ハイパフォーマンス
コンピューティング 82−14
（2000. 8. 4）

ソフトウエア分散共有メモリシステム SCASH 上の OpenMP コンパイラ

佐藤三久† 原田 浩† 石川 裕†

PC クラスタ上のソフトウエア分散共有メモリシステム SCASH 向けの OpenMP コンパイラを開発した。SCASH はユーザレベルのライブラリとして実現されており、共有メモリ領域は実行時に割り当てなくてはならない。そのために、コンパイラは OpenMP で共有される大域変数を実行時に割り当て、その参照をこの領域への参照に書き換える。いくつかのベンチマークで性能評価した結果、16 ノード程度まではスケーラブルであったが、ページを最新のコピーを持つホームノードの割り当てが性能に大きく影響を及ぼすことがわかった。

OpenMP compiler for Software Distributed Shared Memory System SCASH

Mitsuhisa Sato ,† Hiroshi Harada † and Yutaka Ishikawa†

In this paper, we present an implementation of OpenMP compiler for a page-based software distributed shared memory system, SCASH on a cluster of PCs. In SCASH, The address space shared among the nodes must be allocated by its library function at run time. Our OpenMP compiler detects references to a shared data object, and replace it with the reference objects re-allocated in shared memory area. While some OpenMP programs scale up to 16 nodes, we found the home node allocation important to obtain good performance.

1. はじめに

本稿では、ソフトウエア分散共有メモリシステム SCASH 向けの OpenMP コンパイラを開発したので、その実装と性能について報告する。

マイクロプロセッサが高性能化するにつれて、ワークステーションや PC などをネットワークで結合するクラスタシステムが並列処理のプラットフォームの一般的な形態となってきている。このような分散メモリ型のマルチプロセッサでは、並列プログラミングに MPI や PVM などのメッセージ通信ライブラリを用いるのが一般的である。分散メモリ型の並列システムは、構築が容易かつ安価で、スケーラブルなシステムを構成できるのが利点であるが、このシステムを使うためにはメッセージ通信でプログラムを構成しなくてはならないため、プログラミングが複雑になり、プログラミングのコストが高いことが指摘されている。

最近、共有メモリマルチプロセッサ上で逐次プログラムを並列化するプログラミングインタフェースとして、OpenMP[1] が提案され、注目を集めている。OpenMP では、C や Fortran など従来のプログラミング言語に指示文 (C では pragma) を用いて並列記述を行う。OpenMP は、これまで各社独自仕様であった、コンパイラに対する並列化指示を共通化し、可搬性の高い並列プログラム開発を可能とすることを目的としている。並列実行ループ、同期と排他制御などの並列実行制御の指示文として指定し、並列プログラムと逐次プログラムを同一ソースで管理することができる。共有メモリマルチプロセッサにおいては、従来は POSIX Thread などのスレッドライブラリで並列プログラミングを行わなくてはならなかったが、これを簡便に行うことができ、並列化のコストを大幅に低減できる。我々は、既に様々な共有メモリマルチプロセッサシステムに移植性の高い OpenMP コンパイラ Omni OpenMP コンパイラシステム [2] を開発している。

本研究の目的は、OpenMP を共有メモリ型マルチプロセッサだけではなく、分散メモリ型マルチプロセッサのプログラミングに用いようというものである。OpenMP を分散メモリ型マルチプロセッサに用いるための一つの方法は、分散メモリシステム上でソフトウエアで仮想的な共有メモリを提供する、ソフトウエア分散共有メモリシステム (Software Distributed Shared Memory, SDSM) を OpenMP のターゲットのレイヤとして用いることである。SCASH[9] は、並列オペレーティングシステム SCoreE 上で提供されている、ページ管理機構を利用した SDSM システムである。

ほとんどの SDSM システムでは、プロセッサ間でアドレス空間の一部分のみ共有される。SCASH では、実行時に SCASH のライブラリ関数で確保されたアド

† 新情報処理開発機構
Real World Computing Partnership

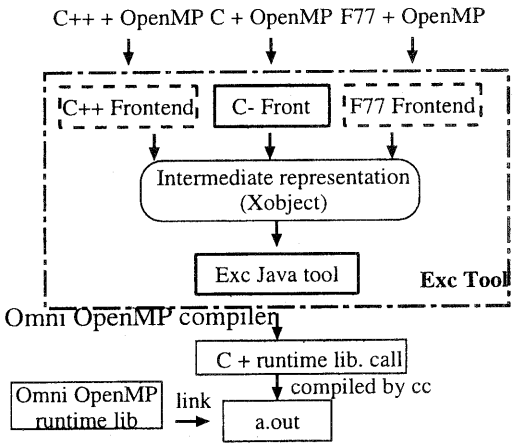

図 1 Omni OpenMP コンパイラシステム

レス空間を複数のプロセッサで共有することができる。プログラム内であらかじめ静的に宣言された変数は個々のプロセッサでのみアクセスされる。共有空間を動的にしか確保できない、このようなモデルを"shmem モデル" と呼ぶことにする。例えば、unix の shmem システムコール (mmap システムコールを用いて実装されることもある) を用いて記述した共有メモリプログラムもこのモデルの一例である。この shmem モデルでは、全ての共有変数は並列実行の前に、実行時に割り当てなくてはならない。我々は、Omni OpenMP コンパイラシステムを用いて、OpenMP プログラムを shmem モデルの並列プログラムに変換する OpenMP コンパイラを開発した。コンパイラは、共有される変数への参照を検出し、その変数に対して実行時に割り当てられる領域へのポインタによる間接参照に置き換える。これにより、ユーザは共有メモリ向けに記述した OpenMP プログラムを変更なしに、SCASH で実行できる。

次に、本研究の背景として、Omni OpenMP コンパイラとソフトウエア分散共有メモリシステム SCASH の概略について述べる。3 章において、SCASH 向けの OpenMP プログラムの変換と実行時システム、初期的な性能評価について述べる。4 章において、関連研究について触れ、5 章において、まとめ、課題について述べる。

2. 背　　景

2.1 Omni OpenMP コンパイラシステム

我々は、OpenMP を並列化インタフェースとした SMP マシン上の並列処理環境、Omni OpenMP コンパイラとその実行時ライブラリ[10] を開発した。図 1 にその構成を示す。

OpenMP コンパイラは、OpenMP プログラムを入力としてライブラリ呼び出しを含む並列 C プログラムに変換するトランスレータである。システムは、入力言語に対するフロントエンドと OpenMP に対する変換を行う Omni Exc ツールキットからなる。フロントエンド部が入力プログラムを Omni の中間表現である Xobject へと変換する。入力言語は C と FORTRAN77 に対応しており、それぞれ C-front と F-front がフロントエンドである。この中間表現 (Xobject) は、変数の型情報、グローバルな宣言情報、および実行文を保持する構文木の 3 つの情報を含んでいる。Exc ツールキットは、java で記述されたクラスライブラリであり、OpenMP の構文を含むプログラムの解析や変換、C への変換を行うことができる。構文木のノードが java のオブジェクトとして表現されているなど、高レベルな変換を簡単に実装できるように設計されている。OpenMP の変換についても、このツールキットの一部分として実装されており、プログラムの解析情報を用いて、OpenMP の指示に基づいて入力プログラムを実行時ライブラリ呼び出しを含む並列プログラムに変換し、C のプログラムとして出力する。

以上、述べた手順により変換されたプログラムは、Omni の実行時ライブラリとリンクすることで、並列実行を行う。この際に、様々な計算機に容易に移植することができることを考慮して設計、開発を行った。SMP 向けには SUN や Linux を始め、POSIX スレッドを持つプラットフォーム対応している。現在、`http://pdplab.trc.rwcp.or.jp/Omni/` より、ダウンロードできるようになっている。

2.2 ソフトウエア分散共有メモリシステム SCASH

SCASH は、Myrinet 上に低通信遅延かつ高通信帯域幅を提供する高速通信ライブラリ PM[14] を用いたソフトウェア分散共有メモリである。オペレーティングシステムのメモリ管理機能を利用し、ユーザレベルのライブラリとして実現されている。

共有メモリ領域の一貫性維持は、オペレーティングシステムが提供するページ単位で行われる。一貫性モデルとして ERC(Eager Release Consistency)[5),6)] を採用し、その実装にはマルチプルライタプロトコル[4)] を用いている。さらに、ページ単位の一貫性維持プロトコルとして、ページの無効化を通知する無効化プロトコルと、ページデータを送信し、ページの更新を通知する更新プロトコルの双方を実装し、実行時に選択できる。

以下に SCASH が提供している機能と API を示す。

- 共有メモリの初期化、割り当て、開放
  共有メモリを全ノード上で、割り当て、開放を行う。SCASH は共有メモリを全ノード上で等しいメモリ空間に割り当てる。
- 同期機構
  SCASH はメモリバリアとロックの 2 つの同期機構を提供している。メモリバリアは、全ノードでバリア同期を取り、共有メモリの全内容が、全ノード上

で同一の最新内容に更新されることを保証する同期機構である。ロックはブロッキングロックが提供される。ロックは分散ロックキューによって実現されている。

- 一貫性制御機構
  SCASHでは、書き込み共有メモリデータのホームノードへの書き戻し、ホームノードからの読み込みなどの、一貫性維持機構の一部をライブラリとしてユーザに開放している。
- その他
  SCASHは、グローバルメモリのデータをブロードキャスト、ページ単位でのホームノードの指定等の機能を提供している。特に、局所性が高い共有メモリ領域に関しては、ホームノードを指定を指定することによって通信量を削減し、実行性能を向上させることができる。

PMでは、従来のメッセージパッシングによる通信の他に、送信元のユーザ空間から、送信先のユーザ空間へ直接メモリ転送を行う、ゼロコピー通信(遠隔メモリ読み込み、遠隔メモリ書き込み)をサポートしている。遠隔メモリ読み込みによるページ転送では、ページ転送元ノードの計算を中断せずに、ページデータを得ること出来る。またPMのゼロコピー通信では、送信元から送信先へ直接メモリ転送を行うので、ページコピーのオーバーヘッドも発生しない。PMのゼロコピー通信機能を用いたページ転送によって、転送元ノードの計算中断、およびページコピーのオーバーヘッドをなくし、Myrinetの高帯域幅を活かした効率的なメモリバリアを実現している。

現在、SCASHは、新情報処理開発機構のPCクラスタ2号機上で稼働している。PCクラスタ2号機では、ページフォルトのハンドラのオーバーヘッドは125マイクロ秒、リモートのページの読み込み時間は112マイクロ秒になっている。

3. SCASH向けOpenMPプログラムの変換

3.1 shmemモデルへの共有メモリプログラムの変換

OpenMPプログラムでは、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。しかし、SCASHでは共有されるメモリ領域は実行時にしか割り当てることしかできない。

冒頭に述べたとおり、このような共有メモリ領域が動的に割り当てなくてはならない共有メモリモデルをshmemモデルと呼ぶことにする。shmemモデルに対しOpenMPプログラムを変換するために、以下のように全ての共有変数を実行時に割り当てるようにコードを変換する。

- 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(ファイル)毎に、その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。

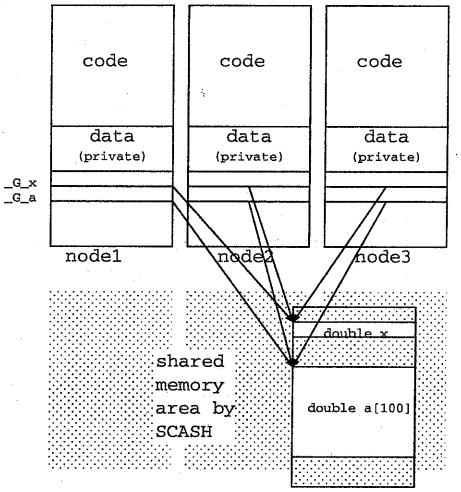

図2 共有変数への間接ポインタ参照

例えば、以下のコード:

```
double x;      /* global variable decl. */
double a[100];/* global array decl.*/
   ...
a[10] = x;
```

は、次のように変換される。

```
double *_G_x; /* indirect pointer to 'x' */
double *_G_a; /* indirect pointer to 'a' */
   ...
(_G_a)[10] = (*_G_x);
       /* reference through the pointers */

/* initialize function */
static __G_DATA_INIT(){
_shm_data_init(&(__G_x),8,0);
_shm_data_init(&(__G_a),80,0);
}
```

この概略を図2に示す。

コンパイラにコンパイル単位毎に生成された初期化関数は、ctorセクション(C++において、クラス初期化関数が置かれるセクション)に置かれ、リンカによって集約され、main関数が実行される前に実行されるようにしている。

SCASHにおいては、実行開始時に各プロセッサにお

いて、この初期化が実行されるが、実際には共有変数に関するテーブルを生成、初期化するだけである。ユーザの main プログラム実行前に、マスタになるプロセッサ 0 において、このテーブルを使って、共有メモリが実際に割り当てられた後、そのアドレスを全てのプロセッサに broadcast する。各プロセッサでは、それを受け取って、個々のプロセッサ内の共有変数へのポインタ変数を初期化する。

3.2 OpenMP 指示文の変換と実行時ライブラリ

OpenMP プログラムは、fork-join 型の並列プログラムである。SCASH においては、OpenMP スレッドは各プロセッサに 1 対 1 に対応している。ネストされた並列性はサポートしていない。

Omni OpenMP コンパイラでは 'parallel' 指示文によって指定された並列実行される並列リージョンを一つの関数にし、この関数を並列実行するコードを生成する。マスタプロセッサがその関数を引数として、実行時ライブラリ関数を呼び出し、マスタ以外のスレーブプロセッサで実行させる。スレーブプロセッサは、プログラム実行開始時に起動されており、この実行時ライブラリ関数による並列実行を待機している状態になっている。そして、スレッドで実行していた関数の実行が終了した時点で、再びマスタ以外のスレーブスレッドは待機状態となる。

並列実行において、マスタの auto 変数が共有する必要がある場合には、並列実行する前に、共有メモリ領域にコピーされ、この領域が各プロセッサにおいて共有される。並列実行が終了すると、この領域が元の auto 変数にコピーされる。

OpenMP の指示文は、コンパイラによって、その指示文に応じた実行時ライブラリを含むコードに変換される。生成されるコードは、基本的には共有メモリに対するコードとほとんど変わらないが、実行時ルーチンでは SCASH のライブラリ関数を使って、効率的にプロセッサ間の同期、通信を行うように実装されている。

3.3 性能評価

RWC PC cluster II (pcc2) において、初期的な評価を行った。CPU は Pentium Pro 200MHz(512KB on-chip cache), メモリ 256 MBbyte(EDO) のノードが、128 ノード、Myrinet によって結合されたシステムである。ノードのオペレーティングシステムは Linux-2.2.16, グローバルオペレーティングシステムとして、SCore-3.1 が稼働している。

ベンチマークとして、4 点ステンシルの jacobi 法による laplace 方程式の解法 lap(サイズ 1024 × 1024, 繰り返し 50 回)、C で記述された NPB1 の CG 法 (クラス A) を用いた。バックエンドのコンパイラ gcc egcs-2.91.66 は、オプションは -O2 である。結果を図 1に示す。SCASH の一貫性維持のプロトコルとして、デフォールトの無効化プロトコルを用いた。SCASH では、特別なホームの指定がない場合には、ページを

# nodes	seq	2	4	8	16	32
lap/BLK	17.74	14.30	7.84	4.69	2.88	1.79
lap/RR	17.74	49.39	33.88	20.15	12.87	10.15
cg/BLK	83.79	48.90	29.49	20.86	18.08	19.65
cg/RR	83.79	55.20	33.88	23.33	18.83	19.73

表 1 PC Cluster II 上での Omni/SCASH の実行時間 (秒)

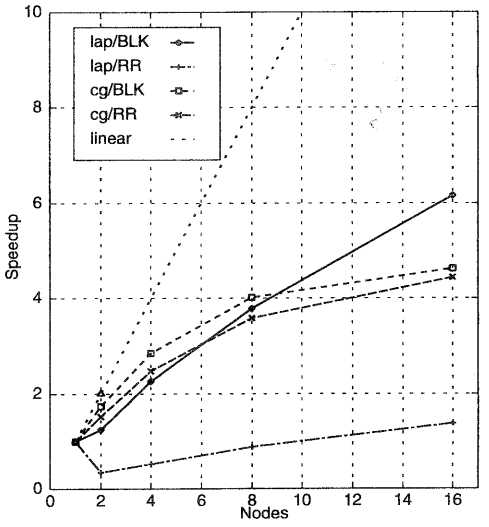

図 3 PC Cluster II での Omni/SCASH での対逐次性能向上率

round-robin にプロセッサに割り当てる。これを RR で示す。性能に対して、ホームの割り当ての影響を見るために、大きな配列に関して、プロセッサに等分割するブロック分割を実装した。これを BLK で示す。これを指定するために、ホームを割り当てを指定する pragma による簡単な指示文を用意した。

図 3に、16 ノードまでの対逐次向上率を示す。lap においては 16 ノード程度まで、CG では 8 ノード程度まで、スケーラブルである。

lap においては、ホームの割り当てが性能に大きな影響を及ぼすことが分かる。これは、配列の参照・更新に規則性があるにもかかわらず、RR においてはこれが考慮されず、更新のたびに大きなトラフィックが生じ、性能低下するためである。一方、cg においては、配列の参照が基本的にランダムであり、両者とも差はあまりみられない。

同じベンチマークを SMP クラスタ COMPaS II(ノードは、CPU Pentium II Xeon 450MHz, 4CPU/ ノード, メモリ 1GBbyte) で実行してみた。バックエンドのコンパイラは gcc2.95.2、オプションは -O4 である。これで、異なるノードで 1 プロセッサごとに 4 つの CPU を使う場合とハードウエアにより共有されている 4CPU の SMP との比較を行う。表 2に実行時間、図 4に対逐

# nodes	seq	2	4
lap/BLK	9.28	7.35	3.92
lap/RR	9.28	27.68	21.27
lap/SMP	9.28	6.55	6.60
CG/BLK	56.67	30.66	20.05
CG/BLK	56.67	34.22	22.67
CG/SMP	56.67	33.35	19.62

表2 COMPaS II での Omni/SCASH の実行時間 (秒)

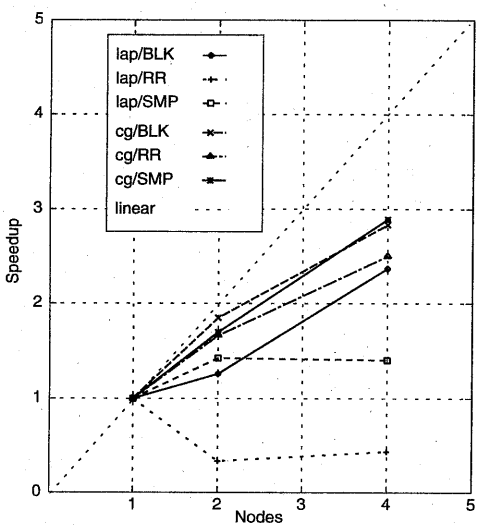

図4 COMPaS II での Omni/SCASH での対逐次性能向上率

次向上率を示す。ノードごとに1プロセッサのみを使い、SCASH において実行した結果は前と同じ凡例で示し、lap/SMP、cg/SMP は SMP 上の結果を示す。

PC の CPU を用いた SMP では、バスの性能が並列プログラムの性能を大きく制限する。例えば、計算に比べてメモリアクセス頻度が大きい laplace では、2CPU から 4CPU で全く性能向上しない。それにくらべて、SCASH では計算は独立なノードで行われるため、実質のメモリバンド幅が大きくなり、ノード間のページ更新のオーバーヘッドがあったとしても、SCASH のほうが性能が良くなっている (!)。

OpenMP でコンパイルされたコードは、共有変数への参照は全てポインタによる間接になっている。ちなみに、これらのベンチマークでは、この変換による性能低下はほとんどみられなかった。

3.4 考察・問題点

3.4.1 ページのホームノードの割り当ての影響

laplace の実験結果で見た通り、SCASH のようなページベースの SDSM ではホームノードの割り当ては大きく性能に影響を及ぼす。laplace のように配列への規則的なアクセスで、かつ、配列全体を書き換えるプログラムでは、ホームノードがランダムに割り当てられているとバリアにおいて、一斉にホームノードへの書き戻しがはじまり、性能が大幅に低下する。この場合には、プロセッサの参照パターンに応じたホームノードの割り当てをおかなくてはならない。今回の実験では、配列全体を等分割するブロック分割のみしかサポートしなかったが、OpenMP プログラムでは配列全体をアクセスするループが等分割されるため、プロセッサがホームとなっているページをアクセスすることになり、リモートのページの更新が少なくて済み、性能が向上した。

CG では、アクセスの参照パータンが比較的ランダムであり、ホームノードの割り当ての影響がすくない。また、書き換えるメモリの範囲が限られており、全体での更新のコストが少なく、ある程度の性能向上を見込むことができる。しかし、プロセッサが多くなるにしたがって、このコストが全体に及ぼす影響が大きくなり、性能向上しなくなっていると思われる。

3.4.2 OpenMP でのデータ配置最適化

OpenMP の API では、ループのスケジューリングはループ変数空間での分割を前提としたスケジューリングしか提供されていない。しかし、前節で述べたとおり、SCASH ではなるべく自ノードがホームになっているページをアクセスするように制御することが性能を得る重要なキーとなる。そのためには、配列などの大規模なデータに関してはノードへのマッピングの記述とマッピングに適合したループのスケジューリングが必要となるであろう。また、1つのループで複数の配列に効率的にアクセスするためには、HPF で見られるような複数配列のアライメントの機能も必要となる。

3.4.3 分散共有メモリシステムのスケーラビリティ

今回の実験では、32ノードまでのスケーラビリティが得られなかった。これは、大規模なマルチプロセッサになると、一つのノードあたりのデータ量が少なくなり、最適にノードをわりあてたとしても、SCASH のオーバーヘッドが顕在化してくるためである。ページが共有される割合も多くなり、更新のためのコストも高くなる。

これを解決する方法としては、SMP をノードにすることにより、ノード数を少なくすることが考えられる。

3.4.4 なぜ、shmem モデルか?

本研究では、共有されるメモリ領域が実行時に割り当てる共有メモリシステムをターゲットとしてコンパイラを開発した。これは、SCASH が共有メモリに関する機能をすべてユーザレベルのライブラリとして提供しているためである。始めからデータ領域を共有にするシステムも不可能ではないが、そのためには既存の libc などのライブラリもノード間で共有されること留意して書き換えなくてはならなくなり、現実的ではない。

ほとんどの OpenMP プログラムは変更なしにコンパイルできるが、このモデルのために共有メモリシステムと互換性のない部分がある。例えば、外部の既存ライブラリの持つデータを直接参照するもの (stderr など) は、threadprivate にしなくてはならないなどの制限が

ある。

4. 関連研究

これまで、OpenMPのソフトウエア分散共有メモリシステムへの実装が発表されている。H. Luら[11]は、TreadMarks[3]ソフトウエア分散共有メモリシステム向けのOpenMPコンパイラについて、発表している。その実装の詳細については述べられていないが、ソフトウエア分散共有メモリシステムを効率的に用いるために、OpenMPの仕様の拡張を提案し、評価している。このような拡張は性能を得るためには良いが、OpenMPの目的の一つである並列ソフトウエアの可搬性を著しく阻害する可能性がある。本OpenMPの実装においては、フルセットのOpenMPの仕様がサポートされており、共有メモリマルチプロセッサシステムでのOpenMPプログラムを動作させることができる。

また、Y.C. Huら[8]は、同様にTreadMarkも用いたSMPクラスタ向けのOpenMPのコンパイラについて述べている。現在、本コンパイラはSMPクラスタには対応していないが、これから対応する予定である。

また、我々は別プロジェクトとして、ページベースの分散共有メモリを用いずに、SMPクラスタ向けにコンパイラによって一貫性制御と通信のコードを最適化するコンパイラの開発も行っている[12][13]。

5. まとめ

本稿では、ソフトウエア分散共有メモリシステムSCASH向けのOpenMPコンパイラの設計、実装とRWC PC Cluster IIにおいての初期的な性能評価について報告した。我々のシステムを用いることにより、共有メモリマルチプロセッサにおいて実行されるOpenMPプログラムを変更なしにそのまま、分散メモリシステム上で実行できる。これにより、並列化、並列プログラミングのコストを大幅に軽減できる。しかし、今回の評価実験において、各データのホームの割り当てが大きく性能に影響を及ぼすことが明らかになった。これを解決するために、現在、OpenMPの指示文に加え、プロセッサに対するデータのマッピングの指定をするための指示文を検討している。また、データのマッピングを活用するためのループの分割スケジューリングの拡張、実装も行っている。

さらに、SCASHではデータのホームを参照頻度に応じて、動的に変更する機構も研究されており[7]、この機構はユーザがあらかじめホームのマッピングを陽に指定できない場合には、大いに性能改善に貢献するものと期待している。

謝辞　本研究に関して、Omni OpenMP compiler開発チーム、ならびにScore開発チームの皆様に感謝致します。

参考文献

1) http://www.openmp.org/
2) http://www.rwcp.or.jp/lab/pdperf/Omni
3) C. Amza, A. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, W. Zwaenepoel. "Treadmarks: Shared memory computing on networks of workstation", IEEE Computer, 29(2):18-28, Feb. 1996.
4) Amza C. Cox A. L. Dwarkadas, S. and Zwaenepoel W. Software DSM Protocols that Adapt between Single Writer and Multiple Writer. In *Proc. of the 3rd IEEE Symp. on High-Performance Computer Architecture (HPCA-3)*, pp. 261–271, February 1997.
5) J.B Carter, J.K. Bennett, and W.Zwaenepoel. Implementation and Performance of Munin. In *Proceedings of the Thirteenth Symposium on Operating Systems Principles*, pp. 152–164, October 1991.
6) K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A.Gupta,and J.Hennessy. Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors. In *Proceedings of the 17th ISCA*, pp. 15–26, May 1990.
7) H. Harada, Y. Ishikawa, A. Hori, H. Tezuka, S. Sumimoto, T. Takahashi, "Dynamic Home Node Reallocation on Software Distributed Shared Memory", In Proc. of HPC Asia 2000, pp. 158–163, Beijing, China, May, 2000.
8) Y.C. Hu, H. Lu, A.L. Cox, and W. Zwaenepoel "OpenMP on Networks of SMPs", Proceedings of the Thirteenth International Parallel Processing Symposium, pp. 302-310, April 1999.
9) 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", 並列処理シンポジウム JSPP'99, pp.237-244, 1999.
10) 草野和寛, 佐藤茂久, 佐藤三久, "Omni OpenMPコンパイラと実行時ライブラリの性能評価", 並列処理シンポジウム (JSPP2000),pp.221-228,2000.pp.229-236,2000.
11) H. Lu, Y. C. Hu, W. Zwaenepel. "OpenMP on Network of Workstations", In Proc. of Supercomputing'98, Nov. 1998.
12) M. Sato, S. Satoh, K. Kusano and Y. Tanaka, "Design of OpenMP Compiler for an SMP Cluster", Proc. of 1st European Workshop on OpenMP EWOMP'99, pp.32-39, 1999.
13) 佐藤茂久, 草野和寛, 佐藤三久, "OpenMPコンパイラにおけるメモリ一貫性制御の最適化", 並列処理シンポジウム (JSPP2000), pp.221-228,2000.
14) H. Tezuka, A. Hori, Y. Ishikawa, and M. Sato. PM: An Operating System Coordinated High Performance Communication Library. *LNCS, High-Performance Computing and Networking*, pages 708–717, 1997.