2004－HPC－98　(8)
2004／4／13

MPI上のソフトウェア分散共有メモリシステム

小島好紀[†]　佐藤三久[††]
朴泰祐[††]　高橋大介[††]

ソフトウェア分散共有メモリシステム SCASH の通信レイヤを MPI を用いて行う SCASH-MPI を設計実装した．SCASH では共有メモリに用いる領域をピンダウンするため，大規模な並列プログラムを実行させる場合に，大容量のメモリが必要となる問題があったが，この制限がなくなる．リモートメモリ機能を実現するために，通信のためのスレッドを生成し，通信を行わせる．MPI を用いることにより，様々なプラットフォームにおいて高速インタフェースを用いることができ，可搬性の高いシステムとなる．16 プロセッサ での初期性能評価の結果，比較的単純なアプリケーションでは，通信プロトコルに PM を用いる従来の SCASH(SCASH-PM) と等しい性能が得られた．また，NPB BT では SCASH-PM の約 86%の性能が得られた．

Software Distributed Shared Memory System on MPI

Yoshinori OJIMA,[†] Mitsuhisa SATO,[††] Taisuke BOKU[††]
and Daisuke TAKAHASHI[††]

We have designed and implemented a software distributed shared memory system (DSM), SCASH-MPI, using MPI as its communication layer of SCASH DSM. While SCASH requires a large amount of pin-down memory for shared memory area in a large-scale parallel program, SCASH-MPI removes this limitation. In SCASH-MPI, a thread is created to support remote memory communication. By using MPI as a communication layer, we can exploit high performance network of several clusters and high portability. The experiment on 16 processors shows the laplace benchmark with SCASH-MPI acheives the almost same performance as the orignal SCASH. And NPB BT with SCASH-MPI runs 0.86 times faster than that with the original SCASH.

1. はじめに

近年，マイクロプロセッサやネットワーク技術の進歩により，ワークステーションや PC をネットワーク結合したクラスタシステムが並列プラットフォームとして主流になりつつある．クラスタシステムは構築が容易であり，また安価であるという利点がある．

我々はこれまで，ソフトウエア分散共有メモリメモリシステム SCASH 向けに OpenMP プログラムの実行を可能にする Omni/SCASH の研究開発を行ってきた[1),2)]．

クラスタは分散メモリ型システムであるため，その上でのプログラミングは MPI や PVM などのメッセージ通信ライブラリを用いて行うのが一般的であるが，その場合メッセージ通信でプログラミングしなければならないため，プログラムが複雑になり易く，プログラミングのコストが高い．一方，共有メモリ型マルチプロセッサでは OpenMP で並列化することができる．その場合逐次プログラム自体には大きな変更を加えずに並列化でき，また段階的な並列化が可能であるため，並列化のコストを低く抑えられる．

そこで，分散メモリシステムの上で共有メモリプログラミングを可能にするために，ソフトウェア分散共有メモリ (Distributed Shared Memory:DSM) システムが研究・開発されている．SCore 上で提供されている SCASH はその 1 つであり，Omni OpenMP コンパイラシステムは OpenMP プログラムを SCASH を用いたコードに変換することにより，OpenMP プログラムを分散メモリ型システムであるクラスタで実行することができる．

マイクロプロセッサの高速化，低価格化とギガビットイーサネットなど高速なコモデティワークの普及により，大規模なプログラムでもクラスタシステムで実行できるようになりつつある．大規模なプログラムを

† 筑波大学大学院システム情報工学研究科
Graduate School, Doctoral Program Systems & Information Engineering, University of Tsukuba
†† 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba

共有メモリモデルで実行する場合の本質的な問題点は大きいアドレス空間が必要となることである．実際，現在，PC クラスタで主流になっている 32 ビットプロセッサではアドレス空間が不足し，大規模な共有メモリプログラムの実行が困難であることが多い．しかし，ようやくインテルの Itanium2 や AMD の Opteron などの 64 ビットのアドレスが利用可能なプロセッサがクラスタで利用できるようになりつつあり，この制限はなくなりつつある．

このような背景を踏まえ，これまでの経験から様々な問題点が明らかになってきた．SCASH では SCore クラスタソフトウエアの PM 通信レイヤを用いており，PM は効率的な通信を提供するだけでなく，ソフトウエア分散メモリシステムの実現に必要なリモートメモリ通信をサポートしている．しかし，効率的なリモートメモリ通信をサポートするために，共有メモリの実現に用いる領域を，ページアウトされない領域，すなわちピンダウンメモリにしなくてはならない．このため，大規模なプログラムを実行する場合には各ノードに大容量の実メモリが必要になってしまう．特に 64 ビットアドレス空間で大規模なプログラムを実行する時には全ノードに大量のメモリを装備するのはかなり現実的に困難な問題点となり得る．

また，PM を利用するためには SCore が必要となるが，SCore はチェックポイントやスケジューリングなどの様々な機能を提供するクラスタソフトウエアを提供するものの，SCore がサポートされていないクラスタ環境にも対応できることが望ましい．

本稿では，通信インターフェースをクラスタ環境で標準的な MPI を用いて実装するソフトウエア分散共有メモリシステムについて述べる．これにより，大規模なクラスタにおいて，ピンダウンメモリを使用するという制約を取り除き，かつ，可搬性に優れた共有メモリプログラミング環境を提供することができる．

2. 研究の背景と動機

本章では本研究の背景として，我々がベースとして用いたソフトウエア分散共有メモリシステム SCASH とそれに対する OpenMP プログラミング環境を提供する Omni/SCASH について述べる．

2.1 ソフトウエア分散共有メモリシステム SCASH

SCASH は，RWC[3] で開発されたクラスタシステム SCore 上で提供されているソフトウェア DSM システムである．Myrinet 及び Ethernet 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[4] を用いており，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）を採用し，その実装にはマルチプルライタプロトコルを用いている．さらにページ単位の一貫性維持プロトコルとして invalidate プロトコルと update 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を示す．

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，リモートメモリ通信をサポートしている．

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．

2.2 SCAH 向け OpenMP コンパイラ:Omni/SCASH

SCASH システム上で動作するプログラムは，OpenMP で並列化されたプログラムを Omni/SCASH コンパイラでコンパイルすることで得ることができる．Omni/SCASH は分散メモリシステム向けの Omni[5] OpenMP コンパイラであり，OpenMP で記述されたプログラムを分散メモリ型システムで実行可能なイメージにコンパイルする．

ソフトウェア DSM 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 上で良い性能を得るためには，できる限りデータをアクセスするノードとその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．しかし 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，独自の拡張としてマッピング指示文を持ち，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ることを可能にしている．また，データの割り当てに合わせてループの割り当てを行う affinity スケジューリングの機能を提供している．

2.3 SCASH の問題点

SCASH は，リモートメモリ通信を用いるため，通信レイヤとして PM 通信ライブラリを用いて実装されている．リモートメモリ通信の対象となる共有メモリ領域の割り当てにピンダウンメモリを用いるため，たとえ実際には使われないとしてもプログラム実行のためには多くの物理メモリが必要になり，実行できる問題のサイズが制限されてしまう．

並列システムでは並列効率を向上させるにはプログラムの規模を大きくすることが有効である．クラスタのノードとして，64 ビットのアドレス空間をサポートするプロセッサが利用できるようになりつつあるが，プログラムの規模を大きくすることができたとしても，実メモリが必要となるという制限は大きい障害になっている．

3. MPI を通信レイヤに用いたソフトウエア分散共有メモリシステム

3.1 設 計 方 針

これまで，述べた問題点を解決するため，SCASH をベースに，その通信レイヤを広く普及している通信レイヤである MPI を用いて実装する．SCASH のシステムは，メッセージ通信とリモートメモリ通信の 2 つの通信を使っている．メッセージ通信の部分については，通常の MPI の通信を用いることができる．リモートメモリ通信には，以下の方法を検討した．

(1) MPI2 のリモートメモリ通信機能を用いる方法.
(2) 別のスレッドを作り，TCP/IP を用いてリモートメモリ通信機能を MPI とは別に実装する．
(3) 同じくスレッドを作るが，メッセージ通信をこのスレッドに行わせることにより，リモートメモリ通信をサポートする．

まず，(1) であるが，リモートメモリを行う期間や範囲を制限する必要があり，ここでの利用は難しい．また，MPI2 の実装はいくつかあるものの，リモートメモリ通信機能についてはまだ実装されていない，あるいはその機能，性能が不明な点が多いことがある．(2) については，通信が TCP/IP のプロトコルに依存してしまうという問題点がある．クラスタにおいては ethernet 以外の様々な高速のネットワークが利用されているが，MPI ではこれらのネットワークが利用できるが，TCP/IP では利用できないことが多い．たとえ，TCP/IP がこのネットワークでサポートされていたとしても，TCP/IP のプロトコルのオーバーヘッドがあり，十分に性能が活用できないことがある．結論として，(3) の実装を用いることにした．

なお，DSM のプロトコルはこれまでの Omni/SCASH でと同様に，invalidate ベースのプロトコルに限定した．

3.2 MPI を用いた SCASH 通信レイヤの実装

通信のために，スレッドを生成し，このスレッドが MPI を用いて SCASH の通信レイヤを行う．現在，広く普及している MPI の実装は thread-safe でないため，通信を担当するスレッドを作成し，そのスレッドのみが MPI の関数を実行するようにした．

データを送信する際には，送信するデータを送信データのリストに加える．通信スレッドはそのリストを常に監視し，送信すべきデータがあればそれをリストから取り出し，`MPI_Isend` を用いて送信する．

また，`MPI_Irecv` でデータの受信をする．`MPI_Test` で受信の有無を監視し，受信したデータがあればそれを受信データのリストに加える．計算スレッドはそのリストから取り出す．

リモートメモリアクセスも同様に，リモートメモリライトは `MPI_Isend` でライト要求及び書き込むデータを送信し，リモートメモリリードはリード要求を送信し，`MPI_Irecv` で要求したデータを受信する．リモートメモリリードの要求に対しては，受け取ったスレッドは指定されたメモリを読みだし，要求元にかえす．リモートメモリライトは送られてきたデータを書きこむ．動作の概略を**図 1** に示す．

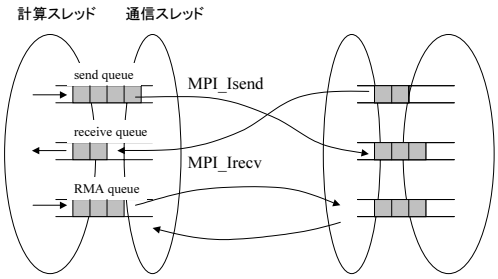

図 1 SCASH-MPI の動作の概略

3.3 スレッドを用いた通信の利点・欠点

通信スレッドが常に送受信リストを監視するため，CPU を占有してしまうという問題がある．しかし，Intel の最新のプロセッサでは HyperThreading 機能を用いることにより，このようなスレッドのオーバヘッドを低減させることができると考えている．Hyper-threading では，メモリのバンド幅の飽和などの問題からスレッドが十分に活用されていないことがあり，通信をこのスレッドを用いて行わせることによって有効活用できる可能性がある．また，クラスタは各ノードに dual PC 等の SMP PC を用いることが多く，メモリバンド幅等，同様な理由から十分に活用できないことが多い．この場合にも，スレッドを片方のスレッドで実行することにより，効率的に処理できることが期待される．

今回の実装では，MPI が thread-safe でないため，通信用のスレッドを生成した．この方法はほとんどの MPI の実装に適用できると考えているが，MPI の実装でもマルチスレッド環境を前提とした実装していないものもある．実際，一般に使われている MPICH ではマルチスレッド環境では不具合があった．今回の実装では，LAM を用いて実装した．

4. 基本性能の比較

SCASH システムにおけるページ転送通信のコスト，及びバリア同期のオーバヘッドを測定し，SCASH-MPI と SCASH-PM の基本性能を比較する．

4.1 評 価 環 境

測定に使用したクラスタの構成を**表 1** に示す．

4.2 SCASH のページ転送のコスト

長さが 8192 の 32 bit 整数型配列，すなわちサイズが 32 KB（= 8 ページ）の 1 次元配列をブロック分割して 8 ノードに分配し，マスタスレッドが他ノー

表 1 PentiumII クラスタの構成

CPU	Pentium II Xeon 450MHz
L2 キャッシュ	1MB
ノード	4-way SMP
Memory	2GB
ノード数	8
Network	100Base-TX Ethernet
OS	Linux 2.4.19
SCore	version 5.4
MPI	LAM 7.0.4
コンパイラ	egcs-2.91.66 (-O3)

ドのデータへ書き込みを行うのにかかる時間を測定した．書き込む範囲を変えながら，バリア同期を行い書き込みを終了するまでにかかる時間を測定した．これはマスタスレッドからの書き込みによる scatter 操作である．結果を**図 2** に示す．グラフの x 軸の値は，マスタスレッドがインデックス 0 番から x 番目までのデータに書き込むことを示す．

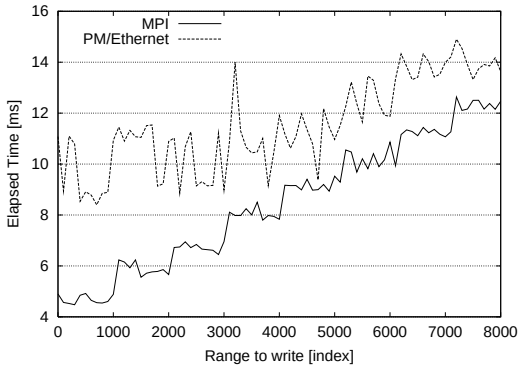

図 2 ページ転送のオーバヘッド

書き込みの範囲を広げると他のノードがホームであるページに書き込むことになり，バリア同期時に更新されたページデータの転送が行われる．そのため実行時間は階段状に増え，1 ページの転送にかかる時間は階段状になっているグラフの 1 段分の高さに相当すると考えられる．言い換えれば，グラフの全体的な傾きに相当する．

図 2 より，SCASH-MPI での 1 ページ転送のオーバヘッドは約 1.2ms である．SCASH-PM では明確な階段状のグラフは得られなかったが，3000[index] 以降では SCASH-MPI と同じような傾きになっているため，ページ転送のオーバヘッドも同様であると考えられる．

4.3 バリア同期のオーバヘッド

子スレッドがマスタスレッドのデータを読み，自分の持つ領域にコピーするのにかかる時間を測定した．8 × 8192 の 32 bit 整数型配列，すなわちサイズが 256 KB（= 64 ページ）の 2 次元配列をブロック分割して 8 ノードに分割し，子スレッドがマスタスレッドのデータを読む範囲を変えながら，自分の領域へのコピーを終えるのにかかる時間を測定した．これは子スレッドからの読み込みによる scatter 操作である．データコピーのみにかかる時間と，処理の最後のバリア同期までにかかる時間を測定した．この場合他ノードのデータの更新は行わないので一貫性維持通信は必要ない．したがって，2 つの時間の差はバリア同期のみのオーバヘッドである．結果を**図 3** に示す．

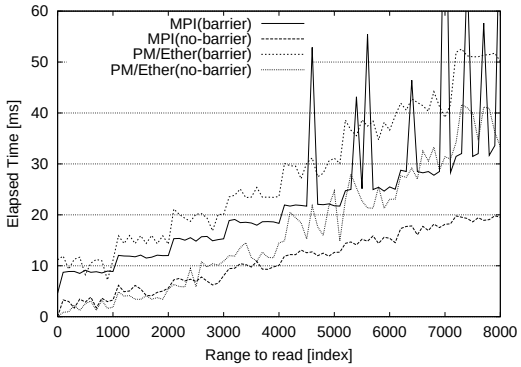

図 3 バリア同期のオーバヘッド

バリア同期のオーバヘッドは，barrier の値と no-barrier の値の差である．図 3 より，SCASH-MPI では約 8.5ms，SCASH-PM では約 9ms であり，SCASH-MPI の方が良い結果が得られた．これらの結果から，SCASH-MPI の基本性能は SCASH-PM と同等かそれ以上であると言える．

5. 性 能 評 価

5.1 対象プログラム

- **laplace**
  Jacobi 法による Laplace 方程式の解法．行列のサイズは 1024 × 1024(倍精度)，反復回数は 50 回とした．C で，SCASH のユーザライブラリを用いて記述した．
- **NPB BT**
  NPB 2.3 の BT を長谷川らが OpenMP を用いて並列化及び Omni/SCASH 向けに最適化したもの[6] を用いた．Class A を使用した．

5.2 測 定 結 果

使用するノード数を 1,2,4,8，ノード内で使用するプロセッサ数を 1,2 と変化させ，測定を行った．SCASH-MPI は SMP に対応していないため，ノード内で 2 プロセッサ使用する場合でもハードウェア SMP の機能は使用されない．

laplace の測定結果を**図 4** に示す．SCASH-MPI と SCASH-PM とでほぼ等しい性能が得られた．基本性能が同等であるため，アプリケーションにおいても等しい性能がえられたのだと考えられる．また，各ノードは 4-way SMP の PC であるため，ノード内で計算

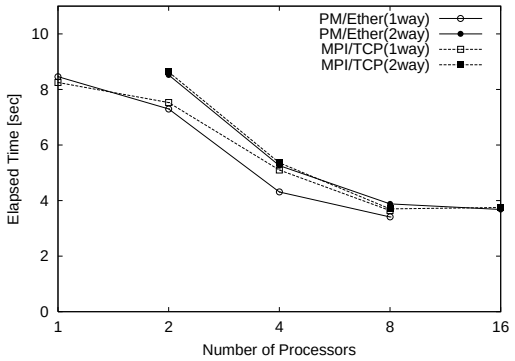

図 4 laplace の実行時間

に 2CPU 使用しても通信スレッドは残る 2CPU を使用することができるため，通信スレッドの負荷による性能低下はなかった．しかし計算に 4CPU 使用すると，計算と通信で 1 つの CPU を共有することになり，性能が低下すると考えられる．16 プロセッサで，最大で 1 プロセッサの約 2.3 倍の性能が得られた．

BT の測定結果を**図 5** に示す．ノード当たり 1CPU のみ使用して測定した．SCASH-MPI では最大で 1 プロセッサの約 3.2 倍，SCASH-PM では約 3.7 倍の性能が得られた．SCASH のシステムイベントのトレースを出力させ比較したところ，SCASH-MPI ではバリア同期やページフォルト時の処理にかかる時間が SCASH-PM よりも長いことが分かった．4 章での比較ではバリア同期のオーバヘッドは同等であったが，BT のような実アプリケーションではバリア同期内で転送されるメッセージが多く，その処理時間に差が生じたと考えられる．詳細な原因を明らかにするために今後さらに解析を進める必要がある．

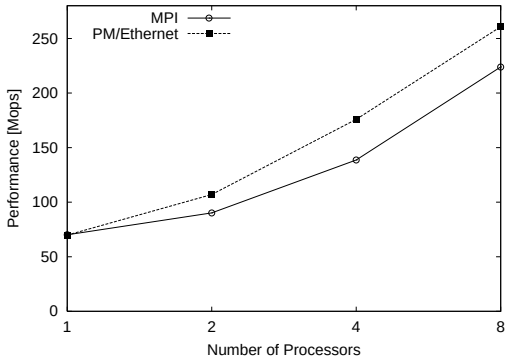

図 5 BT の性能

5.3 64bit システム上での評価

64bit CPU である Intel Itanium プロセッサを搭載したクラスタ上で評価を行った．使用したクラスタの構成を**表 2** に示す．対象プログラムには laplace を使用し，行列サイズは 4096 × 4096 とした．

測定結果を**図 6** に示す．PentiumII クラスタと同様

表 2 Itanium クラスタの構成

CPU	Itanium 800MHz
ノード	4-way SMP
Memory	32GB
ノード数	4
Network	1000Base-T Ethernet
OS	Linux 2.4.21
SCore	version 5.7
MPI	LAM 6.5.4
コンパイラ	gcc 3.2.2 (-O3)

に，SCASH-MPI は SCASH-PM とほぼ等しい性能が得られ，最大で 1 プロセッサの約 6.3 倍の性能と，高いスケーラビリティーが得られた．laplace は計算量のオーダは $O(n^2)$，通信量のオーダは $O(n)$ であるため，問題サイズが大きい方が通信に対する演算の割合が大きく，並列実行による性能向上が得られ易い．ネットワーク性能など異なるため単純な比較はできないが，PentiumII クラスタよりも高い性能向上率が得られたのはより大きい問題を使用したことが 1 つの要因であると考えられる．

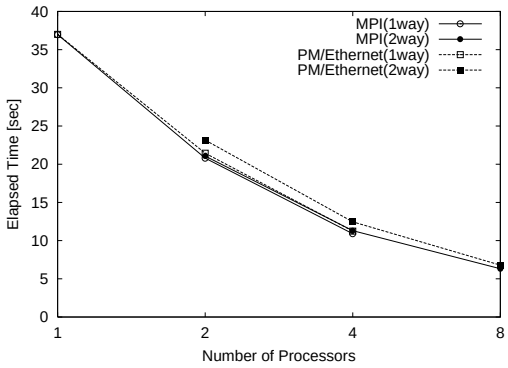

図 6 laplace の実行時間

6. 課　　題

6.1 Prefetch による効率化

今回の初期評価では，ethernet を用いたが，局所性の高い疎粒度のの問題，大きな問題についてはある程度の効果が得られたものの，通信速度が足りないために，十分な効率が得られないことがあることが分かった．DSM では，基本的にページフォルトが検出されてから通信が開始されるために通信が十分でない ethernet では効果を得るのが難しい．

そこで，ページを prefetch することを検討している．OpenMP では，並列性が構造的に記述するために，コンパイラにより，アクセスする領域を解析し，ループを開始する前に prefetch を行うことにより，通信と計算をオーバーラップさせる．このアクセス領域の解析については[7] に述べた方法を使うことができる．通信スレッドを利用した方法の処理の概要を以下に示す．

- prefetch の指示により，通信スレッドに prefetch 領域を伝える．
- 通信スレッドは，prefetch 領域のページを check し，当該ページが fetch されていない場合には，ページを lock し，prefetch の通信を起動する．
- 通信スレッドが prefech のページの通信を受け取り，ページの DSM の管理属性を Read Only に変え，ページを unlock する．
- 計算スレッドはページフォルトにより，ページがないことを検知する．もしも，ページが lock されていれば，通信スレッドが当該ページが prefetch 中なので待機する．lock されていなければ，ページを check し，もしもページがなければ，通常の DSM と同様に処理を行う．ページが既に prefetch されていれば，ページのハードウエアのページテーブルの属性を変えるのみでよい．

この方法は prefetch を行う通信と計算をページフォルトを通じて行うために，同期のオーバヘッドが少ない．また，通信のためのスレッドを有効に活用できるなどの利点がある．丹羽ら[8] や Dwarkadas ら[9] は，コンパイラにより解析を行い，一貫性維持のためのコードを挿入することでプログラムの DSM 上での実行を実現している．特に丹羽らはプリフェッチ機構の実装も行っている[8]．彼らは計算とプリフェッチを 1 つのスレッドで行っているが，我々はそれぞれのスレッドに分けて実装を行う．

6.2 クラスタ専用ネットワークの利用

ソフトウェア DSM システムでは，システムを構成するネットワークの性能がアプリケーションの性能に大きく影響する．今回の測定では Fast，及び Gigabit Ethernet を用いたが，性能をより向上させるためには高速なネットワークを用いることが必要である．

そこで，Myrinet や Infiniband などの高速なクラスタ専用ネットワークを用いて評価を行う必要がある．特に Myrinet 上の MPI を用いて測定を行い，PM/Myrinet を用いた場合との性能差について評価する必要がある．

6.3 Hyperthreading の利用

3 章に述べたが，別のスレッドを用いる方法は Hyperthreading などのマルチスレッドの機能を持つプロセッサに有効であると思われる．実際，プロセッサにマルチスレッドの機能があっても，メモリバンド幅は１プロセッサのままであり，計算にはマルチスレッドが有効に使えないのが現状である．この場合，通信や prefetch にこのスレッド用いることによって有効に活用できることが期待される．これについては，今後，実験を行い検証していく予定である．

7. おわりに

本稿では，ソフトウェア分散共有メモリシステム SCASH の通信に MPI を用いたシステムについて報告した．SCASH-MPI と SCASH-PM の基本性能を測定したところ，同等であり，単純な通信パターンを持つ laplace では SCASH-MPI は SCASH-PM と等しい性能が得られた．また，NPB BT では SCASH-PM の約 86%の性能が得られた．

今後の課題としては，6 章で述べた課題，SCASH-PM との性能差のさらなる解析が挙げられる．

謝辞 本研究に御協力頂いているヒューレットパッカードジャパンの原田浩氏，及び東京大学の石川裕助教授に感謝致します．本研究の一部は文部科学省科学研究費補助金 (基盤研究 (A)(1)) 課題番号 14208026) による．

参考文献

1) 佐藤三久, 原田浩, 長谷川篤史, 石川裕: Cluster-enabled OpenMP: 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, 情報処理学会論文誌：ハイパフォーマンスコンピューティングシステム, Vol. 42, No. SIG9(HPS3), pp. 158–169 (2001).
2) Ojima, Y., Sato, M., Harada, H. and Ishikawa, Y.: Performance of Cluster-enabled OpenMP for the SCASH Software Distributed Shared Memory System, *Proc. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, pp. 450–456 (2003).
3) PC クラスタコンソーシアム: http://www.pccluster.org.
4) Tezuka, H., Hori, A., Ishikawa, Y. and Sato, M.: PM: An Operating System Coordinated High Performance Communication Library, *Proc. 5th International Conference on High Performance Computing and Networking Europe (HPCN Europe 1997)*, Lecture Notes in Computer Science, Vol. 1225, Springer-Verlag, pp. 708–719 (1997).
5) Omni OpenMP Project: http://www.hpcc.jp/Omni.
6) 長谷川篤史, 佐藤三久, 石川裕, 原田浩: 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, 情報処理学会研究報告 2001-HPC-85, pp.181–186 (2001).
7) 建部修見, 佐藤三久, 関口智嗣: PC クラスタにおける TDL を用いた OpenMP コンパイラ, 情報処理学会研究報告 2001-HPC-87 (SWoPP 2001), pp. 123–128 (2001).
8) 丹羽純平: コンパイラが支援するソフトウェア DSM におけるプリフェッチ機構, 情報処理学会研究報告 2004-ARC-156, pp. 7–12 (2004).
9) Dwarkadas, S., Lu, H., Cox, A., Rajamony, R. and Zwaenepoel, W.: Combining Compile-Time and Run-Time Support for Efficient Software Distributed Shared Memory, *Proc. IEEE, Special Issue on Distributed Shared Memory, Vol. 87, No.3*, pp. 476–486 (1999).