルを作成し、初期化するだけである。共有メモリの確保は、主プログラムの実行前に、ノード 0 のスレッドが、このテーブルを使用して行い、確保したアドレスを他のノードに対して broadcast する。各ノードでは、それを受け取り、個々のノード内の共有変数へのポインタ変数を初期化する。

また、Omni/SCASH では、各スレッドはノードに固定で割り付けられている。このため、スレッドに固有な変数である threadprivate 変数は、単に、静的な大域変数にコード書き換えることで実装している。

2.2 データマッピングのための拡張

SDSM システムでは、各ページのホームノードの割り当てがプログラムの性能に大きく影響する。SCASH では、他のノードがホームノードになっているページをアクセスすると、ページフォールトが起きる。ページフォールトを起こしたページは、リモートメモリリードによって、ホームノードからページのデータを自ノードに転送する。また、バリア同期実行時に、ホームノードに対してページを更新するため、更新したメモリの内容を転送する。したがって、SCASH 上で性能を得るためには、できるだけ、データをアクセスするノードとデータのホームノードを一致させ、データの転送量を低減させる必要がある。OpenMP の指示文には、データを特定のメモリ領域にマッピングする機能はない。このため Omni/SCASH では、データ配置指示文として mapping 指示文を拡張している。以下に mapping 指示文の使用例を示す。

```
      dimension a(100,200,300)
!$omn mapping(a(*,block,*))
```

この例では、3 次元の配列変数 a の 2 次元目でブロック分割し、ホームノードの割り当てを行うことを指示している。但し、SCASH でのメモリの一貫性制御は、ページ単位でしか行われないため、実際には、ページ単位の粒度でしかマッピングが行われない。

また、Omni/SCASH では、mapping が指定されない場合、配列全体を block 分割してホームノードの配置を行う。

2.3 ループスケジューリングのための拡張

OpenMP の指示文には、ループをデータの配置に合わせてスケジューリングするため指示節が存在しない。このため、Omni/SCASH では、mapping 指示文で指定したデータ配置に合わせてループの割り当てを行う affinity スケジューリングを拡張している。affinity スケジューリングの使用例を以下に示す。

```
      dimension a(100,200,300)
!$omn mapping(a(*,block,*))
!$omp do schedule(affinity, a(*,j,*))
      do j=1, 200
        a(i,j,k) = ...
      end do
!$omp end do
```

この例では、do ループの繰り返し処理は、a(*,j,*) のホームノードと同じノードに割り当て、並列で実行することを指示している。

3. NPB の並列化

3.1 OpenMP による並列化

Omni/SCASH は、SDSM を使用しているため、

SMP と比較して、データの同期に非常に時間がかかる。このため、OpenMP による並列化を行なうにあたり、極力、バリア同期、および、同期を必要とするデータを減らすように並列化を行った。

バリア同期の回数を減らすため、SP,BT の性能計測区間を 1 つの parallel リージョンで宣言し、その中に含まれる do ループを、nowait 指示節付きの do 指示文で並列化した。これにより、各スレッドは、do ループの並列実行後、バリア同期を実行する事なく、次の処理を実行する。また、データの同期が必要となる箇所に関しては、明示的にバリア同期の指示文を挿入した。この並列化により、SP,BT の性能計測区間で使用した OpenMP の指示文と、挿入した個数は、表 1 に示す通りである。

表 1　使用した OpenMP の指示文と使用回数

	:	SP	BT
parallel	:	1	1
do(nowait)	:	65	39
barrier	:	6	6
master	:	1	1

SP,BT の各処理は 3 重の do ループから構成されている。Omni/SCASH はネスト並列をサポートしていないため、配列の最も右側の次元に対応したループを並列化した。これは、Omni/SCASH が default のループスケジューリングとして使用している static スケジューリングが、ループを “ループ回数/ノード数” でブロック分割して各ノードに割り当てるため、配列の最も右側の次元に対応したループで分割することにより、ループ内でのデータのアクセスパターンが、ホームノード割り当てと一致し、自ノード上にあるデータが使用されるためである。但し、データに依存関係が存在する箇所に関しては、その次の次元に対応したループで並列化している。

Omni/SCASH では、SCASH の制限により、使用できる共有メモリのサイズに制限がある。今回、実験に使用した計算機では、クラス A の BT が使用する全ての変数を共有メモリ上に置くことができない。また、不必要な変数を共有メモリ上に配置すると、同期オーバーヘッドが増大するため、データの同期が必要な変数のみ共有メモリ上に置き、初期化後、書き換えられない変数等、同期を必要としない変数は threadprivate 変数として定義し、ユーザープログラムのデータセグメント上に割り当てた。こらの変数に関しては、参照される前に、copyin 指示節を用いて各スレッドの分散メモリ上に値をコピーするように指示文を追加した。これにより、SP で使用する共有メモリ領域は 80MB のデータ領域のうち 38MB のみとし、BT で使用する共有メモリ領域は 304MB のデータ領域のうち 34MB のみとした。

3.2　データ配置指示文による最適化

OpenMP による並列化では、配列変数のホームノードの割り当てに合わせて、do ループを並列化している。しかしながら、SP では、配列の最も右側の次元に対応した do ループが存在しない変数が存在する。このため、OpenMP の指示文のみでは、ホームノードの配置とデータのアクセスパターンを適切に割り当てることができない。これを解決する手段として、Omni/SCASH では、データ配置指示文として mapping 指示文を拡張している。mapping 指示文では、配列の任意の次元で、データを要素単位でブロック分割し、ホームノードを割り当てる事ができる。本実験では、共有メモリ上に配置された配列変数に対して mapping 指示文を指定し、do ループに対応する最も右側の次元で block 分割するように指定した。これは、Omni が生成する配列変数の構造が、左側の次元から順にメモリ上で連続するようになっているため、より右側の次元で分割する事で、block サイズが大きくなり、ページ単位に丸めた時の誤差が小さくなるからである。

3.3　affinity スケジューリングによる最適化

SP,BT では、do ループの対象が配列全体でないため、配列の一部がループの範囲外に存在する。static スケジューリングでは、ループの範囲をブロック分割して各ノードに割り当てている。このため、mapping 指示文で、do ループに対応したホームノードの配置を指定しても、ホームノードの配置とループの割り当て位置が異なり、do ループの割り当てが最適にならない。これを解決する手段として、Omni/SCASH では、schedule 指示節に対して、affinity スケジューリングを拡張している。affinity スケジューリングは、do ループで使用するループカウンタと、mapping を指定した配列変数をパラメータとして取り、配列に指定された mapping の割り当てに合わせて、do ループを分割し、各ノードに割り当てる。本実験では、do 指示文に対して、affinity スケジューリングを指定し、配列変数のホームノードの割り当てに合ったループスケジューリングを行なうようにした。また、affinity スケジューリングでは、ループに指定された繰り返し回数に関係なく、ループカウンタの値によってのみ、各ループを実行するノードが決定する。これを利用することで、図 1 のように、ループカウンタによりアクセスされるデータが同じである場合、ループ範囲が異るループでも、データの同期無しに連続して実行する事が可能となる。

これを利用することで、affinity スケジューリングを指定した SP,BT では、共に、バリア同期の回数を 6 回から 3 回に減らし、同期によるオーバーヘッドを削減した。

3.4　update 機能によるオーバーヘッドの削減

SP,BT の各処理は、3 重の do ループにより実装されている。ループは配列変数の各次元に対応しており、

```
!$omp do schedule(affinity, array(i))
        do i=0, 100
          array(i) = array(i) + 1
        end do
!$omp end do nowait
!$omp do schedule(affinity, array(i))
        do i=1, 99
          array(i) = array(i) + 2
        end do
!$omp end do
```

図 1 データの同期を省略できるケース

ループ内の処理には、いずれかの次元に対して依存関係が存在する。このため、mapping 指示文で block 分割を指定した次元に対応するループで、並列化できない処理が存在する。このような処理では、ホームノードの配置と異るアクセスパターンで、データをアクセスするように並列化しなければならない。SCASH では、システムのページ管理機構を利用して共有メモリへのアクセスを制御しているため、自ノードにデータが存在しない場合、実際にページフォールトが発生してから、ホームノードに対してリモートメモリリードを行い、自ノードにデータを転送する。この間、ページフォールトを起こしたノードは、データの転送が終了するまで処理が中断されてしまう。このため、データ転送に要するオーバーヘッドにより、性能が著しく低下する。

このオーバーヘッドを軽減する手段として、我々は、データを必要とするノードに対して、ホームノードからデータを事前に送信する update 機能を SCASH に実装し、リモートメモリリードで使用されるメッセージを削除し、オーバーヘッドの一部を削減した。現時点で、Omni/SCASH から update 機能を使用するための指示文は、未実装であるため、Omni/SCASH が出力するコードに対して、手動で update 機能の呼び出しを実装した。また、update 機能によるデータ転送は、Omni/SCASH で指示文として実装する事を考慮し、配列の任意の次元を、do ループのスケジューリングに合わせて、データをブロック分割して配置するように実装した。

4. 性能評価

4.1 実行環境

今回、実験に使用した環境は、ギガビットネットワークである Myrinet を用いて、32 台の PC を接続した PC クラスタである。各ノード (表 2 参照) には、NEC の Express5800/Rc-2 を使用し、並列実行環境として、RWCP で開発した SCore-3 の開発バージョンを使用した。各ノードは 2 プロセッサの SMP であるが、本実験では、各ノード、1 プロセッサのみを使用した。

表 2 各ノードの構成

CPU	:	Dual Pentium III 800MHz
Chip Set	:	ServerWorks III LE
Memory	:	Registered SD-RAM 512MB
NIC	:	Myrinet M2M-PCI64A-2,
		Intel Ether Express 100 Pro × 2
OS	:	RedHat 6.2 + Linux kernel-2.2.17

4.2 実行方法

今回、Omni/SCASH の性能評価を行うにあたり、拡張した指示文による効果を比較するため、以下に示す 5 種類のクラス A の SP,BT を用意した。

- 並列化していない (serial)
- OpenMP により並列化 (none)
- 上記 none に、mapping 指示文を指定 (mapping)
- 上記 mapping に、affinifty スケジューリングを指定 (affinity)
- 上記 affinity に、update 機能を使用 (update)

なお、Omni/SCASH の backend コンパイラ、および、シリアル版のコンパイラには egcs-1.1.2 を使用し、オプティマイズオプションには、"-O3 -malign-double -funroll-loops" を指定した。

実験では、serial 版のみ 1 ノード上で実行し、それ以外は 1,2,4,8,16,32 台とノード数を変化させて計測を行った。実行結果は、それぞれ 5 回づつ実行した結果の平均値を採用した。但し、今回使用した SCASH は、1 ノードでの実行に関して最適化していないため、1 ノード上で実行した場合でも、バリア同期、TLB ミスハンドラ等の処理を、並列実行時と同様に実行する。

4.3 実行結果

SP の計測結果を図 2 に示す。また、同時に、SCASH のページ転送回数を測定した。この結果を図 3 に示す。

実行結果より、Omni/SCASH で並列化したプログラムを 1 ノードで実行した場合、並列化しなかった場合と比較して、1.1〜1.2 倍程度性能が低下した。また、32 台で実行することにより、OpenMP の指示文のみで並列化した場合で約 5.3 倍、最も性能が良かった affinity スケジューリングを指定した場合で約 7.6 倍の性能向上が得られた。

BT の計測結果を図 4 に、ページ転送回数の測定結果を図 5 に示す。

実行結果より、Omni/SCASH で並列化したプログラムを 1 ノードで実行した場合、並列化しなかった場合と比較して、性能低下は 1.1 倍以下であった。また、BT では、SP と異なり、ほぼ同じ結果が得られている。これは、計算量に比べて、ページの転送回数が少ないことから、差が出にくくなっているためである。16 台以下では、Omni/SCASH で拡張した affnity スケジューリングや update 機能を使用した方が良い結果が得られている。しかしながら、32 台で実行した場合、OpenMP の指示文のみで並列化したものが 16.7

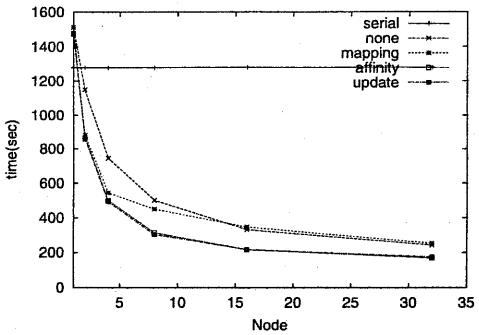


図 2　実行結果 (SP)

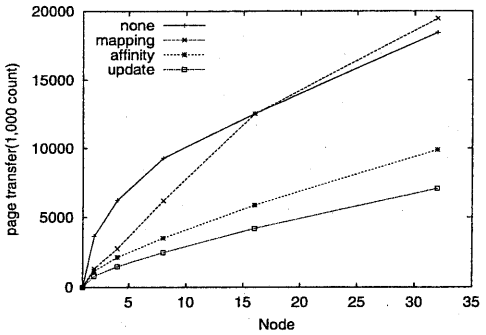


図 3　ページ転送回数 (SP)

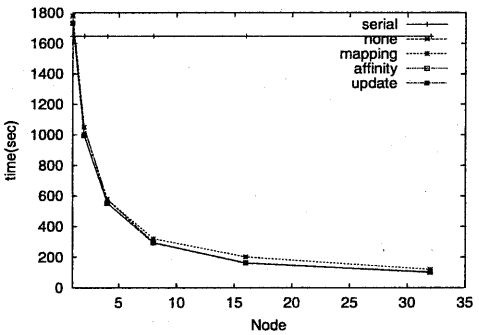


図 4　実行結果 (BT)

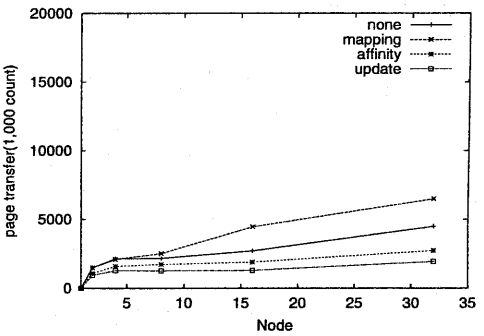


図 5　ページ転送回数 (BT)

倍と最も性能が良く、affinity スケジューリングを指定した場合、update 機能を指定した場合の性能向上は、それぞれ、16.0 倍～16.3 倍であった。

5. 考　　察

上記の実験結果に関する考察、および、Omni/SCASH に拡張した機能を用いて、BT,SP を Omni/SCASH で最適化した場合の問題点に関して述べる。

5.1 mapping 指示文

SP では、default のホームノードの配置がループスケジューリングと異る配列変数が存在したため、mapping 指示文を指定することにより、性能が改善されている。また、図 3 のグラフに示されるように、データをアクセスするノードにホームノードが割り当てられたことで、リモートメモリアクセスが削減されたことが判る。しかしながら、8 ノード以上の環境では、SP,BT 共に、急激に性能が悪くなり、16,32 ノードでは、mapping を指定しない方が良くなっている。これは、mapping 指示文が、指定された次元の要素単位で、ブロック分割していることが原因である。クラス A の SP,BT では、配列の各次元は 0～64 であるのに対して、ほどんどの do ループは 1～62 である。このため、ノード数が多くなるに従い、データ配置とループの割り当てが合わなくなる。

5.2 affinity スケジューリング

affinity スケジューリングを指定して、mapping 指示文で指定したデータ配置にループのスケジューリングを合わせた場合、図 3,5 のグラフに示されるように、リモートメモリアクセスが削減されたことが判る。32 ノードで実行した場合、OpenMP 指示文のみで最適化した場合の結果と比較して、SP,BT で、それぞれ、1/2～1/3 のリモートメモリアクセスが削減されている。また、図 2,4 のグラフより、性能が向上したことが判る。しかしながら、BT の 16,32 ノードで実行した場合、affinity スケジューリングを指定することで、リモートメモリアクセスが削減されているにもかかわらず、性能が低下している。これは、do ループの割り当てに偏りがあるのが原因である。BT で affinity スケジューリングを使用した場合と static スケジューリングを使用した場合の do ループの割り当ては、図 6 のグラフのようになる。32 ノードで実行した場合、static スケジューリングでは、ほぼ全ノードに渡って均等に処理が割り当てられているが、affinity スケジューリングを指定した場合、1/3 のノードに対して処理が割り当てられていない。Omni/SCASH は、mapping が指定された変数を、要素単位でブロック分割して各ノードに配置する。この時、ブロックサイズに “要素数/

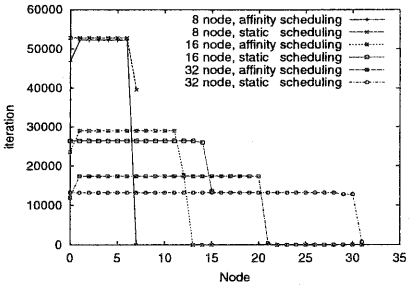


図 6 各ノードへのループの割り当て (BT)

ノード数" を切り上げた値を使用している。クラス A の BT,SP の配列の各次元は 0～64 であるため、ブロックサイズが 3 となり、後半のノードにはホームノードが割り当てられない。このため、affinity スケジューリングで、ループの割り当てに偏りが生じている。static スケジューリングでも、同様のブロック分割を使用して、ループの割り当てを行っているが、do ループは 1～62 の繰り返しであるため、ブロックサイズが 2 となり、ほぼ均等にループが割り当てられている。

5.3 update 機能の使用

update 機能を使用して、使用される変数の一部を事前にホームノードから転送した場合、affinity スケジューリングを用いて最適化した場合と比較して、SP の場合、2,4,8 ノードで実行した場合に性能が向上した。もっとも良い結果が得られたのは、8 ノードで実行した場合で、1.03 倍であった。BT の場合、8,16,32 ノードで実行した場合に性能が向上し、もっとも性能が良かったのは、32 ノードで実行した場合の 1.02 倍であった。また、該当箇所の処理で使用される全ての変数を update 機能で更新した場合、全てのケースで性能が低下した。

上記の結果から、update 機能を使用した場合、実行環境に合わせて適切に update の対象となるデータを指定すれば、性能が向上することが判った。update 機能を使用した場合の性能を低下させる要因としては、同時にデータの更新が発生する事による、ネットワーク上でのデータの競合等が考えられるが、現在のところ、原因は調査中である。

6. おわりに

本稿では、Omni/SCASH を用いて並列化した SP,BT の性能評価に関して述べた。Omni/SCASH を用いることにより、ユーザーは、OpenMP で書かれたプログラムを修正することなく、PC クラスタ上で実行することができるようになり、容易に PC クラスタ向けのプログラムを開発する事ができる。また、データ配置指示文を使用する事で、DSM 環境向けにプログラムを最適化する事ができる。

Omni/SCASH に拡張した機能を用いて最適化した結果より、affinity スケジューリングを使用して、SDSM のホームノードの配置とデータのアクセスパターンを合わせる事で、性能が向上することが判った。問題サイズが NPB のクラス A 程度の規模である場合、affinity スケジューリングを使用する事により、2～8 ノード程度のクラスタで性能が向上することが判った。より大規模な環境で性能向上を得るためには、ロードバランスを考慮した mapping、affinity スケジューリングの実装を行う必要があると考えられる。また、クラス A の SP,BT では、データサイズの問題より、単一の do ループの並列化では、64 ノード程度の並列化が限界であり、大規模な PC クラスタ環境に対応させるためには、ネスト並列による異るレベルでの並列化が必要になると思われる。

SCASH の update 機能に関しては、適切な指定がなされた場合、性能向上のための有効な機能であり、データサイズや実行台数による影響を検証し、自動的に最低な指定ができるようにする必要があると思われる。

謝　辞

本実験に関して貴重な御意見を頂いた、Omni コンパイラ開発グループの諸氏に感謝致します。

参考文献

1) OpenMP ARB Home Page, http://www.openmp.org
2) Omni OpenMP Compiler Home Page, http://pdplab.trc.rwcp.or.jp/pdperf/Omni
3) Parallel and Distributed System Software Laboratory, RWCP, http://pdswww.rwcp.or.jp
4) 原田, 手塚, 堀, 住元, 高橋, 石川. Myrinet を用いた分散共有メモリにおけるメモリバリアの実装と評価, JSPP'99, pp.237-244, 1999.
5) 原田, 手塚, 堀, 住元, 高橋, 石川：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, JSPP'99, pp.237-244, 1999.
6) 佐藤, 原田, 石川：ソフトウェア分散共有メモリシステム SCASH 上の OpenMP コンパイラ, 2000-HPC-82, pp.77-82, 2000
7) 草野, 佐藤, 細見, 妹尾：Cenju-4 の分散共有メモリ機構を用いた Omni OpenMP コンパイラ, 2000-HPC-83, pp.37-42, 2000
8) 草野, 佐藤, 佐藤：Omni OpenMP コンパイラと実行時ライブラリの性能評価, JSPP2000, pp.229-236, 2000