# MPIを通信レイヤに用いる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 の通信レイヤに MPI を用いた SCASH-MPI を設計実装した．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．また，PM を用いた実装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，大容量の実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，通信を行う．4 ノードでの性能評価の結果，laplace ベンチマークでは通信プロトコルに PM を用いる従来の SCASH（SCASH-PM）と同等の性能が得られた．また NPB BT Class B では，従来の SCASH と比較して性能低下を約 6.3%に抑えることができた．

# Design of Software Distributed Shared Memory System Using MPI Communication Layer

Yoshinori Ojima,[†] Mitsuhisa Sato,[†] Taisuke Boku[†]
and Daisuke Takahashi[†]

We have designed and implemented a software distributed shared memory (DSM) system, SCASH-MPI, using MPI as its communication layer of SCASH DSM. By using MPI as a communication layer, we can exploit high performance network of several clusters and high portability. While the implementation which uses PM 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. The experiment on 4 nodes shows the laplace benchmark with SCASH-MPI achieves comparable performance to the original SCASH. And the performance degradation is about 6.3% with NPB BT Class B.

## 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っている．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，ソフトウェア分散共有メモリシステム 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)~3)]．

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やすく，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を可能にする．

マイクロプロセッサの高速化，低価格化と Gigabit Ethernet などの高速なコモディティネット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

† 筑波大学大学院システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba

ラムを共有メモリモデルで実行する場合の本質的な問題点は広大な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 bit 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．しかし近年では Intel の Itanium2 や AMD の Opteron など 64 bit 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．

このような背景をふまえ，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 クラスタシステムソフトウェアの PM 通信レイヤを用いている．PM は効率的な通信を提供するだけでなく，ソフトウェア分散共有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ければならない．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 bit アドレス空間で大規模なプログラムを実行する際には，全ノードに大量のメモリを装備するのはかなり困難な問題となりうる．また，PM を利用するためには SCore が必要となるが，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．

本稿では通信レイヤ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．現在，ほとんどの高速なクラスタ用ネットワークでは MPI がサポートされており，通信レイヤに MPI を用いることで，PM がサポートされないネットワークも利用することができるようになる．本稿では通信のためのスレッドを作成し，それが MPI 関数を用いて通信することによりソフトウェア DSM システムの通信を実現する．デーモンプロセスを作り，それが MPI で通信することでソフトウェア DSM システムを実現した例はあるが[4]，それは共有データにライブラリを用いてアクセスする DSM であり，共有データへのアクセスを明示的に記述しなければならない．SCASH のようなページベースの DSM を MPI を用いて実装した例は報告されていない．

我々はいくつかの実装方法について検討し，通信にスレッドを用いる 2 つの実装方法を実装した．その性能評価を行い，従来の SCASH との性能差を明らかにした．提案システムでは通信のためのスレッドを作るため，それが CPU リソースを占有してしまうという問題点がある．1 つのプロセッサで仮想的な複数のスレッド実行を行うことを可能にする HyperThreading の機能がその問題点を解決するためにどれだけ有効であるかについても検証する．

これにより，可搬性に優れた，また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い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．

次章で，背景と動機について述べ，3 章において MPI を通信レイヤに用いたソフトウェア分散共有メモリシステムについて述べる．4 章と 5 章では，その性能評価を行い，従来の SCASH との比較を行う．6 章でこれからの課題について述べる．7 章で関連研究について述べ，最後の 8 章でまとめる．

## 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ェア分散共有メモリシステム SCASH とそれに対して 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 ソフトウェア分散共有メモリシステム：SCASH

SCASH は，新情報処理開発機構[5] 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．低レイテンシかつ高バンド幅を提供する高速通信ライブラリ PM[6] 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．

PM では，従来のメッセージパッシングによる通信のほかに，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，ノードごと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

ユーザは SCASH のライブラリ関数を直接用いて

プログラミングすることもできるが，Omni/SCASH を用いて，OpenMP で並列化されたプログラムを SCASH 上で実行することもできる．Omni/SCASH は分散メモリシステム向けの Omni [7] 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にアクセスすると，ページフォルトが起きる．ページフォルトが起きると，そのページのデータが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フォルトを起こしたノードに転送される．またバリア同期実行時に，ページのホームノードに対して，更新したメモリの内容を転送する．したがって，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，ノード間の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提供し，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，データの割当てに合わせてループの割当てを行う 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．クラスタのノードとして 64 bit 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，実メモリが必要になるという制限は大きな障害となりうる．ただし，ピンダウンしないとコピーしなければならず，効率が低下する．つまり，効率と利便性とのトレードオフの関係がある．

この問題点を解決するために，従来の SCASH のプロトコルを Home-based にし，ホームのみをピンダウンするようにすることが考えられる．それにより，台数に比例した共有メモリを確保できる．リモートメモリアクセスの対象となるホームはピンダウンされているため PM のリモートメモリアクセス通信を直接利用できる．しかし，この手法ではホームのマッピングが動的に変更される場合などにその管理が煩雑になる．また，PM を使用するため SCore に依存してしまうという問題点が残る．

## 3. MPI を通信レイヤに用いたソフトウェア分散共有メモリシステム

### 3.1 設 計 方 針

これまでに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 つの通信を使用している．メッセージ通信については，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ついては，以下の方法を検討する．

(i) MPI2 のリモートメモリ通信機能を用いる方法．
(ii) 計算とは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．
(iii) 同じくスレッドを作るが，MPI のメッセージ通信をこのスレッドに行わせることにより，MPI を用いてリモートメモリ通信をサポートする．

まず，(i) であるが，MPI2 のリモートメモリ通信では，通信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，性能について明らかでない点がある．(ii) 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 以外の様々な高速な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ても，TCP/IP では利用できないことがある．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 のプロトコルオーバヘッドがあり，高速ネットワークの性能を十分に活用できないことがある．結論として，(iii) の実装を用いることにした．なお，DSM のプロトコルは Omni/SCASH で用られる 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作成し，そのスレッドが MPI を用いて通信することで SCASH の通信レイヤを実現する．スレッドの利用方法として，以下の 2 つの方法を実装した．

（1）送信は計算スレッドが行い，通信スレッドは受

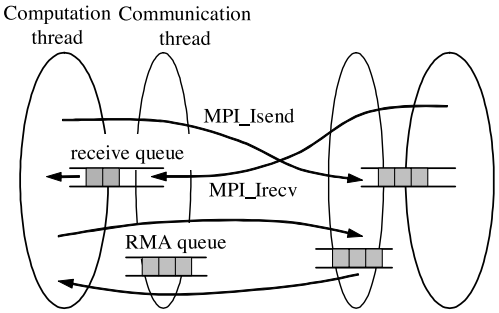


**図 1** SCASH-MPI の動作の概略（実装方法 1）
Fig. 1 The overview of communication in SCASH-MPI (1).

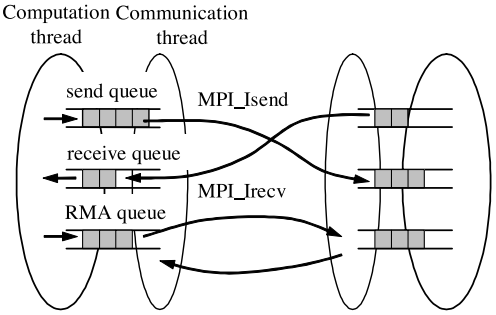


**図 2** SCASH-MPI の動作の概略（実装方法 2）
Fig. 2 The overview of communication in SCASH-MPI (2).

信処理を行う．

（2）通信スレッドがすべての通信を行う．

（1）では，メッセージの送信，リモートメモリアクセス要求送信，およびそれに付随するデータの送受信を計算スレッドが処理する．通信スレッドは受信したメッセージおよびリモートメモリアクセス要求を処理する．

データの送信は計算スレッドが `MPI_Isend` を用いて行う．通信スレッドは `MPI_Irecv` で受信を行う．`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キューに加える．計算スレッドはそこからデータを取り出し，処理を進める．リモートメモリアクセスは，まず計算スレッドが `MPI_Isend` で要求を送信する．要求を受けた側の通信スレッドはそれを `MPI_Irecv` で受信し，もしリード要求であれば要求されたページのデータを `MPI_Send` で送信する．ライト要求であれば `MPI_Recv` で受信したデータをメモリに書き込む．動作の概略を**図 1** に示す．

この実装方法ではデータの送信を計算スレッドが直接処理することができる．しかし，計算・通信両方のスレッドで MPI 関数が実行されるため，thread-safe な MPI 実装を用いるか，あるいは thread-safe でない MPI 実装では MPI 関数をロックで排他制御する必要がある．現在広く普及している MPI の実装は thread-safe でないため，ロックを使用する手法を用いた．また，キューも 2 つのスレッドにより操作されるため，ロックで排他制御される．

（2）では，すべての通信を通信スレッドが行う．計算スレッドと通信スレッド間のデータのやりとりはキューを介して行う．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キューに加える．通信スレッドはそれを監視し，送信すべきデータがあればそれをキューから取り出し，送信する．リモートメモリアクセスも同様に，計算スレッドが要求をキューに入れ，

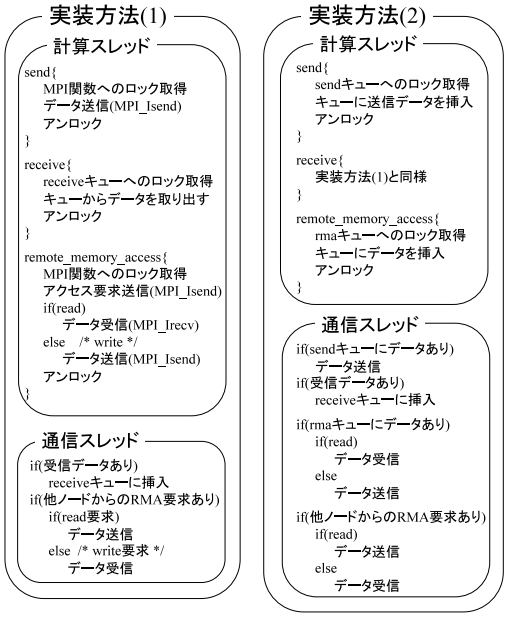


**図 3** SCASH-MPI のアルゴリズムの概要
Fig. 3 The overview of communication algorithm in SCASH-MPI.

通信スレッドがそれを取り出し送信する．受信の処理は（1）と同様である．通信スレッドのみが MPI 関数を実行するため，MPI 関数をロックで排他制御する必要はなくなるが，通信する際には必ずキューを介さなければならない．キューはロックで排他制御される．動作の概略を**図 2** に示す．また 2 方式のアルゴリズムの概要を**図 3** に示す．

### 3.3 スレッドを用いた通信の利点・欠点

提案する方式では，通信スレッドがつねに受信の有無やキューを監視するため，CPU リソースを占有してしまうという問題点がある．しかし，Intel の CPU に実装されている HyperThreading などの SMT の機能を用いることにより，このようなスレッドのオーバ

**表 1** Itanium クラスタの構成
Table 1 The configuration of Itanium cluster.

| | |
|---|---|
| CPU | Itanium 800 MHz |
| ノード | 4-way SMP |
| Memory | 32 GB |
| ノード数 | 4 |
| Network | 1000 Base-T Ethernet |
| OS | Linux 2.4.21 |
| SCore | version 5.7 |
| MPI | LAM 6.5.4 |
| | MPICH-PM 1.2.5 |
| コンパイラ | gcc 3.2.2 (-O3) |

ヘッドを低減させることができると考えている．

HyperThreading では，それによって有効になった論理 CPU を計算に利用すると，メモリのバンド幅の飽和などの問題から性能が低下してしまうことがある．そこで通信に利用することで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 SMP PC を用いることが多いが，メモリバンド幅など，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，通信を片方の CPU で実行することにより，効率的に処理できると期待できる．

## 4. 基本性能の比較

SCASH システムにおけるリモートメモリアクセスのコストおよ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．リモートメモリアクセスのコストとは，リモートメモリリード要求を出してから要求したページデータの取得が完了するまでの時間である．またバリア同期のオーバヘッドとは 1 回のバリア同期実行にかかる時間であり，更新した共有データをそのホームノードへ送信する処理や，ページの invalidate 処理などを含む．また，各プロセスがバリア同期ポイントへ到達するまでの時間のずれも含む．

### 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．なお，ページサイズは 16 KB である．

### 4.2 使用プログラム

実装方法 (1), (2) それぞれを実装し，システムの基本性能の比較を行った．具体的にはリモートメモリリード通信のコストおよびバリア同期のオーバヘッドを測定した．ベンチマークには laplace を使用した．laplace は Jacobi 法によって Laplace 方程式を解くプログラムである．行列のサイズは 4096×4096 (倍精度) とし，反復回数は 10 回とした．C で，SCASH のユーザライブラリを用いて記述した．laplace をノードあたり 1 プロセスずつ，使用するノード数を 2, 4 と変化させて実行し，その際のページあたりのリモートメモリリードおよびバリア同期にかかる時間を測定し，それらの合計時間と回数からそれぞれ 1 回にかかる平均時間を算出した．MPI ライブラリとしては，MPI の通信プロトコルによる性能差を比較するため，LAM 7.0.6 (lam) および MPICH-PM 1.2.5 (mpichpm) の 2 つを使用した．また，オリジナルの SCASH (pm) とも比較を行った．通信プロトコルは lam では TCP/IP が，mpichpm および pm では PM/Ethernet が使用される．

**表 2** リモートメモリリード (RMR) のコストおよびバリア同期のオーバヘッド [msec]
Table 2 The cost of remote memory read (RMR) and the overhead of barrier synchronization [msec].

| | RMR | | Barrier | |
|---|---|---|---|---|
| nodes | 2 | 4 | 2 | 4 |
| lam(1) | 0.81 | 1.13 | 30.3 | 34.4 |
| lam(2) | 0.65 | 0.70 | 20.5 | 16.4 |
| mpichpm(1) | 0.65 | 0.79 | 25.7 | 24.0 |
| mpichpm(2) | 0.43 | 0.49 | 21.8 | 16.5 |
| pm | 0.25 | 0.28 | 20.2 | 16.8 |

### 4.3 測定結果

**表 2** に各実装でのリモートメモリリードのコストおよびバリア同期のオーバヘッドを示す．RMR がページあたりのリモートメモリリードの平均実行時間を，Barrier が 1 回のバリア同期の平均実行時間を示す．いずれも単位は msec である．lam および mpichpm の括弧内の数字は実装方法の番号を示す．リモートメモリアクセス通信では，ページ共有要求メッセージ 12 byte，リモートメモリアクセスヘッダ 16 byte およびページデータ 16 KB を送受信する計 3 回の通信が行われる．また，1 回のバリア同期でノード間で送受信される全メッセージ数は 4 ノード実行時で平均で 130 である．リモートメモリ通信のコストは，プロトコルでリモートメモリ通信をサポートする pm と比較すると，lam では約 2.5 倍以上，mpichpm では約 1.7 倍以上である．しかしバリア同期のオーバヘッドは pm よりも低く抑えることができている．

実装方法で比較すると，いずれも (2) の方が良い結果になった．両実装での 1 回のリモートメモリリード通信中にロック取得に費やされる時間を測定したところ，4 ノード実行時で (1) では平均で約 490 $\mu$s, (2) では約 70 $\mu$s であった．(1) では MPI 関数がロックで排他制御されるが，通信スレッドが受信の有無を MPI_Test でつねに監視しているため，ほぼつねに通信スレッドがロックを保持した状態になる．そのた

**表 3** 通信プロトコルおよび実装の違いによる laplace の実行時間の違い [sec]
Table 3 The difference of execution time of laplace due to the difference of the communication protocol and the implementation of the communication layer.

| | SCASH-MPI(lam) | | SCASH-MPI(mpichpm) | | SCASH-PM | | |
|---|---|---|---|---|---|---|---|
| num of processes | 1 proc/node | 2 proc/node | 1 proc/node | 2 proc/node | 1 proc/node | 2 proc/node | 4 proc/node |
| 1 | 17.12 | | 17.12 | | 17.12 | | |
| 2 | 9.55 | 9.90 | 10.00 | 9.99 | 9.80 | 9.68 | |
| 4 | 4.88 | 5.02 | 4.97 | 5.09 | 4.95 | 5.17 | 11.57 |
| 8 | | 2.66 | | 2.64 | | 2.70 | 6.17 |
| 16 | | | | | | | 3.42 |

め計算スレッドがロックを取得するのに時間がかかっている．一方，(2) ではロックで排他制御されるのはキューのみである．キューを操作する際にのみロックを取得するため競合が起こりにくい．2 方式の基本性能の差はロック取得にかかる時間によるものが大きいといえる．ロックの使用法を変更することで実装 (1) の性能を改善できる余地があると考えられるが，本稿でのこれ以降の性能評価では実装方法 (2) のみ用いて行う．

## 5. 性 能 評 価

### 5.1 対象プログラム

性能評価の対象プログラムには，前述の laplace および NPB BT を使用した．NPB 2.3 の BT を長谷川らが OpenMP を用いて並列化および Omni/SCASH 向けに最適化したもの[8] を用いた．最適化には，マッピング指示文および affinity スケジューリングが使用されている．また，affinity スケジューリングの使用により不要になったバリア同期を削減している．問題サイズは Class A および B を使用した．また，実行されるステップ数は本来 200 であるが，20 に変更して測定を行った．各ステップでのメモリアクセスパターンは同一であるため，ステップ数を変えても性能には影響しない．

### 5.2 laplace の測定結果

SCASH-MPI では 1 プロセスあたり 2 スレッド動作するため，ノード内で実行するプロセス数を 2 までとした．SCASH-MPI は SMP に対応していないため，ノード内で 2 プロセス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．SCASH-PM は 1 プロセスあたり 1 スレッド動作するため，ノード内で実行するプロセス数を 4 までとした．使用するノード数を 1，2，4 と変化させ，測定を行った．

laplace の測定結果を**表 3** に示す．通信レイヤの違いにかかわらずほぼ等しい性能が得られている．10 回の反復でのバリア同期の回数が 13 回，8 プロセス (2 process/node × 4 node) 実行時にリモートメモリリード (RMR) で転送されるページ数が，SCASH-MPI では全プロセスで計 315，SCASH-PM では計 135 と少なく，基本性能の違いによる実行時間の差がわずかなためである．SCASH-MPI と SCASH-PM で RMR で転送されるページ数が異なるのは，SCASH-PM は 1 つのノード内のプロセスはメモリ空間を共有できるためである．ノード内の 1 つのプロセスが RMR によって取得したページをノード内の他のプロセスが利用することができる．一方，SCASH-MPI は SMP に対応しておらず，メモリ空間は各プロセスで独立であるため，ノード内の各プロセスがそれぞれ RMR を行う必要がある．

**表 4** 8 プロセス実行時の laplace の基本性能
Table 4 The basic performance on execution with 8 processors.

| | lam | mpichpm | SCASH-PM |
|---|---|---|---|
| Barrier[msec] | 17.0 | 15.0 | 23.4 |
| RMR[msec] | 0.95 | 0.52 | 0.30 |

8 プロセス実行時の基本性能を**表 4** に示す．lam および mpichpm は RMR のコストは SCASH-PM よりも大きいが，バリア同期のオーバヘッドが小さいため，SCASH-PM よりも実行時間が短くなっている．

SCASH-PM でノードあたり 4 プロセス実行させると，2 プロセス実行したときよりも性能が低下している．これはメモリバンド幅の飽和によるものと考えられる．このように，ノード内の全プロセッサを計算に使用しても良い性能が得られないことがある．そのため，提案システムではノード内のプロセッサ数の半分のプロセス数までならば効率的に実行できるが，それを超えるプロセス数では計算スレッドと通信スレッドが 1 つのプロセッサを共有するため処理効率が落ちるという点は大きな不利点とはならないと考えられる．

8 プロセスで，lam および mpichpm では約 6.5 倍，SCASH-PM では約 6.4 倍の性能向上が得られた．

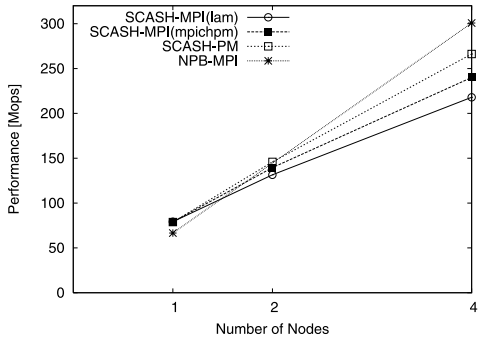


**図 4** BT の性能（Class A）
Fig. 4 The performance of BT (Class A).

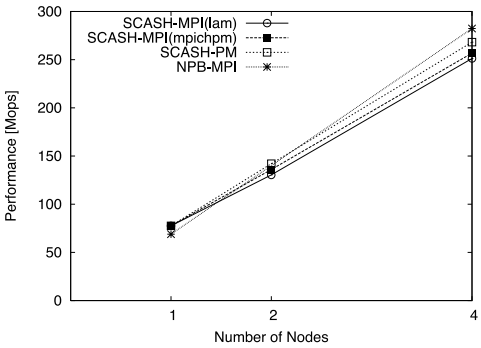


**図 5** BT の性能（Class B）
Fig. 5 The performance of BT (Class B).

**表 5** BT Class A 4 ノード実行時の基本性能
Table 5 The basic performance in BT Class A on execution with 4 nodes.

| | lam | mpichpm | SCASH-PM |
|---|---|---|---|
| Barrier[msec] | 305 | 210 | 164 |
| RMR[msec] | 0.67 | 0.50 | 0.28 |

### 5.3 BT の測定結果

ノード内で 1 プロセスずつ実行させて評価を行った．**図 4**，**図 5** に BT の Class A，B での測定結果を示す．参考として MPI 版 NPB BT を測定した結果も示す．

Class A では，4 ノードで，SCASH-MPI（lam）で約 2.8 倍，SCASH-MPI（mpichpm）で約 3.0 倍，SCASH-PM で約 3.4 倍の性能向上が得られた．その際の基本性能を測定した結果を**表 5** に示す．1 回のバリア同期中に全ノード間で送受信される平均メッセージ数が 3,020 と多いため，通信プロトコルの性能，通信方法の違いによってバリア同期のオーバヘッドの差が大きい．SCASH-MPI（mpichpm）では SCASH-PM の約 90%，SCASH-MPI（lam）では約 82%の性能が得られた．

Class B では，4 ノードで，SCASH-MPI（lam）で 3.2 倍，SCASH-MPI（mpichpm）で 3.3 倍，SCASH-PM で 3.5 倍の性能向上が得られた．4 ノード実行時の基本性能を**表 6** に示す．各プロセスがバリア同期ポイントに到達するまでの時間のずれが大きく，また 1 回のバリアあたりの平均メッセージ数が 11,079 と非常に多いため，バリア同期の平均オーバヘッドが大きくなっている．また，RMR のコストもやや大きくなっている．それにもかかわらず Class A より良いスケーラビリティが得られたのは，計算量のオーダが $O(n^3)$ なのに対して通信量のオーダが $O(n^2)$ であるためである．Class A と B では計算量では約 4 倍の違いがあるが，リモートメモリリードで転送された全ページ数を比較すると Class A では 7,892，B では 24,981 と，約 3.2 倍の違いにとどまっている．そのため Class B では通信時間が相対的に小さくなり，Class A よりも良いスケーラビリティが得られた．pm と比較すると，lam では約 94%，mpichpm では約 96%の性能が得られた．

**表 6** BT Class B 4 ノード実行時の基本性能
Table 6 The basic performance in BT Class B on execution with 4 nodes.

| | lam | mpichpm | SCASH-PM |
|---|---|---|---|
| Barrier[msec] | 2232 | 1741 | 422 |
| RMR[msec] | 0.85 | 0.52 | 0.27 |

### 5.4 HyperThreading の利用

提案するシステムでは，1 プロセスあたり，計算と通信の 2 つのスレッドが動作する．そのため，ノード内のプロセッサ数と同じ数のプロセスを実行した場合，計算スレッドと通信スレッドが 1 つのプロセッサを共有することになり，各スレッドが 1 つのプロセッサを占有できる場合と比較すると性能が低下すると考えられる．そこで，HyperThreading（以下 HT）が性能低下を抑えるのにどれだけ有効であるかを測定した．

測定には Pentium Xeon クラスタを使用した．各ノードは HyperThreading 機能を持つ Xeon 2.4 GHz（L2 キャッシュ512 KB）を 2 つ搭載し，メモリは 1 GB である．ノード間は 1000 base-T Ethernet で接続されている．HT ON，OFF それぞれの状態で laplace を測定した．行列サイズは $4096 \times 4096$，反復回数は 10 とした．

測定結果を**図 6** に示す．参考として MPI 版 laplace を測定した結果も示すが，本測定での目的は HyperThreading の有無による，通信スレッドによるオーバヘッドの違いの測定である．図中において，1 proc/node とは 1 ノードあたり 1 プロセス実行させたことを意味する．その場合計算と通信で計 2 スレッド動作するが，各スレッドは 1 つの CPU を占有

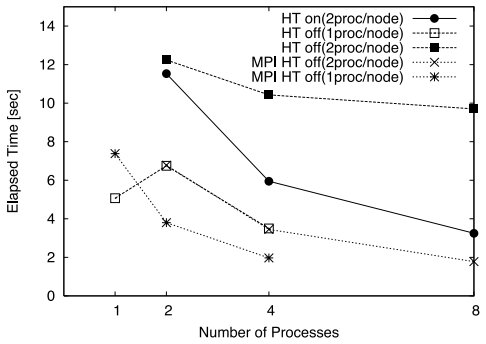


**図 6** HyperThreading の有無による laplace の実行時間の違い
Fig. 6 The difference of execution time of laplace in HyperThreading ON/OFF.

**表 7** 8 プロセス実行時の laplace の基本性能 [msec]
Table 7 The basic performance in laplace on execution with 8 processes [msec].

| | HT OFF | HT ON |
|---|---|---|
| RMR | 8.54 | 0.725 |
| Barrier | 590 | 51.2 |

できる．また 2 proc/node とは 1 ノードあたり 2 プロセス実行させたことを意味する．その場合，計算と通信で計 4 スレッドが動作する．HT OFF では 4 スレッドが 2 つの物理 CPU を共有するが，HT ON では各ノードは仮想的に 4CPU のシステムになるため，各スレッドは 1 つの論理 CPU を占有できる．8 プロセスで実行時の基本性能を測定した結果を**表 7** に示す．

測定の結果，以下のことが分かった．

- HT OFF 1 proc/node と HT ON 2 proc/node で同プロセス数での性能を比較すると，HT ON の方が実行時間が長くなっている．たとえば 4 プロセスの場合，HT OFF 1 proc/node では 4 ノード，HT ON 2 proc/node では 2 ノード使用され，HT OFF の方がノード間通信は多い．しかし，ノード間通信が少ない HT ON の方が実行時間が長くなっている．これはメモリバンド幅の不足，また HT を ON にしたことによるキャッシュミス率の増加によるものと考えられる．
- HT ON 2 proc/node と OFF 2 proc/node で比較すると，HT を有効にすることでオーバヘッドを削減できている．HPC アプリケーションでは HT を ON にして有効になった CPU を利用すると，キャッシュミス率の増加，メモリの競合，メモリバンド幅の飽和などの理由により，性能が低下することがあることが報告されている[9]．提案するシステムでは，計算のみ行うスレッドと通信のみ行うスレッドという性質の異なるスレッドを利用することでメモリバンド幅の飽和などを抑えることができ，HT を有効に活用できると期待できる．
- HT を ON にすることで，RMR のコストおよびバリア同期のオーバヘッドは 10 分の 1 以下に削減されている．8 プロセス実行時に HT OFF では実行時間の約 79%をバリア同期処理時間が占めるが，HT ON ではそれが削減され，実行時間が HT OFF でのものの約 3 分の 1 になっている．
- HT OFF では最大で約 1.45 倍の性能向上だったが，HT ON では約 1.56 倍の性能向上が得られた．

## 6. 課　　題

### 6.1 Prefetch による効率化の検討

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，通信性能が十分でない Ethernet では効果を得るのが難しい．

そこで，prefetch を実装することを検討している．その方式としては以下の 2 つを検討している．

- コンパイラで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ラップさせる手法．
- 1 バリア同期間でのアクセス履歴をとり，それに従って prefetch する方法．これは，laplace など，同じメモリ領域に繰り返しアクセスするプログラムに有効であると考えられる．

コンパイラによるアクセス領域の解析については[10]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．

1 prefetch の指示により，通信スレッドに prefetch 領域を伝える．
2 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 の通信を開始する．
3 通信スレッドが prefetch 対象ページのデータを受け取り，ページの DSM の管理属性を Read Only に変え，ページを unlock する．
4 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．
  – もしページが lock されていれば通信スレッドが当該ページを 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．

– ページがすでに prefetch されていれば，ページのハードウェ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同期のオーバヘッドが小さい，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．Dwarkadas ら[11] 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．また丹羽[12] は，同様な手法で DSM を実現した他，それに対して prefetch 機構の実装も行っている．彼らは計算と prefetch 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 予備評価

予備実装としてプログラマが prefetch 対象ページを指定する方式を実装し，その評価を行った．プログラマが prefetch 対象領域を指定するためのライブラリ関数 `scash_prefetch(caddr_t addr, int pages)` を実装した．`addr` は prefetch 対象領域の先頭アドレス，`pages` はページ数である．`addr` を含むページから `pages` ページ分が prefetch 対象となり，prefetch 通信が開始される．使用例を以下に示す．以下の例では配列 `u` のデータ分割の境界部分が prefetch 対象として指定される．

```
pages = XS*sizeof(double)/SCASH_PAGE_SIZE;
scash_prefetch(&u[(y_from-1)*XS], pages);
scash_prefetch(&u[(y_to)*XS], pages);

for(y = y_from; y < y_to; y++){
    for (x = 1 ; x <= XSIZE ; x++)
        uu[y*XS + x] = u[(y-1)*XS + x] + ...
}
```

評価環境として 5 章で使用した Itanium クラスタを用い，ベンチマークプログラムには laplace を使用した．問題サイズは 4096*4096，反復回数は 10 とした．また MPICH-PM 1.2.5 を用いた．ノードあたり 1 プロセス実行し，4 ノード使用して測定した．10 回の反復実行にかかる時間と，ページフォルトハンドラ内でのページデータの取得待ち時間を測定した．待ち時間とは，prefetch をしない場合はリモートメモリリード要求を出してからページデータの取得が完了するまでの時間，prefetch する場合は対象ページの prefetch が完了するのを待つ時間を意味する．

**表 8** に測定結果を示す．“prefetch あり” では，他のノードから転送してくる必要のあるページすべてが prefetch 対象領域として指定される．prefetch を行うことでページデータの取得待ち時間をなくすことができており，それによって全体の実行時間を削減できている．様々なアプリケーションでの評価，アクセス履歴を用いて prefetch 対象領域を決定する方式およびコンパイラでアクセス範囲を解析する方式の実装が今後の課題である．

**表 8** prefetch の有無による laplace の実行時間の違い
Table 8 The exection time of laplace with/without prefetching.

| | prefetch なし | prefetch あり |
|---|---|---|
| 全体の実行時間 [sec] | 4.604 | 4.584 |
| ページ取得待ち [msec] | 22.87 | 0 |

### 6.3 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 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 を用いた場合の SCASH-PM との性能差について評価する必要がある．

## 7. 関連研究

Nieplocha らは分散システム上に共有メモリプログラミング環境を提供するシステム Global Arrays[13] を実装した．ARMCI というリモートメモリアクセスライブラリを用いて実装されているが，デフォルトでは TCP/IP を用いて PUT，GET が実装されている．

Silva らは MPI 上に DSM ライブラリ DSMPI[4] を実装した．アプリケーションプロセスのほかにデーモンプロセスを作り，それが複製したデータや一貫性プロトコルを管理することでソフトウェア DSM システムを実現している．通信にデーモンプロセスを用いているため，それと計算プロセスとの間の通信はプロセス間通信で行わなければならないが，我々はスレッドを用いているため，データの交換にハードウェア共有メモリを用いることができる．また DSMPI は共有データにライブラリを用いてアクセスする DSM であり，共有データへのアクセスを明示的に記述しなければならない．また，このような共有データへのアクセスを明示的に記述しなければならないシステムの上で OpenMP を実装することは困難である．

Hu らは通信用のスレッドを用いて通信を行う機構

を JIAJIA に実装した[14]．ソケットを用い，UDP で通信を行っている．TCP よりは小さいものの，プロトコルのオーバヘッドが存在する．そのためクラスタネットワークが有効に活用できないことがある．

## 8. 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 SCASH-MPI を実装した．基本性能を比較し，またアプリケーションを用いて性能を評価した結果，以下のことが分かった．

- 送信は計算スレッドが行い，受信を通信スレッドが行う実装方法（1）と，通信スレッドがすべての通信を行う実装方法（2）を比較すると，実装方法（1）は（2）と比較してロック取得にかかる時間が長く，改善の必要がある．
- リモートメモリリードのコストは，LAM を用いた場合は従来の SCASH の 2.5 倍以上，MPICH-PM を用いた場合は 1.7 倍以上となり，PM のリモートメモリアクセス機能を利用する従来の SCASH と比較するとコストが高い．
- 性能評価の結果，laplace では従来の SCASH と同等の性能を得ることができた．また NPB BT Class B では従来の SCASH と比較したときの性能の低下を約 6%に抑えることができた．
- 計算と通信で 1 つのプロセッサを共有してしまう場合，HyperThreading を有効にすることでそのオーバヘッドを削減できることが分かった．
- prefetch の予備実装，評価を行った結果，レイテンシの隠蔽に有効であることが分かった．

今後の課題は，まず改善の必要があることが分かった実装方法（1）を改善することである．thread-safe な MPI 実装を利用することができれば MPI 関数をロックで排他制御する必要がなくなり，性能を改善できると期待できる．また prefetch について，アクセス履歴を用いる方法およびコンパイラでアクセス領域を解析する方法をそれぞれ実装し，評価することがあげられる．またクラスタ用の高速ネットワークを利用して測定・評価を行うことも課題の 1 つである，

**謝辞** 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) 小島好紀，佐藤三久，朴　泰祐，高橋大介：Omni/SCASH における First Touch page allocation の実装，情報処理学会研究報告 2003-ARC-154 (SWoPP 2003), pp.145–150 (2003).
4) Silva, L.M., Chapple, S. and Silva, J.G.: Implementing Distributed Shared Memory on top of MPI: the DSMPI Library, *Proc. 4th Euromicro Workshop on Parallel and Distributed Processing* (*PDP '96*), pp.50–57 (1996).
5) PC クラスタコンソーシアム. http://www.pccluster.org
6) 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, pp.708–719, Springer-Verlag (1997).
7) Omni OpenMP Project. http://www.hpcc.jp/Omni.
8) 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).
9) Leng, T., Ali, R., Hsieh, J., Mashayekhi, V. and Rooholamini, R.: An Empirical Study of Hyper-Threading in High Performance Computing Clusters, *3rd LCI International Conference on Linux Clusters: The HPC Revolution 2002* (2002).
10) 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).
11) Dwarkadas, S., Lu, H., Cox, A.L., 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).
12) 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).
13) Nieplocha, J., Harrison, R.J. and Littlefield, R.J.: Global Arrays: A Non-Uniform-Memory-Access Programming Model For High-Performance Computers, *The Journal of Supercomputing*, Vol.10, pp.197–220 (1996).
14) Hu, W., Shi, G. and Zhang, F.: Communication with Threads in Software DSMs, *Proc. International Conference on Cluster Computing 2001* (*CLUSTER'01*), pp.149–154 (2001).

(平成 16 年 10 月 4 日受付)
(平成 17 年 2 月 24 日採録)

**小島 好紀**（学生会員）

昭和 54 年生．平成 14 年筑波大学第三学群情報学類卒業．平成 16 年同大学大学院修士課程理工学研究科修了．現在，同大学院博士課程システム情報工学研究科在学中．クラスタコンピューティングに関する研究に従事．

**佐藤 三久**（正会員）

昭和 34 年生．昭和 57 年東京大学理学部情報科学科卒業．昭和 61 年同大学大学院理学系研究科博士課程中退．同年新技術事業団後藤磁束量子情報プロジェクトに参加．平成 3 年通産省電子技術総合研究所入所．平成 8 年新情報処理開発機構並列分散システムパフォーマンス研究室室長．平成 13 年より，筑波大学システム情報工学研究科教授．同大学計算科学研究センター勤務．理学博士．並列処理アーキテクチャ，言語およびコンパイラ，計算機性能評価技術，グリッドコンピューティング等の研究に従事．IEEE，日本応用数理学会各会員．

**朴 泰祐**（正会員）

昭和 59 年慶應義塾大学工学部電気工学科卒業．平成 2 年同大学大学院理工学研究科電気工学専攻後期博士課程修了，工学博士．昭和 63 年慶應義塾大学理工学部物理学科助手．平成 4 年筑波大学電子・情報工学系講師，平成 7 年同助教授．平成 16 年同大学大学院システム情報工学研究科助教授，平成 17 年同教授，現在に至る．同大学計算科学研究センター勤務．超並列計算機アーキテクチャ，ハイパフォーマンスコンピューティング，クラスタコンピューティング，グリッドに関する研究に従事．平成 14 年度および平成 15 年度情報処理学会論文賞受賞．日本応用数理学会，IEEE CS 各会員．

**高橋 大介**（正会員）

昭和 45 年生．平成 3 年呉工業高等専門学校電気工学科卒業．平成 5 年豊橋技術科学大学工学部情報工学課程卒業．平成 7 年同大学大学院工学研究科情報工学専攻修士課程修了．平成 9 年東京大学大学院理学系研究科情報科学専攻博士課程中退．同年同大学大型計算機センター助手．平成 11 年同大学情報基盤センター助手．平成 12 年埼玉大学大学院理工学研究科助手．平成 13 年筑波大学電子・情報工学系講師．平成 16 年同大学大学院システム情報工学研究科講師．博士（理学）．並列数値計算アルゴリズムに関する研究に従事．平成 10 年度情報処理学会山下記念研究賞，平成 10 年度，平成 15 年度情報処理学会論文賞各受賞．日本応用数理学会，ACM，IEEE，SIAM 各会員．