ハイパフォーマンス
コンピューティング 83－7
（2000．10．6）

Cenju-4の分散共有メモリ機構を用いたOmni OpenMPコンパイラ

草野和寛† 佐藤三久†
細見岳生†† 妹尾義樹††

共有メモリ向けの並列化インタフェースであるOpenMPプログラムを、分散メモリシステム上でも透過的に動作させる機能の研究が注目されている。我々は、NECで開発したCenju-4に搭載されている、ハードウェアによる分散共有メモリ(DSM)機能を利用する機能を開発し、それをOmni OpenMPコンパイラに組み込んで初期評価を行った。並列プログラムの変換は、ソフトウェアDSM向けに開発したシステムを利用することが可能であり、実行時ライブラリの移植のみで対応可能であった。ベンチマークプログラムを用いてこのシステムの初期評価を行ったところ、NPBのCGで2PEを用いた並列実行が、逐次実行とほぼ同じ性能であったが、4PE以上で性能向上することが確認できた。

The Omni OpenMP Compiler on Cenju-4 distributed shared memory

Kazuhiro Kusano,† Mitsuhisa Sato ,† Takeo Hosomi ††
and Yoshiki Seo††

OpenMP is a proposed standard interface which parallelize a program for a shared memory multi-processor. This paper describes an implementation and a preliminary evaluation of the Omni OpenMP compiler on an NEC Cenju-4 which support hardware distributed shared memory(DSM) system. OpenMP programs can execute on a distributed memory machine with hardware DSM by using the Omni. The performance of OpenMP benchmark programs scales up, though the execution time using two processors is almost the same as the sequential one.

1. はじめに

マイクロプロセッサの高性能化やネットワークの高速化により、安価なクラスタ型の並列計算機が高性能な並列処理プラットフォームとして一般に利用されるようになっている。しかし、このような分散メモリでの並列実行にはMPIなどのメッセージ通信ライブラリを用いる必要があるため、並列プログラムの開発が困難な作業となっている。

これに対して、SMP構成の計算機がWSだけでなくPCでも一般的になってきており、様々な分野で並列計算機が利用可能になってきている。このような共有メモリ向けのプログラム並列化インタフェースとしてOpenMP[1]が提案され、注目を集めている。OpenMPは、これまで並列計算機それぞれで異っていた並列化に関する機能や指示フォーマットを統一し、移植性の高い並列プログラム開発を可能にすることを目的としている。この仕様には、多数のベンダも対応を表明しており、共有メモリ向け並列プログラムの標準インタフェースとして期待されている。

OpenMPの仕様[2]では、データ並列を利用して並列実行可能なループ、同期位置などをコンパイラに指示するフォーマットや実行時ライブラリを定義している。これらのうち、コンパイラに対する指示は全てコメント(Fortran)、またはpragma(C/C++)の形で行う。OpenMPで並列化したプログラムは、追加した指示を無視すれば、全く同じ逐次プログラムとして扱うことができ、並列化したプログラムと逐次プログラムを同じソースファイルで管理することが可能となる。さらに、OpenMPでは並列化する部分をインクリメンタルに指示して並列化していくことも可能である。以上のように、OpenMPを利用することで、これまでスレッドライブラリやメッセージ通信を利用していた共有メモリ上での並列プログラムの作成を容易に行うことができる。

我々は、OpenMPを並列化インタフェースとした並列処理環境を構築するため、共有メモリ型の並列計算機用のOmni OpenMPコンパイラ[3][4][5]を開発し、既に公開☆している。本研究は、SMP向けのOmniをベースにして、SMPクラスタやPCクラスタなど分散メ

† 新情報処理開発機構つくば研究センタ
RWCP Tsukuba Research Center
†† 日本電気(株)
NEC

☆ http://pdplab.trc.rwcp.or.jp/Omni/

モリでもOpenMPを並列化インタフェースとした並列処理環境を構築することを目的としている。このために、これまでに分散メモリのPCクラスタ上で、ソフトウェアで仮想的な共有メモリシステムを実現するソフトウェア分散共有メモリシステム (Software Distributed Memory System: SDSM) であるSCASH[6]を利用してOpenMPプログラムを並列実行する研究を行っている[7]。この結果、SDSM上でOpenMPの並列実行を行った場合でも、16台程度までは性能向上するすることや、メモリのホームノード割り当てと計算の関係が処理性能に与える影響が大きいことが確認されている。

分散メモリシステム上で共有メモリを利用するためには、SDSM以外にハードウェアで実現した分散共有メモリ (DSM) を利用することが考えられる。このような計算機としてNECのCenju-4[8][9]やSGIのOriginなどがある。本稿では、Cenju-4の提供するハードウェアで実現したDSM機能を利用してOpenMPプログラムを並列実行するために行ったOmni OpenMPコンパイラへの変更と、その性能の初期評価について述べる。

以下、2章でOmni OpenMPコンパイラと、今回使用した計算機Cenju-4の概要を述べる。次の3章でCenju-4のDSMに対応するために行った部分を述べた後、4章で初期評価とその結果を述べる。5章において関連研究について触れ、最後の6章でまとめと今後の課題を述べる。

2. プラットフォーム

2.1 Omni OpenMP コンパイラ

Omni OpenMPコンパイラ[3][4]は、我々が開発しているOpenMPを並列化インタフェースとした並列処理環境のベースとなるSMP向けのOpenMPコンパイラである。この構成を図1に示す。

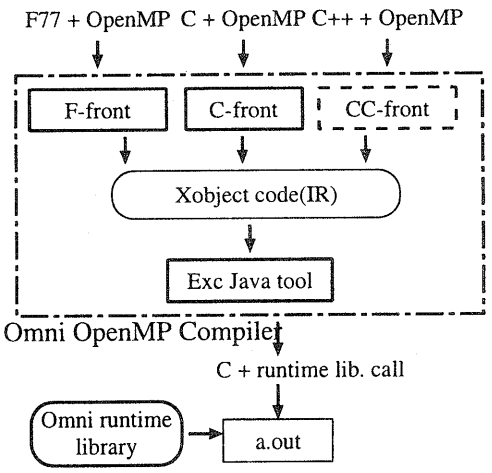

図1 Omni OpenMPコンパイラの構成

Omni OpenMPコンパイラは、入力であるOpenMPプログラムを、マルチスレッドを用いて並列実行するライブラリ呼び出しを含むCプログラムに変換するトランスレータである。システムは、フロントエンド部、プログラム解析およびOpenMP指示変換部、そして実行時ライブラリの三つの部分で構成されている。

入力プログラムは、まずフロントエンドにおいてOmniの中間表現であるXobjectに変換される。現在、CとFORTRAN77に対応したフロントエンドが動作している。中間表現であるXobjectは、変数の型情報，グローバルな宣言情報，および実行文を保持する構文木の3種類の情報で構成される。また、中間表現の生成段階において、インクルードファイルは展開され、以降の処理では入力プログラムの一部として扱われる。

次に、Javaで記述されたクラスライブラリであるExc Java toolにおいて、OpenMP構文を含めたプログラムの解析と並列プログラムへの変換を行う。並列プログラムに変換されたXobjectは、さらにCプログラムに変換して出力される。このライブラリでは、構文木のノードをJavaのオブジェクトとして表現しており、高レベルでの変換を容易に行うことが可能となっている。

最後に、変換されたマルチスレッドCプログラムを、通常のCコンパイラによりOmniの実行時ライブラリとリンクすることで並列実行する。実行時ライブラリは、システムが提供するスレッドライブラリを用いてSMP上で並列実行を行う。この開発では、様々な計算機上で容易に動作することを考慮して設計している。これまでに、Omni はSUN Solaris(Sparc,x86)やLinux(2.2.x)、それにPOSIX threadをサポートしているSMPプラットフォームで動作している。

2.2 Cenju-4 の DSM

Cenju-4[8][9]は、NECで開発したNUMA(Non-Uniform Memory Access)構成のマルチプロセッサマシンである。Cenju-4の各ノードは、MIPSのR10000をCPUとして、二次キャッシュ(1MB)および主記憶(上限512MB)を持ち、このノードを4x4のクロスバースイッチを用いて構成した多段ネットワーク(max: 1024PE)で接続している。このネットワークは、2ノード間の到着順序の保証やマルチキャストなどの機能をサポートしている。Cenju-4のメモリは、各ノードそれぞれが持つ分散メモリ構成となっているが、メッセージ通信だけではなく、ハードウェアによる分散共有メモリ(DSM)機能もサポートしている。DSM機能は、メッセージ通信と同じメモリコントローラ(MC)部分で実現している。

Cenju-4のDSMは、ハードウェアがディレクトリベースの無効化型プロトコルを用いてキャッシュのコヒーレンス制御を行うことで実現している。このディレクトリ管理の単位は、128バイトのブロックであり、このブロック単位にコピーを保持しているノード番号の

管理を行っている。管理データであるディレクトリは、ノード数に関わらず一定で各ブロックごとに64bitであり、メモリ上に確保される。ディレクトリは、メモリの状態とコピーを保持しているノードの管理情報を保持している。ここで管理しているノード情報は、4ノード以下の場合にはポインタ形式で、それ以上の場合にはビットパターン形式となる。

メモリをプライベート領域としてアクセスするか、共有領域としてアクセスするかはOSが管理しており、ページ単位に切り替えることが可能である。実際の利用では、プログラムにおいてDSMを確保するライブラリを呼び出すことで共有領域を確保し、その領域に共有変数を割り当てていく。この割り当て後は、プログラム中ではプライベート領域の変数と同様に扱うことができる。Cenju-4のDSMを使用する際の制限は、まず各ノードの物理メモリサイズが利用できる共有領域の上限となる。また、MIPSプロセッサの制限としてユーザ利用可能な領域が2GBという制限(32bit)があるため、各ノードで確保する共有領域の合計サイズは2GBが上限となる。

Cenju-4のOSはMACHをベースにして開発されたDE4であり、本稿の実験ではこれに分散共有メモリサーバを付加して用いている。このシステムでは、共有メモリ領域はDSMの管理ブロック(128B)単位に、並列実行ノードにインターリーブした形でホームノードが割り当てられる。今回利用したDSMライブラリでは、このホームノード割り当て方法をユーザが指定するインタフェースは持っていない。共有領域での変数や配列の割り当てにおいて、異る変数は全て異るDSMブロックに割り当てられている。変数を割り当てたブロックのホームノードは、割り当て順序と実行プロセッサ数、そして確保する領域サイズにより予測可能である。しかし、今回の評価ではこの情報を利用した最適化などは行っていない。

3. Cenju-4 DSM向けのプログラム変換

本章では、Cenju-4のDSMに対応するために行ったOmni OpenMPコンパイラの対応に関して述べる。

3.1 基本方針

PCクラスタ上で動作するSDSMであるSCASH向けのシステムでは、UNIXのshmemなどで実現される共有メモリを用いて並列実行を行う方式であるshmemモデルを設定し、それに対応した変換を行っている[7]。Cenju-4のようにハードウェアで実現されたDSMも、SDSMと同様にプログラム中で宣言されたメモリ空間を複数PEで共有する形となる。このため、Cenju-4のDSMも基本的にこのshmemモデルの一例と捉えることができ、SCASH向けの変換と基本的に同じ方針で対応することができる。ただし、並列実行するユーザプログラムの起動方法や、様々なライブラリ関数などは当然異るため、それらに対応した変更は必要である。

ここで、shmemモデルへのOpenMPプログラムの変換項目を以下にあげる。

- 静的な大域変数の宣言を、実行時に割り当てる共有メモリ領域へのポインタ変数の宣言に変換。
- プログラムに含まれている上記大域変数への参照を、上記ポインタ変数を用いた間接参照に変換。
- コンパイル単位(ファイル)に共有変数を割り当てる初期化関数を生成。

これらの変換は、Cenju-4向けのシステムでもそのままの形で適用される。

3.2 Cenju-4での並列実行とOmniの実行時ライブラリ

OpenMPプログラムは、fork-join型の並列実行モデルを採用している。Cenju-4では、OpenMPスレッドは各プロセッサに1対1に対応しており、ネストした並列実行には対応していない。

Omniの実行時ライブラリでは、プログラムは1台のマスタで実行が開始され、その後forkにより並列実行を開始する構造となっている。これに対して、Cenju-4における並列実行は、MPIなどと同じく、並列実行を指定されたプロセッサ台数全てのノードにおいて、ホストマシンにおいてコマンドラインで指定されたプログラムが起動される。このため、forkを利用してマスタと同じ環境設定を行っている部分を、共有変数を用いて行うように変更する必要がある。

Omniでは、OpenMPの並列実行部分である'parallel'を指定されたコードは別関数にされ、この関数を並列実行するように変換される。Cenju-4では、各プロセッサ上で動作しているOpenMPスレッドは、同じプログラムを実行する別プロセスとなる。このうち、マスタのみがユーザプログラムのメイン部の実行を開始し、他のスレーブとなるプロセッサでは、ライブラリに含まれているスレーブ用の関数で待機状態となる。そして、並列実行の開始時点でマスタがスレーブの待機状態を解除し、並列実行を開始する。

3.3 Cenju-4の実行環境

Cenju-4向けのOmni OpenMPコンパイラを作成するに当たり、まずOmni OpenMPコンパイラが、並列実行するマシン上でコンパイルするnative環境を想定している点が問題となった。Omniはconfigの実行時に'int'や'double'などのサイズ、それにalignmentを調べている。このために、テストプログラムをコンパイルして実行している部分がある。また、並列化したプログラムで宣言が必要な関数などを調べるため、システムのヘッダファイルをチェックしている。ここで調べたサイズなどの情報は、配列サイズの計算や実行のテンプレート作成などで利用している。

これに対して、Cenju-4は基本的にクロスコンパイル環境であるため、まずこの点が問題となった。しかし、今回のCenju-4での評価は、初期評価ということもあ

り、各モジュールのコンパイルに必要なパラメータを人手で設定して行い、Omniをクロスコンパイル環境に対応させることは行っていない。

また、前述の通り、Omniの変換部(Exc java tools)はJavaで書かれているが、Cenju-4と今回使用したホスト(EWS4800/310)で利用可能なJavaの処理系がなかった。しかし、Cenju-4向けの並列プログラムは基本的にSCASH向けのものと同じであるため、この部分にはSCASH向けのものを利用することができる。このため、今回の評価では、Javaを用いた変換部は異る計算機上で動作している、SCASH向けのものを利用した。ただし、中間表現であるXobjectに変換するフロントエンドは、ヘッダの展開を行うため、Cenju-4のフロントエンドであるEWS4800上のクロスコンパイル環境を利用する。Omniの実行時ライブラリは、Cenju-4で動作させる他のプログラムと同様に、フロントエンドのクロスコンパイル環境でコンパイルして利用した。

4. 性能評価

4.1 Cenju-4での性能評価

512MBのメモリが搭載された16ノード構成のCenju-4を用いて、Cenju-4のDSM向けOmni OpenMPコンパイラの初期評価を行った。今回はCプログラムをOpenMPにより並列化した場合の性能評価を行った。そのコンパイラには、gcc 2.6.3をベースにしたCenju-4用のクロスコンパイラを、ホスト計算機であるNEC EWS4800/310上で利用し、最適化オプションには'-O3 -funroll-loops'を指定した。並列プログラムの起動には、Cenju-4が提供している'cjsh'を利用した。このコマンドは、引数で指定したユーザプログラムを、実行PE全てで起動するライブラリである。

性能評価には、OpenMPで並列化した以下のCプログラムを利用した。

- Cで記述されたNPB1[10]のCG(class A)
  OpenMPの並列化はorphan指示を利用し'parallel'はプログラムで一回のみ行う。
- 4点ステンシルのJacobi法によるLaplace方程式を解く(行列サイズ1024x1024, 繰り返し50回)ベンチマーク
  二次元配列をアクセスする二重ループの外側で並列化。

図2にCenju-4上で上記ベンチマークを並列実行した実行時間を、図3に逐次実行に対する速度向上率を示す。ここで、図2のPE0における値は、OpenMP指示を無視して逐次実行した場合の実行時間である。この結果を見ると、cgとlaplace共に2台で並列実行した場合の性能が逐次実行とほとんど変らない実行時間となっていること、そして2台以上では性能向上が得られていることがわかる。また、OpenMPで並列化したものを1台で実行した際には、1割強性能が低下しているこ

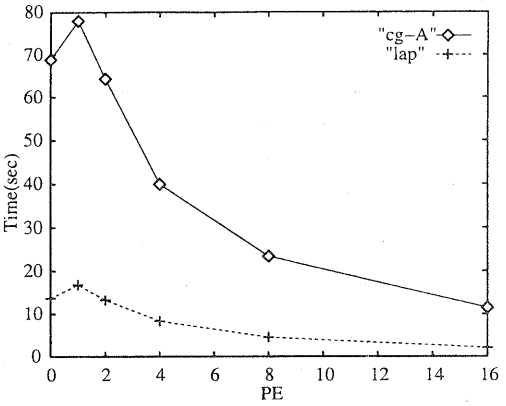

図2 Cenju-4での実行時間

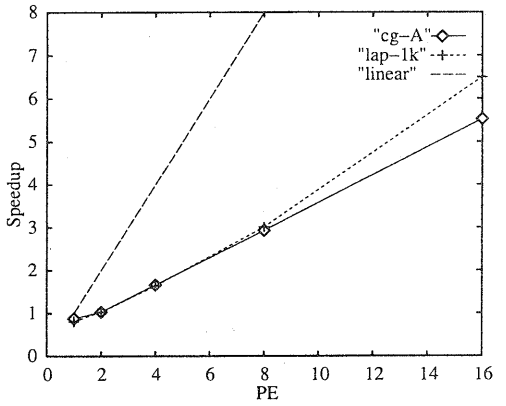

図3 Cenju-4でのOmniの性能向上

とがわかる。ここで、OpenMPの並列化ループでは、staticなどのスケジューリング方法は指定せず、Omniの既定値である静的なブロック分割を利用している。これに対して、共有領域での配列はDSMの管理ブロック単位に並列実行ノードにインターリーブして割り当てられている。したがって、プログラムで利用される配列サイズがノード数より十分に大きく、リニアにアクセスされるlaplaceでは、同一ブロックにアクセスするノードは高々2ノードとなる。これを考えると、2台の並列実行でほとんど性能向上が得られていないのは、プログラムの並列実行以外に性能向上を阻害している要因があると考えられる。

4.2 Cenju-4でのOpenMPのオーバヘッド

OpenMPを用いた並列化で利用される指示の一部に関して、Cenju-4におけるオーバヘッドを測定した。ここで測定したのは、'parallel', 'for', 'parallel-for', 'barrier', 'reduction'の五種類である。この測定にはMicrobench[11]を利用した。この結果を図4に示す。この結果から、Cenju-4における'parallel'などの

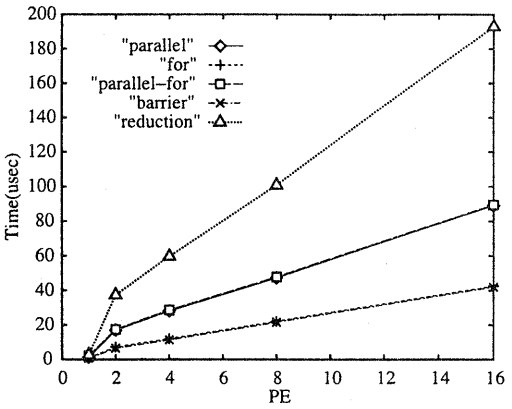

図4 Cenju-4 での OpenMP 指示のオーバヘッド

OpenMP 指示のオーバヘッドは、PC や SUN の SMP で得られた結果と比較すると大きく、しかも台数にほぼ比例した時間を要していることがわかる。しかし、ベンチマークで並列化したループの粒度と比較すると、性能低下の要因となるとは考えにくい。また、Cenju-4 のライブラリ関数をそのまま利用しているバリア同期のコストは、文献 8) において、各ノードが全てのノードに同期カウンタの atomic add 命令を送り、各ノードにおけるカウンタの値を見る方式の同期コストとほぼ等しい。

4.3 考　　察

Cenju-4 の DSM に関する文献 9) での NPB2.3 を用いた評価結果では、2 台の場合も含めて 16 台まで良好な台数効果が得られている。使用言語の違いもあるため単純な比較はできないが、本実験で得られた評価結果とは、明らかに異る傾向となっている。この違いの一部は、OpenMP の並列化によって、入力プログラムが変更されていること、および Omni や OpenMP のライブラリ呼び出しが増えているためであると考えられるしかし、OpenMP のオーバヘッドを測定した結果からは、OpenMP のオーバヘッドが 2 プロセッサにおいて性能向上しないこと主要な原因であるとは考えられるほど大きくなかった。この原因は現在調査中であるが、ソフトウェア DSM である SCASH における評価ではこのような結果は得られていないので、Cenju-4 に特有の問題があると考えられる。

SCASH 上の実験では、ページベースの SDSM においてはホームノードの割り当て方法が性能に大きく影響することが明らかになっている[7]。今回の評価で利用した Cenju-4 において、ホームノードは DSM の管理ブロック (128B) を単位として各ノードにインターリーブして割り当てられる。このため、規則的なアクセスを行うプログラムでは、大きな配列を扱う並列ループをブロック分割した場合、同じブロックにアクセスするプロセッサは高々 2 つであり、DSM を利用するオーバヘッドも見えにくい。

そこで、CG の並列ループのスケジュールに (static, 1) を指定して、ループの実行を cyclic に各ノードに割り当て、DSM のブロックを複数のプロセッサでアクセスした場合の性能を調べた。これは、同一キャッシュブロックを複数のプロセッサが競合してアクセスした場合であり、DSM において最も大きなオーバヘッドを要する可能性が高い。図 5にスケジューリングを指定しない場合と、(static,1) を指定した実行時間を示す。この結

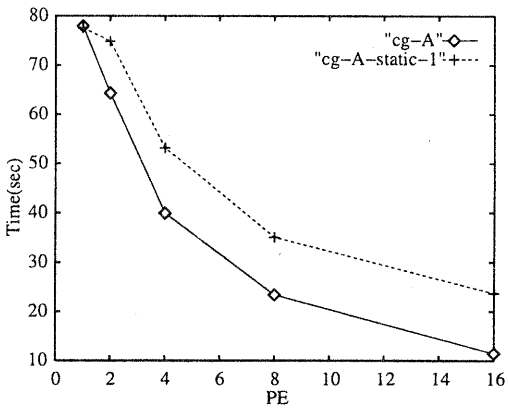

図5 NPB CG のスケジューリングによる性能の違い

果、ブロック分割した時と比較して明らかに遅くなっており、これは DSM のアクセスにおいてオーバヘッドが大きくなった影響であると考えられる。このことから、ハードウェア DSM を利用する場合でも、DSM の管理単位ブロックを複数のプロセッサでアクセスするような場合には、ソフトウェア DSM の場合と同様に性能が低下すると言える。したがって、並列実行で頻繁にアクセスするノードにデータのホームを割り当てることで、最も大きな性能向上が期待できる。

5. 関連研究

分散メモリシステム上で OpenMP を動作させる研究では、OpenMP プログラム (Fortran) を TreadMarks ソフトウェア分散共有メモリを用いて分散環境で実行する研究が Rice 大で行われている[12]。この研究では、分散環境で効率的に実行するために仕様の拡張を行って評価しているため、OpenMP の特徴である可搬性が損なわれることがある。本研究では、OpenMP の仕様拡張は行っていないため、SMP と同じプログラムが分散メモリ上で動作可能となっている。

また、TreadMarks を用いた SMP クラスタ向けの OpenMP コンパイラの研究も行われている[13]。我々も分散メモリである PC クラスタ上で共有メモリを実現する SDSM である SCASH 上で OpenMP プログラムを動作させる機能を Omni に組み込む研究を進めてい

る[7]。

OpenMP と MPI を用いて SMP クラスタのメモリ階層を効率的に利用するプログラミング手法の研究[14]も注目されている。OpenMP 処理系の研究として、Omni と同様にトランスレータ形式であるフリーの処理系が Lund 大で開発されている[15]。OpenMP のベンチマークでは、エジンバラ大が OpenMP を用いた場合のオーバヘッドを測定するマイクロベンチマーク[11]を開発、公開☆している。OpenMP の ARB もベンチマークの開発を進めていることを表明しているが、具体的なことは明らかになっていない。

6. まとめ

以上、本稿では Cenju-4 の DSM 機構を利用する Omni OpenMP コンパイラの設計および実装と Cenju-4 での初期性能評価の結果を述べた。このシステムを利用することで、分散メモリシステムのハードウェア DSM を用いて、OpenMP プログラムを変更なしに分散メモリ計算機上で実行することが可能となる。この結果、ユーザは分散メモリと共有メモリで異る並列化インタフェースを利用する、並列プログラム作成コストを削減することができる。

Cenju-4 の DSM 機構を利用した初期評価の結果では、cg と laplace で 2 ノード使用した場合の性能が、逐次実行とあまり変らなかったが、2 ノード以上では性能向上が得られていた。2 ノードで性能向上が得られていない原因については現在調査中で、詳しいことはわかっていない。また、並列ループのスケジューリングで (static,1) を指定して、cyclic にループの割り当てを行った場合、NPB の cg で DSM のオーバヘッド増加によると思われる性能の低下が見られた。このことから、ハードウェア DSM を用いた場合も、ソフトウェア DSM と同様に共有データのホームノードの割り当て方法が性能に影響することがわかった。

今後の課題としては、cg と laplace を用いた評価において、2PE で性能がほとんど向上していない原因を明らかにすることがある。また、現在は共有メモリ領域を割り当てる際にホームノードを意識していないが、これをホームノードや並列実行を考慮して割り当てを行う機能を加えて評価を行う予定である。さらに、Cenju-4 の DSM で開発されている、first touch によるマッピング機能を利用した場合の、ユーザプログラムへの影響などを評価していく。

参考文献

1) http://www.openmp.org/
2) OpenMP Consortium, "OpenMP C and C++ Application Program Interface Ver 1.0", Oct, 1998.
3) 草野, 佐藤 (茂), 佐藤 (三) : Omni OpenMP コンパイラと実行時ライブラリの性能評価, 並列処理シンポジウム (JSPP2000), pp. 229–236 (2000).
4) M. Sato, S. Satoh, K. Kusano and Y. Tanaka, "Design of OpenMP Compiler for an SMP Cluster", EWOMP '99, pp.32-39, 1999.
5) 佐藤 (茂), 草野, 佐藤 (三) : OpenMP コンパイラにおけるメモリ一貫性制御の最適化, 並列処理シンポジウム (JSPP2000), pp. 221–228 (2000).
6) 原田, 手塚, 堀, 住元, 高橋, 石川, "Myrinet を用いた分散共有メモリにおけるメモリバリアの実装と評価", 並列処理シンポジウム (JSPP'99), pp.237-244, 1999.
7) 佐藤, 原田, 石川, "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", 2000-HPC-82, pp.77-82, Aug, 2000.
8) 加納, 細見, 中村, 広瀬, 中田, "並列計算機 Cenju-4", 並列処理シンポジウム (JSPP '99), pp. 7-14, Jun, 1999.
9) 細見, 加納, 中村, 広瀬, 中田, "並列計算機 Cenju-4 の分散共有メモリ機構", 並列処理シンポジウム (JSPP '99), pp. 15-22, Jun, 1999.
10) David Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, and H. Simon, "The NAS Parallel Benchmarks", RNR-94-007, NAS, 1994.
11) J. M. Bull, "Measuring Synchronisation and Scheduling Overheads in OpenMP", EWOMP '99, pp.99-105, 1999.
12) H. Lu, Y. C. Hu and W. Zwaenepoel, "OpenMP on Networks of Workstations", SC'98, Orlando, FL, 1998.
13) Y. C. Hu, H. Lu, A. L. Cox and W. Zwaenepoel, "OpenMP on Networks of SMPs", Proc. of the Thirteenth International Parallel Processing Symposium, pp. 302–310, 1999.
14) Cappello, F. and Richard, O.: Performance characteristics of a network of commodity multiprocessors for the NAS benchmarks using a hybrid memory model, PACT '99, pp. 108–116 (1999).
15) Brunschen, C. and Brorsson, M.: OdinMP/CCp - A portable implementation of OpenMP for C, EWOMP '99, Lund, pp. 21–26 (1999).

☆ http://www.epcc.ed.ac.uk/research/openmpbench