ソフトウェア分散共有メモリ上の OpenMP Omni/SCASH の SPLASH2 による性能評価

栄純明[†] 松岡聡[†] 佐藤三久[††]
長谷川篤史[†††] 原田浩[††]

共有メモリ向け並列化インタフェースである OpenMP を，分散メモリシステム上で実装するシステムの一つとして Omni/SCASH がある．Omni/SCASH により分散メモリシステム，共有メモリシステムの両方のアーキテクチャで同一のプログラムが透過的に動作可能となる．本稿では Omni/SCASH の性能評価を共有メモリ向けのベンチマークプログラムである SPLASH2 のうち FFT, Ocean, Water を用いて行った．SPLASH2 プログラムの OpenMP 化，Omni/SCASH に特有な変更，キャッシュヒット率，フォールトハンドラ・バリア同期実行頻度の解析結果などを報告する．さらに，性能ヘテロなクラスタにおける Omni/SCASH の評価も行った．

OpenMP compiler on Software Distributed Shared Memory System, Omni/SCASH Performance Evaluation with SPLASH2

Yoshiaki Sakae,[†] Satoshi Matsuoka,[†] Mitsuhisa Sato,[††]
Atsushi Hasegawa[†††] and Hiroshi Harada [††]

Omni/SCASH is a implementation of OpenMP on top of a DSM system SCASH, allowing portable execution of shared-memory OpenMP programs on SMPs as well as on clusters. To validate the effectiveness of Omni/SCASH, we conduct the following benchmarks: porting of selected sets of SPLASH2 benchmarks onto OpenMP and execution thereof on Omni/SCASH to measure the effectiveness of the implementation, such as the costs/frequencies of cache hit/cache miss/DSM fault handler/barrier invocations. We then test the effectiveness of whether Omni/SCASH serves as a effective programming platform for heterogeneous clusters, using some of its load balancing primitives to balance the load among processors of differing performance. Preliminary results are mixed, and indicate that further work is needed for portable parallel programming on (heterogeneous) clusters.

1. はじめに

使いなれた OS，ツールなどが利用できるという高いユーザビリティ，コストパフォーマンスの高さ，最新技術へのキャッチアップの容易性などを背景に，近年クラスタ型並列計算機が普及してきている．クラスタ型並列計算機は一般にコモディティハードウェアを構成要素とする特性上，本質的に多様性を持つ．この多様性にはノード間性能の差異も含まれ，さらに最近のハードウェア技術の動向をみると今後しばらくコモディティハードウェアにおける多様性が続き，性能的にヘテロ（不均質）なクラスタの増加が容易に予想される．

したがって性能の可搬性を有するポータブルなプログラムを作成するためのソフトウェア基盤が必要であり，そのソフトウェア基盤としてのプログラミング言語には以下のような要請がある．

- 処理系自身のポータビリティ
- 性能の可搬性
- 高い抽象度
- 記述の容易性

クラスタ上のプログラミング環境としては MPI に代表されるメッセージ通信ライブラリや，Split-C, Charm++, HPC++, MPC++ をはじめとする数多くの並列言語が研究・開発されている．前者は比較的低レベルな API のため並列化のコストが高く，一般に高性能を達成できると考えられているものの，その低レベルな API ゆえ性能可搬ではない．後者は，比較的抽象度が高く並列化のコストが安く，またある程度の性能の可搬性が期待できると考えられているもの

† 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
†† 技術研究組合 新情報処理開発機構
Real World Computing Partnership (RWCP)
††† 株式会社 NEC 情報システムズ
NEC Informatec Systems, Ltd.

の，依然としてデータ・タスク分割などをプログラマが明示的に行う必要があるなどの理由により，性能ヘテロなクラスタ環境において性能の可搬性を有するには至っていない．

本研究では性能可搬なプログラミングインタフェースとして OpenMP の SDSM 上の実装である Omni/SCASH をとらえ，そのベースラインとしての性能を SPLASH2 から FFT, Ocean, Water を用いて検証する．また，性能ヘテロなクラスタにおける性能の可搬性に関しても検証する．

2. Omni/SCASH

Omni/SCASH[1] は分散メモリシステム向けの OpenMP コンパイラであり，ソフトウェア分散共有メモリ（SDSM）SCASH[2] を利用し，OpenMP で記述されたプログラムをクラスタ型並列計算機上で実行可能な実行イメージにコンパイルする．

2.1 Omni OpenMP Compiler

Omni[3] は RWCP で開発している OpenMP の実装であり，Fortran および C 言語向けの仕様[4),5)] をサポートする．

Omni は入力として Fortran および C で記述された OpenMP プログラムを受取り，C で記述されたフロントエンド部で AST（Abstruct Syntax Tree）を基本とした中間表現（Xobject）に変換，最適化を行い，Java で記述されたコード生成部（Exc Java tool）で目的コードである C + pthread or solaris-thread（on Solaris）or sproc（on Irix）を出力するトランスレータとして実装されている．最終的な実行イメージはネイティブの C コンパイラが Omni の目的コードをコンパイル，実行時ライブラリとリンクすることによって得る．

2.2 Omni の SCASH 向け変更

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のに対して，SCASH では共有メモリ領域は実行時に専用のアロケータを用いて割り当てる．そのため Omni/SCASH では大域変数に対して以下のような処理を行う．

- すべ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．
- コンパイル単位毎に，そのコンパイル単位で宣言された大域変数を実行時に共有メモリ領域に割り当てるための初期化関数を生成する．
- 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配置されるセクション）に置かれ，main 関数が実行される前に起動される．

共有領域に割り当てられるオブジェクトのうち，一定サイズ以上の配列に関しては，配列をブロック分割し，各ブロックのホームノードをノード番号順に割り当てる．

2.3 Omni/SCASH が独自に提供する機能

SDSM 環境では共有メモリのローカリティが性能に大きな影響をおよぼす．そのため Omni/SCASH では（1）配列の任意の次元でデータをブロック/サイクリック分割し，ノード番号順にホームノードを割り当てることを指示する mapping 指示文，(2）mapping 指示文によって分割された配列に合わせてループのイタレーション分割を行う affinity スケジューリング機能，(3）実行時に共有メモリ領域を，指定したホームノードの割り当てにしたがって確保する専用のアロケータ，を提供する．

本研究ではこれらの機能のクラスタにおける実際の有用性，および性能ヘテロなクラスタにおける有効性を検証する．

3. SPLASH2 の OpenMP 化

SPLASH2[6] のうち，FFT（高速フーリエ変換），Ocean-contiguous（海洋の移動問題の解法），Water-spatial（水分子の力とポテンシャルの計算（$\mathcal{O}(n)$ のアルゴリズムを使用））を OpenMP に移植し，Omni/SCASH の評価に用いた．

SPLASH2 に含まれるプログラムはそれぞれスケジューリングにスタティックスケジューリング，task queue，self-scheduled loop などを用いて最適に近いと思われる方法で並列化されている．スレッドの生成についてもオーバーヘッドを避けるため，プログラムの最初に一回だけ作成し最後に join する，深さ 1 段の fork-join スタイルをとる．

実験では最適に近い並列プログラムにおける Omni/SCASH の性能を計測するという観点から，各マクロを対応する OpenMP の命令で置き換えた．一般的なスレッドライブラリと OpenMP とで異なる点で主なのは，一般的なスレッドライブラリではスレッドの生成はプログラマが陽に行う点である．この点に関してはマクロによる置き換えはシンタックス上難しいので，各プログラムに`#pragma omp parallel` を入れることで対応した．したがってプログラムは計測範囲全体で 1 つのパラレルリージョンとなっており，その他 lock 関係，barrier，omp_set_num_threads(), threadprivate などの OpenMP 命令を使用している．

また，SPLASH2 の想定する共有メモリと Omni/SCASH における SDSM である SCASH では共有されるメモリ領域と，メモリのコンシステンシの保証されるタイミングが若干異なる;

- 共有メモリの確保に関しては，Omni/SCASH で独自に提供する共有メモリ領域を明示的に確保するための命令を使用した．

```
1  volatile int a = 0;
2  #pragma omp parallel
3  {
4      if (omp_get_thread_num() == 1) {
5          a = 1;
6          #pragma omp flush (a)
7      }
8      while (a == 0) {
9          /* buisy wait */
10         #pragma omp flush (a)
11     }
12 }
```

図 1 共有メモリ型並列計算機と，SDSM においてコンシステンシをとるタイミングの違いが問題となる例

- メモリのコンシステンシのタイミングの問題では，OpenMP は比較的 lazy なメモリモデルをとると言うことがあげられる．たとえば，パラレルリージョンにおけるメモリのコンシステンシはバリア同期命令，flush 命令，critical リージョンの前後，またワークシェアリング構文やパラレルリージョンの最後などでとられることになっており，その他の場所ではコンシステンシの保証は必要ない．そのため，SDSM のメモリモデルとマッチさせやすいのだが，SPLASH2 では pthread などに見れるような，よりシーケンシャルなコンシステンシを前提に記述されているため，共有データに関して複数のスレッドからの読み書きが競合するような場合，適宜 flush を行いその変更を他のスレッドに反映する必要がある．たとえば，**図 1** のような volatile 変数に対する buisy wait などが該当し，Omni/SCASH では 6 行目と 10 行目の flush が新たに必要となる．

4. 性能評価

まず性能ホモ（均質）なクラスタ環境における性能評価を行う．

4.1 実行環境

評価には我々の研究室で所有する *Presto* クラスタを利用した．Presto はシングルプロセッサをノードとする 64 ノード構成のクラスタであるが，本実験では Myrinet を備えた前半 32 ノードのうち 16 ノードのみを使用する．**表 1** に各ノード性能を示す．並列実行環境としては RWCP で開発されている SCore-3[7] の開発バージョンを，コンパイラには pgcc-2.95.3[8] を用いコンパイルオプションは -O6 -mcpu=i686 -march=i686 -malign-double -fstrength-reduce-funroll-loops -fexpensive-optimizations とした．

4.2 実行方法

評価には 3 節で説明した FFT, Ocean, Water を用いた．各ベンチマークで計測した問題サイズを以下に示す．

表 1 Presto クラスタの仕様

CPU	Celeron 500MHz
Cache	128KB
Chipset	440BX
Memory	SDRAM 512MB
NIC	Intel E.E. Pro 10/100
NIC	Myrinet M2M-PCI64A-2
OS	Linux-2.2.17

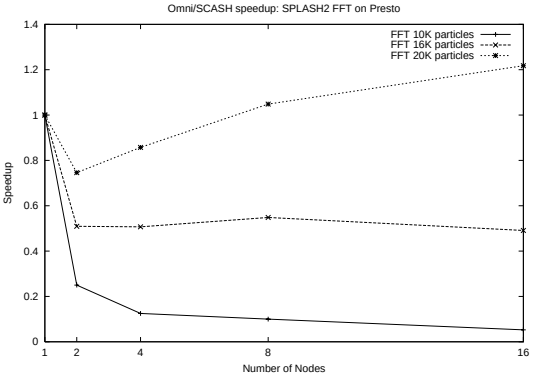

図 2 Presto における FFT のスピードアップ

FFT 2^{10}K, 2^{16}K, 2^{20}K points
Ocean 258^2, 514^2 grid ocean
Water 512, 8000, 27000 molecules

性能解析ではキャッシュヒット率と，フォールトハンドラおよびバリア同期の実行数をカウントした．L1, L2 ヒット率は Intel P6 アーキテクチャに備わるパフォーマンスイベント DCU_LINES_IN, DATA_MEM_REFS および L2_LINES_IN, DCU_LINES_IN の各イベントの発生回数を計測し以下のように計算した[9]．

$$\text{L1 cache hit ratio} = 1 - \frac{\text{DCU\_LINES\_IN}}{\text{DATA\_MEM\_REFS}}$$

$$\text{L2 cache hit ratio} = 1 - \frac{\text{L2\_LINES\_IN}}{\text{DCU\_LINES\_IN}}$$

フォールトハンドラおよびバリア同期の実行数は SCASH にプロファイリングコードを挿入することで計測した．

4.3 実行結果

表 2 に各プログラムの実行に要した時間，**図 2**，**図 3**，**図 4** に各プログラムのスピードアップのグラフを示す．また，**表 3** に FFT（2^{20}K points），Ocean（258^2 ocean），Water（27000 molecules）のキャッシュヒット率，フォールトハンドラ・バリア同期の起動頻度を示す．

FFT 2^{10}K, 2^{16}K では表 2 にあるようにそもそも実行時間自体が短く，分散メモリ環境での実行はふさわしくないとも言える．問題サイズ 2^{20}K でようやくス

表 2 Presto クラスタにおける SPLASH2 FFT, Ocean, Water および NPB CG の実行時間 [msec]

ノード数	1	2	4	8	16
FFT (2^{10}K points)	1	4	8	10	19
FFT (2^{16}K points)	108	212	213	197	220
FFT (2^{20}K points)	2042	2737	2382	1948	1677
Ocean (258^2 ocean)	2020	6682	10108	11246	5997
Ocean (514^2 ocean)	7508	22859	49319	34376	10631
Water (512 molecules)	2132	1369	786	449	349
Water (8000 molecules)	31645	19936	11173	5756	4051
Water (27000 molecules)	105299	68010	39149	22366	12834

表 3 SPLASH2 プログラムのキャッシュヒット率，フォールドハンドラ・バリア同期の実行頻度

	nodes	L1 hit	L2 hit	フォールトハンドラ	バリア
FFT 2^{20}K	2	98.0%	59.5%	3888.6 [回/node/sec]	4.0 [回/sec]
	4	98.4%	60.3%	1777.1 [回/node/sec]	4.6 [回/sec]
	8	98.5%	61.1%	885.0 [回/node/sec]	5.6 [回/sec]
	16	98.6%	57.9%	451.4 [回/node/sec]	6.6 [回/sec]
Ocean 258^2	2	97.3%	43.7%	3953.6 [回/node/sec]	166.3 [回/sec]
	4	98.2%	61.2%	1477.3 [回/node/sec]	187.5 [回/sec]
	8	98.7%	68.7%	330.5 [回/node/sec]	127.4 [回/sec]
	16	99.2%	70.1%	127.7 [回/node/sec]	263.6 [回/sec]
Water 2.7K	2	98.1%	86.4%	503.6 [回/node/sec]	0.35 [回/sec]
	4	98.1%	83.2%	288.2 [回/node/sec]	0.61 [回/sec]
	8	98.2%	81.2%	159.2 [回/node/sec]	1.07 [回/sec]
	16	98.3%	78.0%	99.4 [回/node/sec]	1.87 [回/sec]

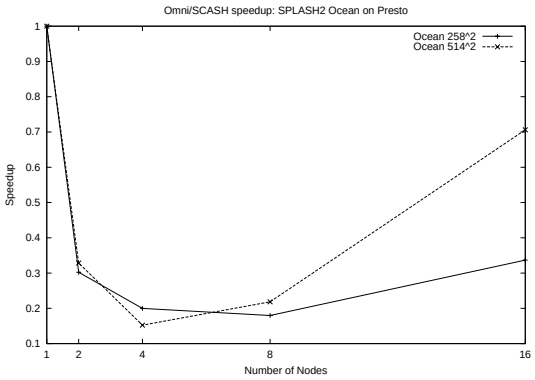

図 3 Presto における Ocean のスピードアップ

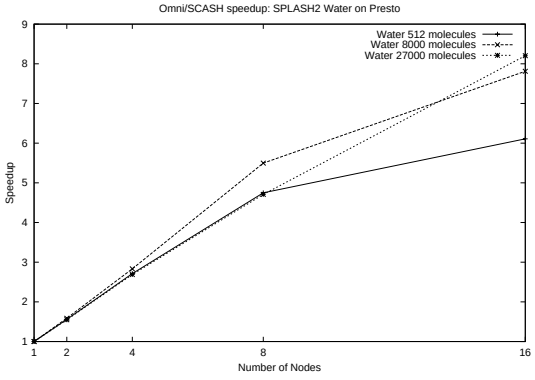

図 4 Presto における Water のスピードアップ

ピードアップ率が 1 を越えるが，表 3 に示したように FFT では L2 キャッシュヒット率が低いこととフォールトハンドラの起動頻度が非常に高いため望ましいスピードアップを達成できていない．このフォールトハンドラの非常に高い起動頻度は，FFT の共有メモリ領域への不規則なデータアクセスによる共有メモリへのアクセスミスからきており，リモートメモリからのデータの取得など通信処理が含まれるため，オーバーヘッドとなっている．

Ocean のスピードアップ率が 1 より落ち込んでしまう原因は，共有メモリ領域のほとんどが実行時に動的に確保されており，動的な共有メモリ領域確保はデフォルトではノード 0 上で行われることによる，ノード 0 へのアクセス集中，またそれによるフォールトハンドラの起動頻度の高さにある．加えて，Ocean ではバリア同期の頻度も非常に高いことがノード数を増やしたときの性能低下につながっている．

Water では問題サイズ 512 molecules の時こそ実行時間が小さくなり過ぎて 16 ノード以上で性能向上が頭打ちになってしまうものの，8000 molecules や 27000 molecules の時には良好なスピードアップを示している．これは表 2 に示した通り，Water はフォールトハンドラおよびバリアの起動頻度がいずれも低いという，通信レイテンシの大きい分散メモリにとって好ましい特徴を持っているためである．

4.4 ローカリティに関する考察

Ocean-contiguous のスピードアップ率が 1 より落ち込んでしまう原因は，共有メモリ領域のほとんどが

表 4 Hetero クラスタの仕様

	前半 8 ノード	後半 2 ノード
CPU	Pentium III 500MHz	Celeron 300MHz
Cache	512KB	128KB
Chipset	Intel 440BX	同左
Memory	SDRAM 512MB	同左
NIC	DEC 21140 chip	同左
NIC	Myrinet M2M-PCI32C	同左

ノード 0 上に確保されていることにあった．そこで本研究では Ocean の共有領域のホームノード指定を行うことで性能の改善を試みた．

Ocean では主なデータは 3 次元配列になっており，1 次元目がプロセッサ，2,3 次元目が x, y 方向を表している．したがって，主なデータを 1 次元目でブロック分割しそのホームを各ノードに割り当てることによってローカリティの向上が望める．

具体的には Omni/SCASH が提供している，動的に共有メモリ領域を確保する命令である `ompsm_galloc(int sz, int mode, int arg)` を**図 5** のように使用した．ompsm_galloc() の第二引数で共有データの配置方法を指定する．DEST_BLOCK でブロック分割を，DEST_DIRECT で第三引数 arg で指定したノード上に共有データを配置することを，DEST_NONE でデフォルトの配置方法を指定する．

さらに頻度の高いバリア同期のオーバーヘッドの削減のため，パラレルリージョン内において参照のみで代入の行われていない共有変数を threadprivate に指定し，パラレルリージョンの入り口で各スレッドに copyin することでバリア同期時にメモリ同期の必要な変数を減らした．

共有データのホームノード指定の結果とメモリ同期の必要な変数の削減により性能の向上が期待されるが，実際にはノード数によってはさらに性能を低下させる事になってしまった．詳しい原因は現在調査中である．

5. ヘテロ環境における性能評価

つぎに性能ヘテロなクラスタ環境における Omni/SCASH の性能を検証する．ここでは，先の実験で一定の性能を得られた Water-spatial を用いる．

5.1 実 行 環 境

評価のために新たに 10 ノード構成の性能ヘテロなクラスタ（便宜上 hetero クラスタと呼ぶことにする）を用意した．

ソフトウェア環境は 4 節で示した presto の環境と同等である．**表 4** に各ノード性能を示す．

5.2 実 行 方 法

評価は Water 27000 molecules に関して（1）すべてのノードが同一性能の場合，(2）常に 1 ノードだけ性能の低いノード（具体的には Celeron 300MHz を 1 ノード）が含まれる場合，(3）常に 2 ノード性能の低いノード（具体的には Celeron 300MHz が 2 ノード）が含まれる場合，の 3 つのケースについて性能測定を行った．以降それぞれのケースを homo, hetero-1, hetero-2 と表記する

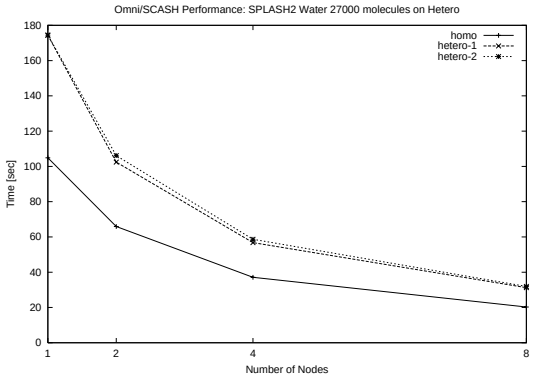

図 6 性能ヘテロなクラスタにおける SPLASH2 Water の性能

表 5 Homo からの性能低下率

ノード数	1	2	4	8
hetero-1	166.3%	155.3%	153.0%	154.0%
hetero-2	166.3%	161.0%	158.1%	157.4%

5.3 実 行 結 果

図 6 に実行時間のグラフを示す．ここでは性能の低いノードを混在することによる性能低下に着目し，**表 5** に hetero-1, hetero-2 それぞれの homo からの性能低下率を示す．

この性能低下率には，性能の低いノードが 8 台中 1 台ないしは 2 台含まれるという絶対的な原因もあるが，より大きな性能低下の要因はロードアンバランシングによるアイドルノードの存在である．たとえば，多少の誤差を覚悟の上，低性能ノード 8 台からなるクラスタを想定し，その環境でも homo と同程度のスピードアップが起ると仮定し，1 台あたりの仕事量を見積もり，最適なロードバランシングが行えたとすると，8 ノード時で hetero-1 では 108% 程度，hetero-2 では 117% 程度の性能低下率で押さえられる可能性があると見積もることができる．

従来の一般的な並列言語では，一般にデータ・タスク分割をユーザが陽に行う必要があるため，このような性能の不均質などを原因とするロードアンバランスの問題に対して，手でロードバランスを行うようなコードを入れない限り，簡単には性能の可搬性を維持できない．

それに対して OpenMP では一般にユーザはコンパイラに対し並列性の指示を行うだけで，明示的なデータ・タスク分割は行わないため，処理系における最適化の余地がある．また，OpenMP のワークシェアリング構文には標準で dynamic や guided などの動的なデータ分割を行うためのスケジューリングポリシー

```
1  d_size = ...;
2  for (i = 0; i < nprocs; i++) {
3      ...
4      work1[i][0] = (double **)ompsm_galloc(d_size, DEST_DIRECT, i);
5      work1[i][1] = (double **)ompsm_galloc(d_size, DEST_DIRECT, i);
6      ...
7  }
```

図 5 ompsm_galloc() によるホームノード指定を伴う共有領域の確保

がある．そのため性能の可搬性の維持が期待できる．

ただし，本実験で用いた FFT, Ocean, Water はもともとそれ自身でデータ・タスク分割を行っており，大幅な変更無しには OpenMP によるロードバランシングは行えない．

6. まとめと今後の課題

以上，本稿では RWCP で開発された SDSM 上の OpenMP である Omni/SCASH の性能評価を，SPLASH2 FFT, Ocean, Water を用いて行った．分散メモリシステム上で動作する Omni/SCASH を用いることによって，ユーザーは共有メモリシステム向けに記述されたプログラムを，些細な変更が必要になることもあるが，基本的には変更なしに分散メモリシステム上で動作させることができる．さらに Omni/SCASH が独自に提供するデータマッピング命令によって，分散メモリシステムの特性にあった最適化を行うこともできる．

評価の結果，キャッシュヒット率，フォールトハンドラ・バリア同期の起動頻度というプログラムの性能向上に寄与する特性を明らかにした．各プログラムに関しては，Water ではもともとのプログラムの分散メモリシステムに好ましい特性上，良好な性能向上が見られた．しかし，FFT では不規則な共有データアクセスから来る共有データアクセスミスによるフォールトハンドラの起動頻度が非常に高いため，性能向上が得られなかった．また，Ocean では動的に確保された共有データがノード 0 に集中し，ノード 0 へのアクセス集中，フォールトハンドラの起動頻度増加，さらにバリア同期の頻度の高さからやはり性能向上が得られなかった．Ocean では共有データのホームノードを調整することによって性能の向上を試みたが，本稿執筆時点では，かえって性能が低下する場合もあるという結果になってしまった．この点に関しては，今後詳細な調査を行う予定である．

さらに，性能ヘテロなクラスタにおいても評価を行った．性能ホモな構成に対する性能低下の主な要因はロードアンバランスであり，最適なロードバランスが行えたと仮定したときの理想的な性能低下率を見積もった．OpenMP では性能ヘテロなクラスタ上でも性能の可搬性を維持できる可能性を指摘したが，本研究で移植した SPLASH FFT, Ocean, Water はそれ自身でデータ分割などのスケジューリングを行っており，OpenMP による半自動的なロードバランシングを確認できなかった．この点に関しても，早急に確認する必要がある．

参 考 文 献

1) 佐藤三久, 原田浩, 石川裕. 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. 情報処理学会ハイパフォーマンスコンピューティング研究会 (SWoPP 2000), pp. 77–82, August 2000.
2) 原田浩, 手塚宏史, 堀敦司, 住元真司, 高橋俊行, 石川裕. Myrinet を用いた分散共有メモリシステムの評価. ハイパフォーマンスコンピューティング研究会資料, pp. 73–78, 1998.
3) 草野和寛, 佐藤茂久, 佐藤三久. OpenMP コンパイラの試作と評価. 情報処理学会ハイパフォーマンスコンピューティング研究会, 第 99 巻, pp. 7–12, 1999.
4) OpenMP ArchitectureReview Board. *OpenMP Fortran Application Program Interface Version 2.0*, November 2000. http://www.openmp.org/.
5) OpenMP ArchitectureReview Board. *OpenMP C and C++ Application Program Interface Version 1.0*, October 1998. http://www.openmp.org/.
6) Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, JaswinderPal Singh, and Anoop Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proceedings of the 22nd International Symposium on Computer Architecture*, pp. 24–36, June 1995.
7) 堀敦史, 手塚宏史, 高橋俊行, 住元真司, 曽田哲之, 原田浩, 石川裕. クラスタ上のプログラミング開発環境 – SCore クラスタシステムソフトウェア –. 情報処理学会研究報告 (SWoPP1999), 99-HPC-77, pp. 83–88, August 1999.
8) http://www.goof.com/pcg/.
9) Intel. *IA-32 Intel Architecture Software Developer's Manual – Volume 3: System Programming Guide*, 2000.