

Omni/SCASH における性能不均質なクラスタ向け 動的負荷分散機能の実装と評価

栄 純明^{†1} 松岡 聡^{†2}
佐藤 三久^{†3} 原田 浩^{†4}

HPC の分野に限らず様々な分野で PC/WS を構成単位とするコモディティクラスタが重要なプラットフォームになっている。コモディティクラスタ環境では、プロセッサ技術の急速な進歩、ユーザーニーズ、予算上の都合など様々な理由によりノード間に性能の不均一性が生じるケースが増加している。これはロードインバランスの原因のひとつであり、何らかの動的負荷分散機能が必要とされている。これまでに動的負荷分散を実現する実行時性能に基づくループ再分割機能を Omni/SCASH に実装したが、データアクセス範囲の変化を伴うため、データのローカリティが性能に大きな影響を持つようなアプリケーションでは性能低下がみられるケースがあった。本論文では新たに実装したページ参照数に基づくページマイグレーション機能とループ再分割機能と組み合わせた場合の性能に関して報告する。評価の結果、単体で Laplace を 4 ノードで実行した際に 60% 程度の性能向上がえられ、ループ再分割と組み合わせることでデータアクセス範囲の変化による性能低下を改善できることを示した。

Implementation and Evaluation of Dynamic Load Balancing for Performance Heterogeneous Clusters on Omni/SCASH

YOSHIAKI SAKAE,^{†1} SATOSHI MATSUOKA,^{†2} MITSUHIISA SATO^{†3}
and HIROSHI HARADA^{†4}

Increasingly large-scale clusters of PC/WS continue to become majority platforms in HPC field. In such a commodity cluster environment, there may be incremental upgrade due to several reasons, such as rapid progress in processor technologies, or user needs and it may cause the performance heterogeneity between nodes from which the application programmer will suffer as load imbalances. To overcome these problems, some dynamic load balancing mechanisms are needed. We have implemented and reported on loop re-partitioning mechanisms based on the runtime performance so far. However, loop re-partitioning involves changes of data access ranges so that some applications whose performance rather depends on data locality shows performance degradation. In this paper, we report our recent work on page migration mechanisms based on page reference counting and its performance. Results show that we can achieve about 60% performance gain with Laplace on 4 nodes cluster by page migration and restore the performance that degraded by loop re-partitioning.

1. はじめに

Fast Ethernet や Gigabit Ethernet もしくはより高速な Myrinet¹⁾, InfiniBand などのネットワークで密結合されたクラスタ型並列計算機が HPC の分野

で主流になりつつある。中でも汎用部品を構成要素とするコモディティクラスタが、コストパフォーマンスや管理の容易性などから多くの研究機関や大学などで使用されている。しかしながら、プロセッサやネットワーク技術の急速な進歩のため、段階的なノードの追加や、プロセッサやメモリなどの増強、増設がおこなわれ、しばしば**性能の不均一性**を生じている。さらにマルチユーザー環境においてはノード間性能に不均一性がなくても結果的に性能の不均一性が問題になるケースもある。

環境やアプリケーションごとに、プログラマが明示的にロードバランシングを行うことは難しく、ランタイムシステムによる自動的なロードバランシングが必

^{†1} 東京工業大学大学院情報理工学研究科 数理・計算科学専攻
Tokyo Institute of Technology

^{†2} 東京工業大学大学院学術国際情報センター / NII
Tokyo Institute of Technology / NII

^{†3} 筑波大学 電子・情報工学系 計算物理学計算センター
Tsukuba University

^{†4} ヒューレットパッカードジャパン
Hewlett-Packard Japan, Ltd.

要とされている。この問題を解決するため、われわれはコモディティクラスタ向けに OpenMP²⁾ をプログラミングインターフェースとし自動的にデータ再配置と負荷分散を行うシステムを開発している。

具体的には、われわれはコモディティクラスタをターゲットに Software Distributed Shared Memory, SCASH³⁾ 上の OpenMP の実装である Omni/SCASH⁴⁾ を開発している。また、これまでに性能不均一な環境でロードインバランスの改善を行う実行時性能モニタリングに基づくループ再分割機能の実装と、その性能を報告した⁵⁾。

本稿ではループ再分割機能の性能評価結果、データのローカルリティを改善するページ参照数に基づくページマイグレーション機能の実装とその性能評価に関して報告する。さらに、ループ再分割機能とページマイグレーション機能を組み合わせて利用した際の性能に関しても報告する。これらの機能を用いることで、アプリケーションプログラマが明示的にデータとタスクの配置を指定することなく、ランタイムシステムによって自動的にロードバランシングが行えることを期待している。

なお、本稿ではスペースの都合上 Omni/SCASH の概要、ループ再分割機能の実装に関しては割愛させていただく。詳細に関しては参考文献を参照されたい。

2. 動的負荷分散

ロードインバランスの原因はいくつか考えられる：(1) ターゲットアプリケーションが本質的にロードインバランスである場合、(2) マルチユーザ環境の影響でロードがノード間で異なる場合、(3) アプリケーションが性能不均一なクラスタ上で実行される場合。(2)と(3)のケースのロードインバランスは実行環境やタイミングごとに異なるため、静的なロードバランシング手法では不十分であり、実行時性能計測に基づいた動的なロードバランシング手法が不可欠である。

これまでに OpenMP の並列ループのスケジューリングポリシーとして、実行時性能モニタリングに基づいた動的ループ再分割手法を Omni/SCASH に実装し (profiled スケジューリング)、その性能を報告した⁵⁾。profiled スケジューリングではループの再分割によりデータのアクセス範囲が変わり、当初予想した通りアプリケーションによっては、性能予測を誤りページフォールトおよびバリア同期の増加のため性能低下がみられた (図 1^{*})。

profiled スケジューリングによるデータのローカルリティ低下を軽減するため、われわれは次節以降で述べるページ参照数に基づくページマイグレーション機能を SCASH に実装している。

Execution Time of OpenMP version of Laplace 2048x2048 with Static, Dynamic and Profiled Scheduling on Homogeneous Settings

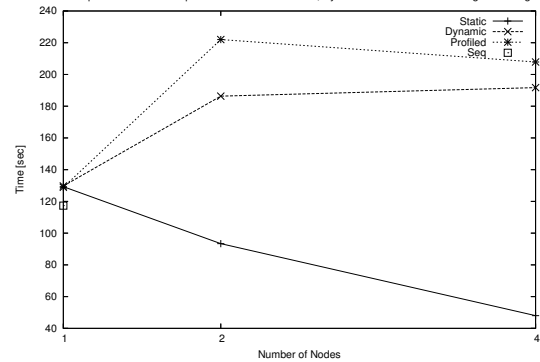


図 1 性能不均一な環境 (1 台だけ Celeron ノード使用) での OpenMP 版 Laplace 2048×2048 の実行時間

2.1 ページ参照数に基づくページマイグレーション

一般的に、コモディティクラスタは専用のネットワークを備えた MPP などに比べてネットワークレイテンシが大きい。そのため大きなデータを数回送受信するよりも、SDSM の様に比較的小さいデータを多数送受信するようなパターンの通信の方が大きな性能低下につながる傾向にある。したがって SDSM 環境ではデータ配置を適切に行うことが性能の観点から非常に重要であり、いくつかの手法が提案されている。たとえば、(1) 実行環境がメモリ配置手法としてファーストタッチ制御をサポートしている場合、ファーストタッチを利用してメインループの直前に初期化ループを挿入する手法、(2) データとスレッド間の affinity 情報を annotate する手法⁶⁾、(3) ディレクティブやプログラミング言語の機能を用いてアプリケーションプログラマが明示的にデータ配置を記述する手法、などがある。しかしながら、これらはいずれも静的な手法であり、dynamic スケジューリングや profiled スケジューリングなどの動的スケジューリングなどに起因する実行時のデータアクセスパターンの変化に対応できなかったり、実行環境ごとにデータ配置の調整をする必要があったりする。

本研究では、リモートメモリアccessを削減し全体性能を改善するため、ページアクセス回数に基づくページマイグレーション技法を利用する。

2.1.1 SCASH のページ管理方法

SCASH は OS の提供するページ保護機構 (mprotect(2)) を用いて仮想的に共有メモリを実現するページベースの SDSM システムである。各ノードで共有メモリのページテーブルを持ち、各ページの保護状態 (e.g. Read-Write, Read-Only and Un-mapped) や管理ノード、つまり base と home などが保持されている。

Base

ページの最新の home を常にトラックしているノード。すべてのノードは共有メモリ空間のすべてのペー

^{*} 評価には第 3.1 節に示した環境を使用。

ジの base を記録している。ページマイグレーションなどで home を見失った場合には base に問い合わせることで最新の home を得ることが可能。各ページの base は プログラムの実行を通して変化しない。

Home

ページの最新データとそのページを共有しているノードの集合を管理しているノード。あるページを共有するノードは home からそのページの最新データをリモートコピーしてくる。メモリバリア同期の際にはページデータを更新したノードは、更新前のページデータとの差分 *diff* を作成し、home にそれを送信する。home は受信した *diff* をページデータに適用することによりページデータを最新の状態にする。

SCASH では共有メモリの初期化直後は base と home を同一の物理ノードに割り当てる。その際負荷分散のため、ラウンドロビンで各ノードに割り当てを行う。

2.1.2 近似ページ参照数情報

最適なページマイグレーションを達成するためには、対象区間におけるメモリアクセスパターンを完全に把握する必要がある。しかしながら、コモディティクラスタでは一般にメモリ参照数を数えるためのハードウェアサポートがないため、効率的かつ正確にページ参照数を数えることは困難である。代わりに、われわれは SCASH のページ管理機能を用いページフォールトの回数を数えることでページ参照数を近似する。

ページベースの SDSM である SCASH では、ローカルに保持していないページデータへのアクセスがあった際にページフォールトが発生し、フォールトハンドラがページの home からデータをリモートコピーする。また、SCASH では dirty page 情報を管理するためにメモリバリア後に home においてもページデータの保護情報を RO (Read-Only) に変更する。その結果、メモリバリア後の最初のローカルアクセスでもライトページフォールトが発生する。本実装では、ページフォールトハンドラで各ページごとに、アクセス元のノード別にリード、ライトを区別して数え、ページ参照数の近似値として利用する。各ノードで数えた参照数は後述する *Flush Diff* メッセージに含めて home に送信される。

ページ参照数の近似値に基づくマイグレーション先決定は以下の理由から妥当性があり効率的であると期待できる: (1)SPMD プログラムでは一般的に短時間のうちに特定のメモリ領域が多数のノードから共有アクセスされるよりも、アクセスするノードは少数であるように書かれることが一般的である。(2)SCASH を含む一般的なページベースの SDSM システムではリモートページコピーは最初のページアクセス時のみに起こり、その後のリード、ライトアクセスはページのローカルコピーに対して行われる。そのため、ページ

アクセスよりもページのリモートコピーを伴うページフォールトの回数に基づいてマイグレーション先を決定した方が、性能向上が大きいと期待できる。

2.1.3 ページマイグレーションの方法

バリア区間においてあるページに対する最初のアクセスが起った際に、ページフォールトハンドラがページデータを home からリモートコピーを行う。その後、バリア区間の最後、バリア同期の際に、ページ *diff* が作成され home に送信される。したがって、ここまでページマイグレーションとして述べてきたものは SCASH では home の再配置と考えることができる。われわれは、バリア同期とあわせて home の再配置を行う関数 `scash_barrier_migrate_home()` を新たに実装した。

Home の再配置は以下のように行われる。ここで、home の再配置が行われる区間を “migrate region” と呼ぶことにする。

(1) Barrier Sync

すべてのノードでバリア同期を行い共有メモリへのアクセスを停止する。

(2) Flush Diff

各ノードで migrate region 内で変更を行ったページに対して *diff* を作成し home に送信、ページの保護状態を RW (Read-Write) から RO (Read-Only) に変更。

(3) Barrier Sync

(4) Home 再配置

diff を受け取った home は、*diff* とともに送信されてきた各ノードのページ参照数から新しい home を計算する。新しい home の計算方法に関しては次節で説明する。home 再配置が行われると、ページデータが新しい home にコピーされるとともに、新しい home がベースノードに通知される。また、ページを共有しているノードすべてに invalidation メッセージが送信され、ページアクセスカウンタもクリアされる。この invalidation メッセージには新しい home の情報も含まれる。したがって、ページを共有しているすべてのノードは旧 home からの invalidation メッセージを受け取ることで常に新 home を知ることができる。

(5) Barrier Sync

最後にすべてのノードでバリア同期をもう一度行い `scash_barrier_migrate_home()` を終了。

2.1.4 新しい home の決定

新しい home を決定する際の理想は、次回以降の migrate region でページアクセス数が最大となるノードを新しい home として決定することであるが、予備知識なしでメモリアクセスパターンを完全に予測するのは困難である。そこで、多くの SPMD プログラムは同じ migrate region でメモリアクセスパターンが大きく変化しないことを想定する。

この想定に基づき、近似ページ参照数をローカリティを改善するための指標として用いる。SCASH では、migrate region で home 以外からの書き込みがあった場合のみ、diff を home へ送信する必要が生じ、コモディティクラスタではこれがもっともコストのかかる操作となるため、リードよりもライトにより強い重み付けをする。現在の実装では1ライトは2リードに相当するよう設定してある。

また、現実装では home によるライトが起った際には home 再配置を行わない設定になっている。これは、home では scash_barrier() や scash_barrier_migrate_home() で diff の作成が必要でないためであり、さらには SCASH のフォールトハンドラではローカルリードアクセスは数えることができないためである。これらの事実に加えて、home によるライトが起った際には、後続の migrate region でも home によるライトが起る可能性が高いという仮定に基づき、ページコピーと diff 作成のオーバーヘッドを避けるために、home 再配置を抑制することで、オーバーヘッドを避けている。

3. 評価

まずはじめに、ページマイグレーション機能単体の性能を測定し、つぎに profiled スケジューリングとページマイグレーションを組み合わせて使用した際の性能を測定する。

ページマイグレーション機能単体の性能評価には SCASH 版の Laplace を使用した。profiled スケジューリングと組み合わせて使用した際の性能評価には OpenMP 版の Laplace を使用した^{*}。

実験で使用した Laplace の配列サイズは 1024×1024 と 2048×2048 でカーネルループは 100 回繰り返される。各プロセッサからの配列データへのアクセスは行方向にブロック分割であるが、(Omni)/SCASH の base/home ノードの割り当てはページ単位でサイクリックに行われるため最適ではない。したがってデータ配置を最適化することによる性能向上の余地がある。

3.1 評価環境

表 1 に評価に使用した計算機環境を示す。評価に使用した性能不均一なクラスタは CPU 性能のみ不均一な設定となっており Pentium III 500MHz (512K Cache, 100Mhz bus speed) のノードと Celeron 300MHz (128K Cache, 66Mhz bus speed) のノードから構成される。多くの数値計算プログラムにおいて Pentium III ノードは Celeron ノードのおよそ倍の性能を示す。

OS には RedHat 9.0 を、クラスタシステムソフトウェアには SCore-5.6.1 を、コンパイラには gcc-3.2.2 -O2 をそれぞれ用いている。

^{*} 使用した Laplace のオリジナルは筑波大の朴助教授によって書かれたものである。

表 1 評価環境: Performance Heterogeneous Cluster

	Fast nodes	Slow node
CPU	Pentium III 500MHz	Celeron 300MHz
Cache	512KB	128KB
FSB	100Mhz	66Mhz
Chipset	Intel 440BX	Intel 440BX
Memory	SDRAM 512MB	SDRAM 512MB
Myrinet	M2M-PCI32C	M2M-PCI32C

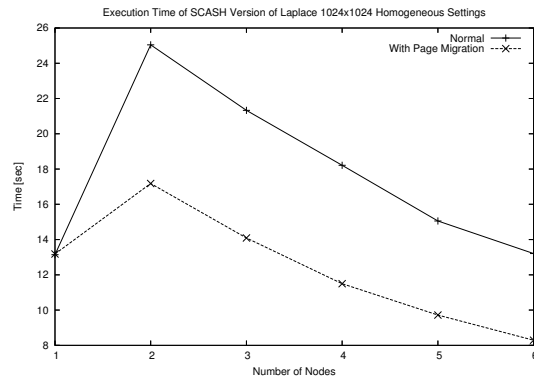


図 2 性能均一な環境 (Pentium III ノードのみ使用) での SCASH 版 Laplace 1024×1024 の実行時間

3.2 ページマイグレーションの性能

ページマイグレーション機能の性能測定を Laplace 1024×1024 および 2048×2048 を用いて行った。本論文では 1024×1024 の結果のみ記す。

実験では、カーネルループの最初のイテレーションの直後に、カーネルループでのページ参照数に基づき一度だけページマイグレーションを行っている。図 2 に示したように、ページマイグレーションによって、通常実行した場合に比べて性能の向上が得られた。表 2 にマイグレートしたページ数を示す。

配列サイズ 1024×1024 の場合、行列の各行は Pentium III / Celeron の物理ページ 2 ページに一致する 8KB を占めている (1024×sizeof(double))。ここで 2 ノードで実行することを考える。SCASH による base, home の初期配置後は図 3 のようになっており、全ページのうち半数が不適切な割り当てとなっている。つまり、ノードあたり 1024 ページのうち 512 ページが不適切な割り当てとなっており、これらがマイグレーション候補となる。実際に、われわれの実装では全体で 1022 ページ (ノードあたりでは 511 ページ) がマイグレートされており、ほぼ最適なページマイグレーションが行えたと言える。ただし、4 ノード実行時にはマイグレートすべきページ数は 1792 ページのところ、実際には 1534 ページしかマイグレートされなかった。

このように近似ページ参照数に基づくページマイグレーションが最適でない原因は以下にあると考えている。(1) home write が起った際に常にマイグレー

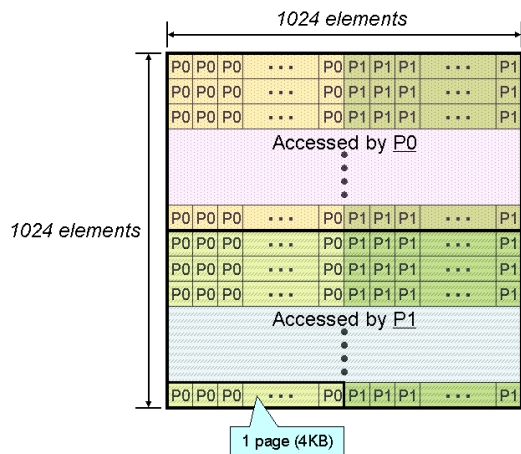


図 3 共有メモリ初期化直後の home の配置とメモリアクセスパターン

表 2 性能均一な環境 (Pentium III ノードのみ使用) での SCASH 版 Laplace 1024×1024 を実行した際にマイグレートしたページ数

Nodes	Migrated Pages
2	1022
3	1364
4	1534
5	1630
6	1650

ションを抑制している, (2) 行列がブロック分割されているため, いくつかの境界ノードでページの近似参照数が複数のマイグレーション先ノードの候補で等しくなってしまうケースがあり得るが, そういったケースで全体のバランスをとるようなページマイグレーションを行っていない。

4. ループ再分割とページマイグレーションを組み合わせた場合の性能

性能不均一な設定で profiled スケジューリングとページマイグレーションを組み合わせて Laplace を実行したときの性能を示す。ページマイグレーションを行うことによって profiled スケジューリングでデータのローカリティが低下したことによる性能低下が改善され, static スケジューリングと同等かそれ以上の性能を達成することを期待している。static スケジューリングに対して性能が少なくとも同等であるということは, profiled スケジューリングを利用しやすくするという点においても重要である。

Omni によって生成された C の中間コードに図 4 に示したようにページマイグレーションコード挿入した。評価では profiled スケジューリングのオプション “eval_skip” を 1 に設定した。まずはじめにページマ

```
#pragma omp parallel private(k,x,y)
{
  for(k = 0; k < NITER; k++) {
    /* old < new */
    #pragma omp for schedule(profiled, 0, 1, 1)
    for(x = 1; x <= XSIZE; x++)
      for(y = 1; y <= YSIZE; y++)
        u[x][y] = u[x][y];

    /* update */
    #pragma omp for schedule(profiled, 0, 1, 1)
    for(x = 1; x <= XSIZE; x++)
      for(y = 1; y <= YSIZE; y++)
        u[x][y] = (u[x-1][y] + u[x+1][y] + u[x][y-1] + u[x][y+1]) / 4.0;
  }

  if (CONDITION)
    scash_barrier_migrate_home();
}
```

図 4 ページマイグレーションコードの挿入箇所

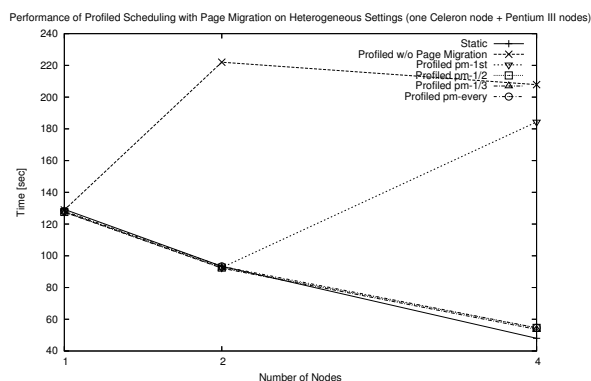


図 5 性能不均一な環境で profiled スケジューリングとページマイグレーションを組み合わせた場合の性能

イグレーションによって最適に近いデータ配置にしてから, profiled スケジューリングを行うようにするためである。図 4 中の “CONDITION” で示した条件を変更することによって 4 種の条件でページマイグレーションを行った: (1) 最初のイテレーション後に一回のみ (pm-1st), (2) 二回に一回 (pm-1/2), (3) 三回に一回 (pm-1/3), (4) 毎回 (pm-every)。

これらの条件は最適なタイミング, 頻度でのページマイグレーションではないが, 図 5 に示したようにその効果を確認した。pm-1/2, pm-1/3, pm-every では profiled スケジューリングとページマイグレーションがともに安定な状態なり過度な page faults が抑制されたために, 明らかな性能の改善が見られた。

5. 関連研究

Nikolopoulos らは SGI Origin 2000 上で Origin の提供するハードウェアページ参照カウンタを用いて OpenMP プログラムの並列ループ区間における完全なページ参照数に基づくユーザレベルのページマイグレーションによるデータ配置を実現している⁷⁾。これによって適切な区間における正確なページ参照数に基づき, 適切なタイミングでページマイグレーションを

行うことが可能となっており、OS の提供するページマイグレーション機能よりも高い性能を NPB のいくつかのプログラムで得られている。われわれの提案は彼らの手法を、ハードウェアサポートのないコモディティクラスタ環境に拡張するものである。

原田らは SCASH に、各プロセッサにおけるページ変更量に基づくホームマイグレーション機能を実装している⁸⁾。バリア同期ポイントごとに各ページの変更量の多いノードを特定し、ページの home をそのノードに変更することによって、リモートアクセスによるオーバーヘッドを削減している。SPLASH2 の LU を用いた評価の結果、8 ノードまででは home を最適に割り当てたケースよりもこの方法によって高い性能を達成している。なお、原田らの手法はバリア区間でのマイグレーションを対象としており、本稿で示した手法は複数のバリア区間を含む“migrate region”を想定している点で異なり、競合しない。

6. おわりに

本論文では、われわれが Software Distributed Shared Memory, SCASH 上の OpenMP の実装である Omni/SCASH に対して現在行っている動的負荷分散拡張に関して報告した。われわれの目標は、ノード間性能の不均一性やマルチユーザ環境などに起因するアプリケーションのロードインバランスを半自動的に解決する手法の実現である。このような問題に対して静的なアプローチは不十分であり、われわれはターゲットループの実行時性能セルフプロファイリングに基づくループ再分割と、ターゲットループにおける近似ページ参照数に基づくページマイグレーションを用いた動的負荷分散技法を提案した。これらの手法を用いることにより、ユーザープログラマが明示的にデータとタスクの配置を記述することなく、ラインタイムシステムによって最適なロードバランシングが行える。

提案したページ参照数に基づくページマイグレーションによって Laplace を 4 ノードで実行した際におよそ 60% 程度の性能改善が見られ、profiled スケジューリングと組み合わせることで、profiled スケジューリングによるデータのローカリティの低下を改善できることを確認した。今後、profiled スケジューリングとページマイグレーションのより効率的に組み合わせることとシステムのチューニングを行うことによって、性能不均一な環境において static スケジューリングよりも多くの場合に高性能が得られるようになると期待している。

今後の課題としては以下のようなことを考えている。

- ページマイグレーション先の決定において、現在は home write があった際に単にマイグレーションをしない決定をしているが、適切な重み付けを行いマイグレーション先の決定に利用する。

- より正確に性能測定、ノード間性能比の予測を行う。
- Omni によるページマイグレーションコードの自動挿入するようにする。
- profiled スケジューリングとページマイグレーションの協調動作をより効率的で自動的なものにし様々な状況に対応できるようにする。

謝辞 本研究は、科学技術振興機構・戦略的創造研究「低消費電力化とモデリング技術によるメガスケールコンピューティング」および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) Myricom: . <http://www.myri.com/>.
- 2) OpenMP: . <http://www.openmp.org/>.
- 3) Harada, H., Tezuka, H., Hori, A., Sumimoto, S., Takahashi, T. and Ishikawa, Y.: SCASH: Software DSM using High Performance Network on Commodity Hardware and Software, *Proceedings of Eighth Workshop on Scalable Shared-memory Multiprocessors*, ACM, pp. 26–27 (1999).
- 4) Sato, M., Harada, H. and Ishikawa, Y.: OpenMP compiler for Software Distributed Shared Memory System SCASH, *Proceedings of Workshop on OpenMP Applications and Tool (WOMPAT'2000)* (2000). San Diego, USA.
- 5) Sakae, Y., Matsuoka, S., Sato, M. and Harada, H.: Preliminary Evaluation of Dynamic Load Balancing Using Loop Repartitioning on Omni/SCASH, *Proceedings of the Third IEEE/ACM International Symposium on Cluster Computing and the Grid / DSM (DSM2003: Distributed Shared Memory on Clusters workshop @ CCGrid)*, IEEE/ACM, pp. 463–470 (2003). Tokyo, Japan.
- 6) 長谷川篤史, 佐藤三久, 石川裕, 原田浩: ソフトウェ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) Nikolopoulos, D. S., Papatheodorou, T. S., Polychronopoulos, C. D., Labarta, J. and Ayguadé, E.: Is Data Distribution Necessary in OpenMP?, *Proc. of Supercomputing 2000* (2000). Dallas, TX.
- 8) Harada, H., Ishikawa, Y., Hori, A., Tezuka, H., Sumimoto, S. and Takahashi, T.: Dynamic Home Node Reallocation on Software Distributed Shared Memory System, *Proceedings of IEEE 4th HPC ASIA 2000*, pp. 158–163 (2000).