2003－ARC－154　(25)
2003／8／6

Omni/SCASH における First Touch page allocation の実装

小島好紀[†]　佐藤三久[††]
朴泰祐[††]　高橋大介[††]

Omni/SCASH は，ソフトウエア分散共有メモリシステム SCASH を用いたクラスタ向けの OpenMP 処理系である．本稿では，Omni/SCASH へ First Touch page allocation の機能を実装し，評価を行った．NPB の BT 及び SP を用いて性能評価を行った結果，16 プロセッサでの実行で BT ではホームノードをブロック分割で割り当てたものの 1.3 倍，最適に割り当てたものの 88%の性能，SP ではブロック分割方式の 1.17 倍，最適な方式の 93%の性能が得られた．SP での 16 プロセッサ実行時のホームノード割り当て結果は，最適な方式とブロック分割方式では，全体の 65%のページのホームノードが異なっていたが，first touch 方式では 10%に抑えられ，ブロック分割方式よりも適切にホームノード割り当てが行えることが分かった．

Implementation of First Touch page allocation on Omni/SCASH

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

Omni/SCASH is an OpenMP implementation for SCASH software distributed shared memory system. In this paper, we report the implementation of First Touch page allocation on Omni/SCASH and its performance. The performance evaluation using the BT and SP from NPB2.3 on 16 processors shows BT under the First Touch page allocation is 1.3 times faster than that under block distribution and 0.88 times faster than that of statically optimized home node. SP under the first touch method is 1.17 times faster than that under block distribution and 0.93 times faster than that of optimal method. The difference of home node allocation of SP between the optimal method and block distribution method is 65% of all pages, but the difference between optimal method and first touch method is only 10%. It is shown that first touch method can allocate home node more appropriately than block distribution method.

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．

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，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 SCASH や TreadMarks[1] などの，分散メモリマシンの上でソフトウェアで共有アドレスを実現するソフトウェア分散共有メモリ（Ditributed Shared Memory: DSM）システムが研究・開発されている．

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．良い性能を得るためには，データのローカリティーを高め，できる限りノード間通信を減らすことが重要である．SCASH 向けの OpenMP 処理系である Omni/SCASH[2] は，ホームノード割り当て手法としてブロック分割とサイクリック分割をサポートしているが，複雑なデータ分散が必要なアプリケーションではプログラマが適切な分割を指定しなければならず，プログラミングのコストが高い．また実行前にホームノード割り当てを決定するため，与えるデータによっ

† 筑波大学大学院理工学研究科
Master's Program in Science and Engineering, University of Tsukuba
†† 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba

てデータのアクセスパターンが変わるプログラムにも適切に対応することができないなどの問題点がある．

そこで本稿ではデータに初めてアクセスしたときのアクセスパターンに合わせてホームノード割り当てを行う，First Touch page allocation を Omni/SCASH に実装し，性能を評価する．First Touch page allocation とは，初めにそのページにアクセスしたノードがそのホームノードになるという方式である．first touch は IRIX 等でも実装されているが，ソフトウェア DSM で実装されている例はほとんどない．

以下，2 章でソフトウェア分散共有メモリ SCASH の概要を述べる．次の 3 章で First Touch page allocation とその実装について述べる．4 章でその性能評価を行う．5 章で関連研究について触れ，最後の 6 章で結論を述べる．

2. ソフトウエア分散共有メモリシステム SCASH

2.1 SCASH

SCASH は，クラスタシステム SCore[3] 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 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）[4] 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．

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られる．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 で並列化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．

また，SCASH のユーザライブラリを用いてプログラムを記述することも可能である．その場合，データの確保や各ノードへの分配，ループの配置等はプログラマが明示的に指定しなければならない．データの配置やループの配置をより柔軟に決定できるが，逐次版のプログラムを変更しなければならず，OpenMP でプログラムを並列化するよりもプログラミングのコストは高い．

2.2 Omni/SCASH のページ割り当て手法

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 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，更新したページのデータをホームノードに転送する．SCASH 上で良い性能を得るためには，データにアクセスするノードとデータのホームノードをできるだけ一致させ，ノード間通信を減らす必要がある．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．そこで，Omni/SCASH は独自の拡張としてデータ配置指示文をサポートしている．ホームノード割り当て方式としては，ブロック分割とサイクリック分割をサポートしている．以下に mapping 指示文の使用例を示す．

```
double A[100][200];
#pragma omni mapping(A[block][*])
```

この例では，2 次元配列 A を 1 次元目でブロック分割し，ホームノードを割り当てることを指示している．mapping が指定されなかった時は，配列全体をブロック分割してホームノードを割り当てる．

また，データの配置に合わせてループの配置を行えるようにするため，affinity スケジューリングを実装している．affinity スケジューリングの使用例を以下に示す．

```
#pragma omp for schedule(affinity, A[i][*])
  for(i = 1; i < 99; i++)
    for(j = 0; j < 200; j++)
      A[i][j] = ...;
```

この例では，for ループのループ範囲は，A[i][*] のホームノードと同じノードに割り当てられる．

Omni/SCASH のページ割り当て手法はいずれも静的なものであり，複雑なデータ分散が必要なアプリケーションではプログラマが適切な分割を指定しなければならない．そのためにはプログラマがアプリケーションのアクセスパターンについて知っていなければならず，プログラミングのコストも高い．また事前にアクセスパターンを知ることのできないアプリケーションにも対応することができないなどの問題点がある．

3. First Touch page allocation

3.1 概　　要

First Touch page allocation とは，最初にページに touch（書き込みまたは読み込み）したノードがそのページのホームノードになるという方式である．複雑なデータ分散が必要なアプリケーションの場合，それを静的に適切に割り当てることは困難である．しかし first touch 方式ならば実行時のアクセスパターンを用いて自動的に割り当てられるため，複雑な記述をせずに適切に割り当てることができる．通常，プログラムが適切に並列化されている時には，同じプロセッサ

が同じ領域にアクセスすることが多いため，アクセスパターンに応じて割り当てることで性能向上が期待できる．

3.2 実装方法

SCASHでは，各ノードが各ページとそのホームノードの対応を示すテーブルを持っている．他のノードがホーム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．そこでそのテーブルを用いてページフォルトを起こしたページのホームノードを調べ，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．

ページフォルト時の処理を変更することで First Touch page allocation の機能を実装する．アルゴリズムの概要を**図 1** に示す．

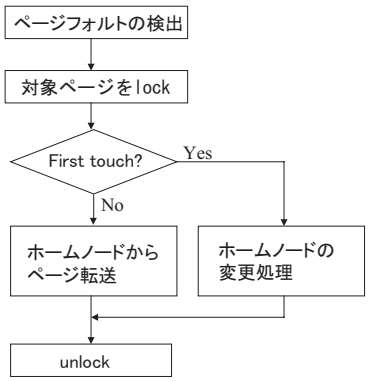

図 1 ページフォルト時の処理の概要

3.2.1 API

SCASH のユーザライブラリに，first touch 方式でホームノード割り当てを行うことを指示する scash_home_first 関数を追加した．これが Omni/SCASH からの実行時ルーチンから呼ばれる．

```
scash_home_first(caddr_t pagep, int pages)
```

pagep は first touch 方式で割り当てる領域の先頭アドレス，pages は割り当てるページ数である．すなわち，アドレス pagep を含むページから pages ページ分が first touch 方式で割り当てられる．

scash_home_first 関数が呼ばれると，全ノードのページ-ホームノード対応テーブル中の該当ページのホーム番号が，まだホームノードが割り当てられていないことを示す値に置き換えられる．それにより，以降の処理中に起きたページフォルトが first touch であるかどうかを判断する．

3.2.2 first touch の判定

ページフォルトを捕捉した時にそのテーブルを参照し，ページフォルトを起こしたページのホームノードを調べる．もしホーム番号がホームノードが割り当てられていないことを示す値であったらそのアクセスは first touch であると判断し，ホームノード変更処理を行う．ホームノード変更処理は既存の SCASH の API を使用する．

3.2.3 排他制御

ホーム割り当てを 1 つのノードが一元管理することで排他制御を行う．アルゴリズムの概要を以下に示す．

要求ノード側

```
・first touch の検出
・管理ノードへホーム管理要求メッセージを送信

・管理ノードからの返信を受信
・受信した ID を調べる
・if(自ノードの ID)
    /* 自ノードがホームノードして管理する */
    ・ホームノード変更処理
  else
    /* すでに他のノードによって管理されている */
    ・ホームノードからページ転送処理
```

管理ノード側

```
・要求ノードからの要求を受信
・対象ページのホームノード番号を調べる
・if(ホームノードが割り当てられて
     いないことを示す値)
    /* すなわち要求ノードによるアクセスが
       そのページへの first touch */
    ・テーブル中の対象ページのホームノード番号を
      要求ノードの ID に変更
    ・要求ノードのノード ID を要求ノードへ返す
  else
    /* すでにホームノードが割り当てられている */
    ・ホームノードの ID を返す
```

例えば，あるページへの first touch が複数のノードでほぼ同時に起きたとしても，それらのノードは管理ノードに問い合わせた後でホームノード変更処理を行うか否かを決定するため，排他制御を実現することができる．

4. 性能評価

4.1 評価環境

評価環境には 8 ノードからなる PC クラスタ *COSMO*（Cluster Of Symmetric Multi prOcessor）を用いた．構成を**表 1** に示す．

表 1 COSMO 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 800Mbps Myrinet
OS	Linux 2.4.19
SCore	version 5.4
コンパイラ	egcs-2.91.66
最適化オプション	-O3 -funroll-loops

4.2 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，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直接記述した．ホームノードの割り当て処理のオーバヘッドを調べるため，以下の 2 通りのホーム割り当て方について評価を行った．
  - 最適な割り当て方 (opt)
    laplace での最適な割り当て方は，使用する 2 次元配列を 1 次元ブロック分割する方式である．
  - first touch 方式 (first)
    first touch 方式で割り当てる．データの初期化はループをブロック分割して並列に行われ，そのアクセスパターンは計算時のものと同じであるので，first touch のための初期化ループは必要ない．
- **NPB BT, SP カーネル**
  NPB 2.3 の Fortran 版 BT, SP カーネルを長谷川らが OpenMP を用いて並列化及び最適化したもの[6] を使用した．最適化には mapping 指示文及び 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及び no wait 指示によってバリア同期の回数が削減されている．それを元に以下の 3 通りのホームノードの割り当て方について評価を行った．問題のサイズは Class W を使用した．
  - 最適な割り当て方 (opt)
    長谷川らが並列化及び最適化を行ったものを用いた．
  - ブロック分割 (block)
    opt から mapping 指示文及び affinity スケジューリング指示文を除き，各配列がブロック分割されるようにした．また必要なバリア同期を挿入した．
  - first touch 方式 (first)
    block に first touch 方式を適用したもの．first touch のための初期化ループとして，計算を行うルーチンをプログラムの先頭で一度呼び出し，そこで各ページへの first touch が起きるよう変更を加えた．

4.3 評価結果

性能評価において，実際に使用される物理的なプロセッサ数は，ノード数とノード内のプロセッサ数の積である．ノード数を 1，2，4，8，さらに各ノード内で使用するプロセッサ数を 1way, 2way と変化させて計 8 通りの場合について測定を行った．laplace では Myrinet と Ethernet 両方を，BT と SP では Myrinet のみを使用して測定を行った．

4.4 laplace

laplace の実行時間の測定結果を**図 2**，**図 3** に示す．

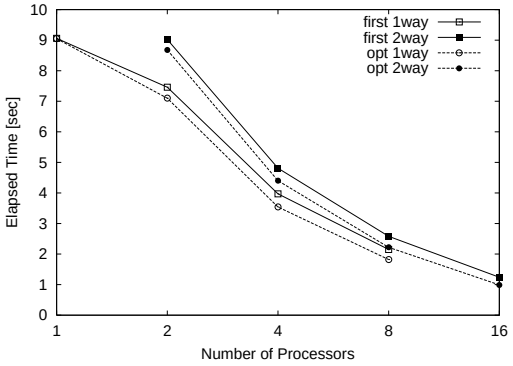

図 2 laplace の実行時間 (Myrinet)

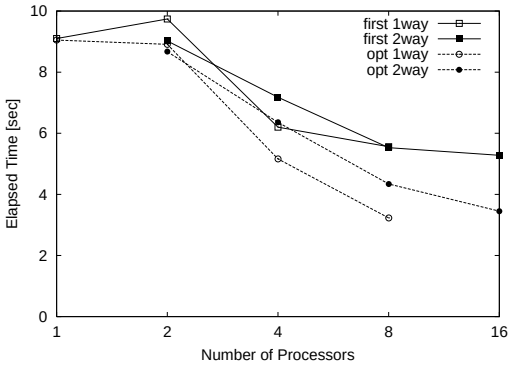

図 3 laplace の実行時間 (Ethernet)

first touch 方式でもブロック分割でホームノードが割り当てられ，最適な方式と同じホームノード割り当てになる．すなわち，2 方式の実行時間の差は，ホームノード割り当て処理のオーバヘッドである．そのオーバヘッドは，Myrinet では最大で約 0.43 秒，Ethernet では最大で約 2.33 秒と，約 5.4 倍の違いがあった．first touch 方式では，first touch を検出する度にホームノード割り当てを管理するノードとの間で通信を行うため，処理のオーバヘッドは通信性能に依存する．この差は Myrinet と Ethernet の通信性能を反映したものであると言える．管理ノードとの間の通信のラウンドトリップ時間は，16 プロセッサで Myrinet では最短で約 100μ 秒，最長で約 800μ 秒，Ethernet では最短で約 120μ 秒，最長で約 900μ 秒であった．管理ノードにメッセージが集中するため，メッセージが処理されるタイミングによってラウンドトリップ時間が大きく異なっているのだと考えられる．このオーバヘッドは first touch 検出時にのみ発生し，すでにホームノードが割り当てられているページへのアクセスでは発生しない．そのため，実行時間が十分長いアプリケーションならば割り当て処理のオーバヘッドは相対的に小さくなり，ホームノードを適切に割り当てられ

る first touch 方式は有用であると考えられる．

4.5 BT

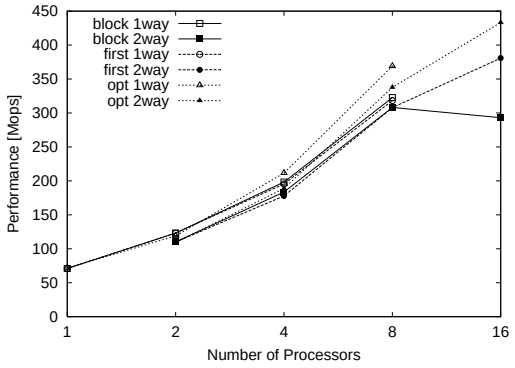

図 4 NPB BT の性能 (Myrinet)

BT の測定結果を**図 4** に示す．ホームノードをブロック分割で割り当てたものは，16 プロセッサで 8 プロセッサよりも性能が低下してしまっているが，first touch 方式では性能低下は起こらず，16 プロセッサではブロック分割方式の約 1.3 倍の性能が得られた．また最適な割り当て方の約 88%の性能が得られた．最適な方式では 1 プロセッサの約 6.1 倍，ブロック分割方式では約 4.3 倍，first touch 方式では約 4.5 倍の性能が得られた．

16 プロセッサで実行した時の各方式でのホームノード割り当て結果を比較したところ，最適な方式とブロック分割方式とでは，全ページの約 85%のページのホームノードが異なっていたが，first touch 方式では，最適な方式と異なっていたのは全体の約 14%であった．ブロック分割よりも適切にホームノード割り当てが行われていると言える．またページフォルトによって引き起こされたノード間のページ転送回数は，ノード ID が 1 のノードで，最適な方式で約 19,000 回，ブロック分割方式で約 30,000 回，first touch 方式で約 24,000 回だった．ホームノードの割り当ての違いがページフォルトの回数の差，そして転送されるページ数の差につながり，それが各方式の性能差の一因になっていると考えられる．また，最適な方式とその他 2 つの方式はバリア同期の回数が異なるため，それも性能に影響していると考えられる．

4.6 SP

SP の測定結果を**図 5** に示す．16 プロセッサで，first touch 方式はブロック分割方式より約 17%高い性能，最適な方式の約 93%の性能と，BT よりも最適な方式に近い性能が得られた．最適な方式では 1 プロセッサの約 2.3 倍，ブロック分割方式では約 1.9 倍，first touch 方式では約 2.2 倍の性能が得られた．

16 プロセッサで実行した時の各方式でのホームノード割り当て結果は，最適な方式とブロック分割方式とでは全ページの約 65%のページのホームノード割り当てが異なっていたが，first touch 方式では異なっていたのは全体の約 10 %だった．ノード間ページ転送回数は，最適な方式で約 110,000 回，ブロック分割方式で約 138,000 回，first touch 方式で約 111,000 回であった．first touch 方式と最適な方式との差は約 1000 回と小さく，それが最適な方式の 93%という高い性能が得られた要因であると考えられる．

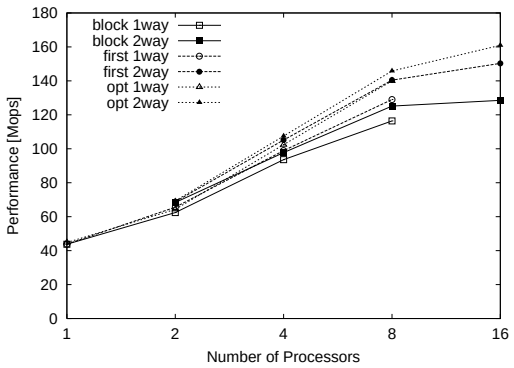

図 5 NPB SP の性能 (Myrinet)

5. 関 連 研 究

原田らは，SCASH にホームノードの動的再配置機構を実装した[7]．各ページへの書き込み量に着目し，多く書き込みを行ったノードにホームを移動させることでリモートアクセスによるオーバヘッドを削減した．SPLASH2 の LU を用いて 64 プロセッサで性能評価を行った結果，ホームノードをラウンドロビンで割り当てた場合の 1.17 倍の性能が得られたが，ホームを最適化した場合と比較すると 0.722 倍に留まった．

栄らは，Omni/SCASH に動的に負荷分散を行う機能を実装した[8]．ループの初期数回の実行時の性能を測定し，それに基づいて以降のループ範囲を再分割することで動的負荷分散を図った．NPB CG で性能評価を行った結果，静的なスケジューリングよりも低い性能しか得られなかった．それはホームノードを動的に移動させる機能が実装されておらず，データのローカリティが損なわれてしまうためである．そのためページフォルト数の計測によるページマイグレーションの機能の実装が行われている．

first touch 方式で適切にホームノードが割り当てられるようにするためには，多くの場合何らかの first touch 用の初期化ループをプログラムに加える必要がある．与えるデータによってアクセスパターンが変化するアプリケーションの場合，データの初期化時にホームノード割り当てが行われる first touch 方式では適切に対応できず，良い性能を得るためには一時的に使用するデータ領域を設けるなど，プログラムを変更する必要があり，複雑な指示なしで適切にホームノード割り当てが行えるという first touch 方式の利点が

失われてしまう．そこでプログラムを変更せずに良い性能を得るために，廣岡らはデータ分散方式をコンパイラで制御する方式を提案している[9]．また Bircsak らは，任意の時点以降の初めての touch で割り当てを行う，next touch[10] 方式を提案している．

6. 結　　論

本稿では，ソフトウェア分散共有メモリ OpenMP Omni/SCASH に First Touch page allocation の機能を実装し，評価を行った．ホームノード割り当て処理のオーバヘッドを測定した結果，処理には管理ノードとの通信を必要とするため，Ethernet でのオーバヘッドは Myrinet でのものの約 5.4 倍と，ネットワーク性能を反映したものであることが分かった．

NPB の BT 及び CG で性能評価を行った結果，16 プロセッサでの実行で BT ではホームノードをブロック分割で割り当てた場合の約 1.3 倍，最適に割り当てた場合の 88 %の性能が得られた．また SP ではブロック分割方式の 1.17 倍，最適な方式の 93 %の性能が得られた．16 プロセッサ実行時のホームノード割り当て結果を調べたところ，最適な方式とブロック分割方式では BT では全ページの 85%，SP では 65%のページのホームノードが異なっていたが，first touch 方式では BT では 14 %，SP では 10%にまで抑えられた．ブロック分割方式よりも適切にホームノード割り当てが行えていると言える．それによりページ転送回数も削減できており，性能向上につながった．

しかし，first touch ではデータへ初めてアクセスするときのアクセスパターンによってホームノードが割り当てられるため，それが計算時のアクセスパターンと異なっているとデータのローカリティが低下し，性能が低下してしまう可能性がある．そのようなアプリケーションでは，計算時のアクセスパターンに合わせてデータにアクセスする処理を，それまで初めてアクセスしていた時点よりも前に挿入する必要がある．また，与えるデータによってアクセスパターンが変化するアプリケーションの場合は，データの初期化前にアクセスパターンを知ることができないため，first touch 方式で適切に割り当てられるようにするためには，一時的に使用するデータ領域を設け，その中で初期化し，計算時にはそこから実際のデータ領域へコピーするなどの処理を挿入する必要があり，プログラミングのコストが高くなる．そこで，そういった処理を加えなくてもホームノードが適切に割り当てられるようにするため，next touch 方式を実装する必要があると考えている．

また，first touch 方式は一度ホームノード割り当てを行ったらそれ以降は変更をしないため，動的にアクセスパターンが変わるアプリケーションでは良い性能が得られない可能性がある．そこでそういったアプリケーションのために，first touch 方式と動的にホームノード割り当てを変更する手法を組み合わせる必要性があるかどうかを検討することも今後の課題である．

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) Amza, C., Cox, A., Dwarkadas, S., Keleher, P., Lu, H., Rajamony, R., Yu, W. and Zwaenepoel, W.: Treadmarks: Shared Memory Computing on Networks of Workstations, *IEEE Computer*, Vol. 29, pp. 18–28 (1996).
2) 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).
3) PC クラスタコンソーシアム: http://www.pccluster.org.
4) Gharachorloo, K., Lenoski, D., Laudon, J., Gibbons, P., Gupta, A. and Hennessy, J.: Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors, *Proc. 17th Annual International Symposium on Computer Architecture (ISCA 1990)*, pp. 15–26 (1990).
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) 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 1999-HPC-77, pp. 89–94 (1999).
8) Sakae, Y., Matsuoka, S., Sato, M. and Harada, H.: Preliminary Evaluation of Dynamic Load Balancing Using Loop Re-partitioning on Omni/SCASH, *Proc. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, pp. 463–470 (2003).
9) 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式, 情報処理学会論文誌, Vol.41, No.5, pp.1430–1438 (2000).
10) Bircsak, J., Craig, P., Crowell, R. Cvetanovic, Z., Harris, J., Nelson, C. and Offner, C.: Extending OpenMP For NUMA Machines, *Proc. Supercomputing'2000* (2000).