計算機アーキテクチャ 137－27
ハイパフォーマンスコンピューティング 80－27
（2000. 3. 3）

SMP-PC クラスタにおける OpenMP+MPI の性能評価

吉川茂洋† 早川秀利† 近藤正章†
板倉憲一†† 朴泰祐† 佐藤三久†††

本稿では, 4台のIntel Pentium-II Xeonを搭載したSMP-PCを1ノードとし, 4ノードを100Base-TX EthernetとMyrinetで結合したSMP-PCクラスタ *COSMO* を用いて, RWCPで開発中のOmni OpenMPコンパイラとMPIを組み合わせたハイブリッドプログラミングについて, 典型的なHPCベンチマークを対象に評価する. また, Pthreadsを用いたマルチスレッドプログラミングとMPIを組み合わせたハイブリッドプログラミングと比較して, プログラミングの容易さと性能の点で検討する. その結果, OpenMPとMPIによるハイブリッド並列プログラムは, PthreadsとMPIによるプログラムとほぼ同様の性能向上が得られることがわかった.

Performance Evaluation of OpenMP+MPI on SMP-PC Cluster

Shigehiro YOSHIKAWA,† Hidetoshi HAYAKAWA,†
Masaaki KONDO,† Ken'ichi ITAKURA,†† Taisuke BOKU†
and Mitsuhisa SATO†††

In this paper, we describe the performance evaluation of HPC benchmarks parallelized by OpenMP directives and MPI using RWC Omni OpenMP compiler on an SMP-PC cluster named *COSMO*. We compare two styles of hybrid-programming with mixture of shared and distributed memory paradigms: using OpenMP directives and MPI and using explicit multi-threading and MPI. As a result, it is shown that the parallel program using OpenMP and MPI achieves good speedup as that with multi-threading and MPI.

1. はじめに

近年, サーバタイプのワークステーション(WS)やパーソナルコンピュータ(PC)は, 複数のプロセッサを搭載したSMP (Symmetric Multi-Processor: 対称型マルチプロセッサ)が一般的になっており, 複数のSMPノードをネットワークで結合したSMPクラスタは, 価格・性能比の優れたシステムとして現在注目されている. 我々は, Intel Pentium-II Xeon 4台を結合したSMP-PCをノードとし, これらを100base-TX Ethernet SwitchおよびMyrinet[3]で結合したSMPクラスタCOSMO (Cluster Of Symmetric Multi prOcessor)を構築し, SMPクラスタのHPC向けの利用方法に関する研究を行っている[1].

† 筑波大学 電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba
†† 筑波大学 計算物理学研究センター
Center for Computational Physics, University of Tsukuba
††† 新情報処理開発機構
Real World Computing Partnership

SMPクラスタのアーキテクチャは, 分散メモリアーキテクチャと共有メモリアーキテクチャの混合アーキテクチャである. SMPクラスタで高い計算性能を引き出すためには, 両方のアーキテクチャの利点を活かしたプログラミング方法を選択する必要がある. 我々は文献[1]において, ノード間ではMPIを用いたメッセージパッシングによる分散メモリプログラミング, ノード内ではマルチスレッドによる共有メモリプログラミングを行なうハイブリッドプログラミングに注目し, HPCの代表的なベンチマークを対象に性能評価を行なった. その結果, ノード内の処理においてデータの時間的局所性を引き出し, キャッシュを有効に利用できれば高い性能が得られることを確認した.

上記のようなハイブリッドプログラミングは, 複数のプログラミングパラダイムを考慮しなくてはならず, 分散メモリプログラミングに比べ複雑になり易い. これまでCOSMOにおけるハイブリッドプログラミングとしては, メッセージパッシングライブラリとして事実上の標準であるMPIを用い, 共有メモリプログラミングにはマルチスレッドライブラリであるPthreadsを用いてきた[1]. 一般に, マルチスレッドプログラミングは性能の解析や最適化が難しく, プログラミング

コストが高い．しかし近年，共有メモリプログラミングの標準となるべく OpenMP[2] が提案され，現在多数のベンダやユーザによって実装が進められている．OpenMP ではプログラム中に並列性を記述するための指示文を追加することによって，比較的容易に並列プログラミングが行なえる．したがって，共有メモリプログラミングに OpenMP を用いることにより，ハイブリッドプログラミングのプログラミングコストを抑えることができると期待される．

現在 COSMO 上では，RWCP 並列分散システムパフォーマンスつくば研究室で開発中の OpenMP コンパイラである Omni-1.0[4] が利用可能である．本稿では，共有メモリプログラミングに OpenMP を用いて，MPI と組み合わせたハイブリッドプログラミングとその評価を行なう．また，COSMO にインストールされている Omni OpenMP コンパイラは Pthreads ライブラリを用いて実装されている．そこで文献[1] において，Pthreads と MPI を用いてハイブリッドプログラミングを行なった HPC ベンチマークに対して，本稿では OpenMP と MPI を用いてハイブリッドプログラミングを行ない，その性能を比較評価する．

次節では COSMO の仕様について，3 節では OpenMP と MPI を組み合わせたハイブリッドプログラミングの方法について述べる．4 節では HPC ベンチマークを対象にして行なった実験結果を示し，5 節では COSMO のメモリバス性能の問題に関して考察を行なう．最後に，まとめと今後の課題について述べる．

2. COSMO の仕様

図 1に COSMO の構成を示す．COSMO の各ノードは Intel Pentium-II Xeon (450MHz, 16KB 4way L1 データキャッシュ, 512KB 4way L2 ユニファイドキャッシュ) を 4 台搭載した DELL PowerEdge6300 (450NX chip-set, 主記憶 512MB) である．全体では 4 ノード構成であり，ノード間は 100base-TX Ethernet Switch と 800Mbps Myrinet[3] で接続されている．OS には SMP 対応の Linux 2.2.10 を使用している．

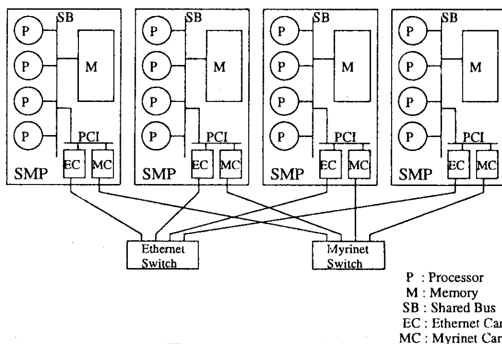

図 1　COSMO の構成

現在，COSMO の上で利用できる並列プログラミング環境は以下の 3 つである．

- **MPI:** MPI ライブラリ (MPICH-1.1.2[6]) がインストールされている．MPICH の標準的な下位通信ライブラリには，TCP/IP を用いて通信を行う ch_p4 を選択してある．(同一ノード内のプロセス間での MPI 通信も TCP/IP を使う)
- **Pthreads:** Linux RedHat 5.2 に含まれている，LinuxThreads[7] (Posix 準拠のスレッドライブラリ) を使用できる．LinuxThreads はカーネルレベルスレッドである．
- **OpenMP:** RWCP 並列分散システムパフォーマンスつくば研究室で開発された OpenMP コンパイラである Omni version1.0[4] が利用可能である．COSMO にインストールされている Omni は Pthreads ライブラリの上に実装されている．

共有メモリプログラミングとして，Pthreads を用いるマルチスレッドプログラミングと，OpenMP を用いる方法の 2 種類を選択でき，MPI と組み合わせたハイブリッドプログラミングとして 2 つのアプローチが可能である．

3. OpenMP+MPI プログラミング

Pthreads と MPI を組み合わせる場合，ノード内で並列処理を行なう適当なブロックを並列実行した後，マスタースレッドのみがノードを代表して通信を行なうことにより，安全な通信が可能となる．各スレッドが勝手に通信を行なうには，MPI 通信に対し何らかの排他制御が必要になる．MPI 通信に対し，我々の研究では MPI 通信は全てマスタースレッドのみが行なうようにする．

OpenMP[2] は fork-join 型の実行モデルを採用している．すなわち，プログラム中で並列実行の指示があった時点で，複数のスレッドを生成し，並列実行部分が終了した時点でマスタースレッド以外のスレッドは消滅する．したがって，指示文で指定される並列に実行される部分領域内に MPI 関数がなければ，MPI 通信は自然にマスタースレッドが行なうというスタイルになる．ただし，並列に実行される部分領域内に MPI 関数がある場合には，上述の Pthreads と同様，ユーザによる明示的な排他制御が必要である．

OpenMP と MPI によるハイブリッドプログラミングを行なう際には，MPI で並列化されているプログラムに OpenMP の並列実行指示文を追加することによって並列化を行なう．プログラム中で対象となるループの並列化可能性やプログラムの整合性はユーザが判断する．この方法で並列化を行なった場合，OpenMP の指示文を無視することにより，MPI のみを用いたプログラムとして実行可能である．また，OpenMP はマルチスレッドライブラリとしても利用できるが，プログ

ラムの書き換えが必要になるので，原則としてここでは行なわない．

4. 性能評価

以下の 2 種類のベンチマークを用いて，Pthreads と MPI によるハイブリッド並列プログラム（以下 Pthreads+MPI 版と略す）と，OpenMP と MPI によるハイブリッド並列プログラム（以下 OpenMP+MPI 版と略す）について比較評価する．また，ハイブリッドプログラミングによる性能向上を確認するために，MPI のみを用いたプログラム（以下 full-MPI 版と略す）の結果も示す．MPI のみを用いる場合，並列プロセスはシステム中の合計 16 台のプロセッサに適宜マッピングされ，同一ノード内のプロセス間でもメッセージパッシングによる通信を行なう．なお，ここに示す full-MPI 版はプロセス間通信の順序をプロセスのマッピングに基づき最適化し，ノード間をまたがる通信量が最小となるようにしてある[1]．

Linpack 密行列を係数行列とする連立一次方程式の解をガウスの消去法によって求める問題．行列サイズは 1000 × 1000 と 2000 × 2000 を扱う．

CG NAS Parallel Benchmarks version 1[8] の Kernel CG．正値対称な大規模疎行列の最小固有値を CG 法によって求める問題．問題サイズとして Class-A, Class-B の二種類を扱う．

Linpack はキャッシュブロッキングを施すことによってデータの時間的局所性を引き出せる．一方 CG は，1 反復分の計算において，最も大きなワーキングセットとなる疎行列の各要素に対し，各々一度だけ読み出しが行なわれるため，データの時間的局所性が低いアプリケーションである．このように，データの時間的局所性の有無に関して相反するアプリケーションとしてこの 2 つをターゲットとした．

full-MPI 版での各プロセッサへのプロセスマッピングは，使用するノードに対しサイクリックに割り当てる．実際に稼働するプロセッサ数は総プロセス数となる．ハイブリッド版での各プロセッサへのプロセス及びスレッドのマッピング方法は，まず，使用するノードでそれぞれ 1 つプロセスを起動し，それぞれのプロセスが担当ノード内でスレッドを起動する．実際に稼働するプロセッサ数は，プロセス数（ノード数）とスレッド数の積となる．full-MPI 版，ハイブリッド版ともに，ノード数を 1, 2, 3, 4, さらに各ノード数毎のプロセッサ数を 1, 2, 3, 4 と変化させ，計 16 通りの場合に対し実行時間を計測した．なお，ノード間ネットワークとして 100Base-TX Ethernet を用いて実験を行ない，今回の性能評価では Myrinet は使用していない．なお，以降の性能評価では速度向上率によって性能を表すが，特に断らない限り，その方式によるプログラムを 1 ノード・1 プロセッサで逐次実行した場合の速度

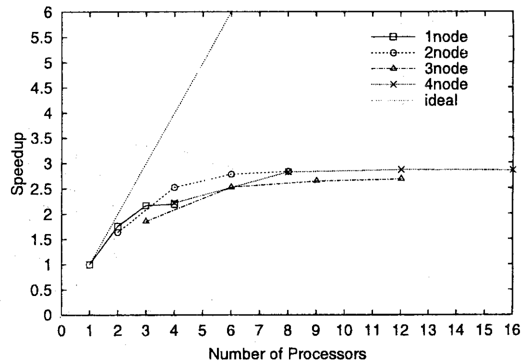

図 2 Linpack 1000x1000: full-MPI 版の速度向上率

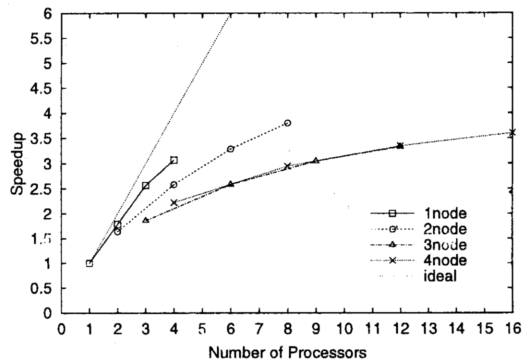

図 3 Linpack 1000x1000: OpenMP+MPI 版の速度向上率

を基準にしている．

4.1 Linpack における性能評価

図 2 に Linpack の行列サイズ 1000 × 1000 における full-MPI 版の速度向上率を示す．ノード当たり 2〜3 プロセッサ用いた場合に性能が飽和し，ノード当たりそれ以上のプロセッサを用いてもは性能はほとんど向上していない．これは，Linpack は各プロセスが自分の担当消去列を計算した後，全プロセスへそのデータをブロードキャストするが，プロセス数の増加に比例して通信回数と通信量が増加することにより，総プロセス数が増えるに従って通信オーバヘッドが大きくなることが原因である．また，SMP ノード内の MPI 通信は，スレッドを用いて共有メモリを積極的に利用した通信に比べて，約 4 分の 1 の性能しか出せない[1] ことも原因である．

図 3 に Linpack の行列サイズ 1000 × 1000 における OpenMP+MPI 版の速度向上率を示す．full-MPI 版に比べ全体的に性能が向上しており，全てのノード数においてプロセッサ数の増加に伴い性能が単調増加している．計算部分にかかる時間は full-MPI 版と同じであるが，消去列を全プロセスへブロードキャストする部分で，ハイブリッド版では送出相手のプロセス数がノード数と等しいために，full-MPI 版に比べ通信

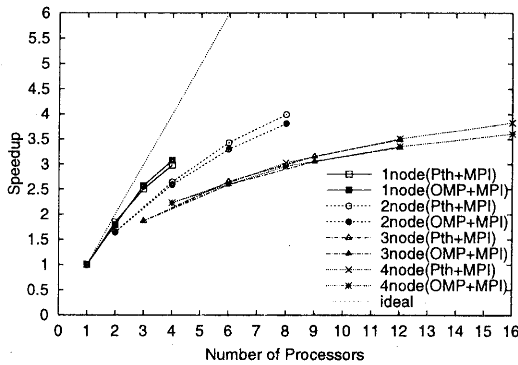

図4 Linpack 1000x1000: OpenMP+MPI 版と Pthreads+MPI 版の速度向上率の比較

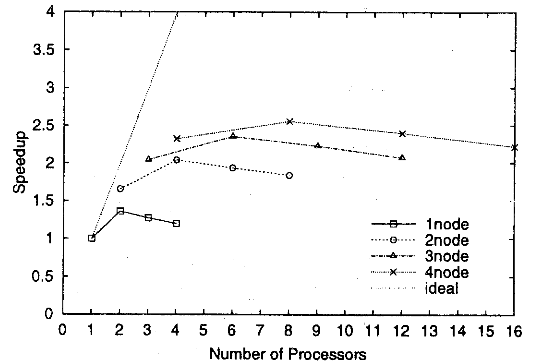

図6 CG Class-A: full-MPI 版の速度向上率

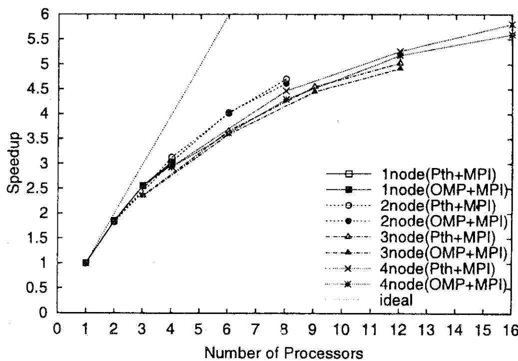

図5 Linpack 2000x2000: OpenMP+MPI 版と Pthreads+MPI 版の速度向上率の比較

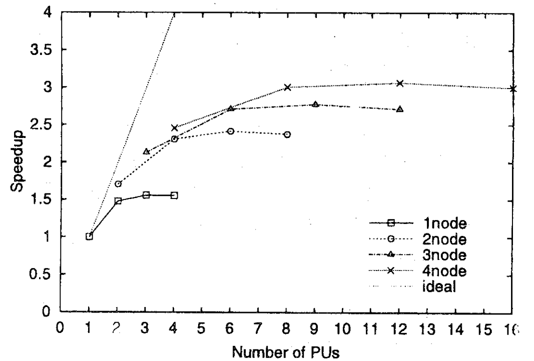

図7 CG Class-A: OpenMP+MPI 版の速度向上率

回数及び通信量が低減されることが原因である.

図4は Pthreads+MPI 版の1ノード・1プロセッサの実行時間を基準にした速度向上率を示している. この図からノード当たり 3, 4 プロセッサにおいて, OpenMP+MPI のスケーラビリティがやや落ちることがわかる. OpenMP の並列実行指示文で並列化した場合, スレッドへのデータのマッピングがユーザによって制御できないため, Pthreads+MPI 版とは異なっている. そこで, OpenMP を用いてマルチスレッドプログラミングを行ない, Pthreads+MPI 版のデータのマッピングと等しくなるようにしたところ, ノード当たり 3, 4 プロセッサにおけるスケーラビリティが向上した. したがって, 図4に見られるスケーラビリティの低下は, スレッドへのデータのマッピングの違いが主な原因であると考えられる.

また, 行列サイズが 1000 × 1000 では計算の粒度が十分に大きくないため, ノード数を増やしたときのスケーラビリティが低い. そこで, 図5に Linpack の行列サイズを 2000 × 2000 にした場合のハイブリッド版の速度向上率を示す. 計算粒度が大きくなったことにより 16 プロセッサを使用した場合に最も良い性能が出ている.

4.2 CG における性能評価

NPB Kernel-CG は, プロセッサ空間を仮想的に 2 次元化し, 対象行列を 2 次元分割してマッピングすることによってプロセス間通信時間を短縮できることがわかっている[9] が, 今回はプログラミングの簡単化のため, 行列を単純に 1 次元ブロック分割し, 各ブロックをプロセッサにマッピングした.

図6に, CG のデータサイズ Class-A における full-MPI 版の速度向上率を示す. ノード内プロセッサ数の増加に対するスケーラビリティが低く, ノード当たり 3, 4 プロセッサを用いた場合性能が低下する. これは, 計算時間の大半を占める行列ベクトル積での, メモリアクセスによるバスの混雑と, 通信回数の増加による通信オーバヘッドの増加が原因と考えられる.

図 7は CG のデータサイズ Class-A における OpenMP+MPI 版の速度向上率である. ノード当たり 2 プロセッサまでは性能が向上しているが, 2 プロセッサ以上にプロセッサ数を増やしても, メモリバスボトルネックのために性能は向上しない. データのサイズを Class B にすると, ノード当たり 2 プロセッサにおいてもメモリバスボトルネックが顕著になり, 図9のように, ノード内のプロセッサ数に対するスケーラ

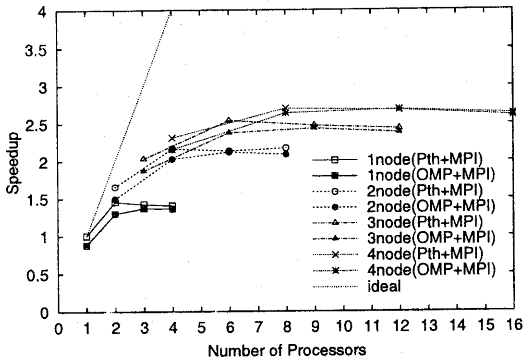

図 8　CG Class-A: OpenMP+MPI 版と Pthreads+MPI 版の速度向上率の比較

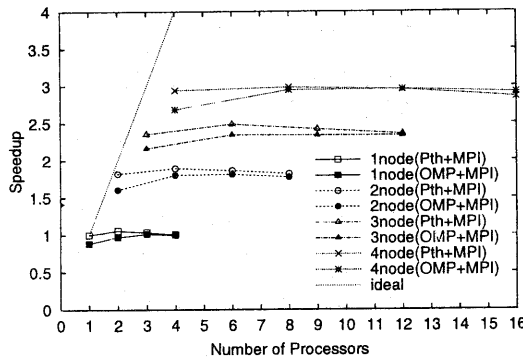

図 9　CG Class-B: OpenMP+MPI 版と Pthreads+MPI 版の速度向上率の比較

ビリティはほとんど得られない．

図 8, 9は Pthreads+MPI 版の 1 ノード・1 プロセッサの実行時間を基準にした速度向上率を示している．グラフから分かるように，OpenMP+MPI 版はノード当たり 1 プロセッサのときの性能が 1 割程度低下している．これは，OpenMP による並列化によって手続きが増えているためのオーバーヘッドであると考えられる．Linpack では，ノード当たり 1 プロセッサの実行時間はそれほど低下していなかった．これは，Linpack では OpenMP の並列実行指示文を 1 行だけ追加したのに対し，CG では約 20 行程度追加しており，さらにプログラムの実行時に OpenMP の指示文が実行される回数も相対的に多いので，並列化によるオーバーヘッドに違いがあると考えられる．一方，ノード当たりのスレッド数を増やしていくと，Pthreads+MPI 版との性能の差はほとんど無くなり，OpenMP+MPI 版でも SMP クラスタの性能を引き出せることがわかる．

5. メモリバスボトルネックの問題

今回性能評価を行なった Linpack では，16 台のプロセッサを用いても約 6 倍という低い性能向上しか得られておらず，よりデータの時間的局所性を引き出すアルゴリズムを用いる必要があるが，プログラミングはそれに応じて複雑になる．さらに，CG のようにメモリへのアクセスが集中するようなアプリケーションでは，ハイブリッドプログラミングを行なったとしても，メモリバスボトルネックのために高い性能が得られない．COSMO のようなメモリバスの性能が低い SMP-PC クラスタにおいて，メモリバスボトルネックは深刻な問題となっている [5]．

表 1　メモリバスのバンド幅 [MB/s]

read

スレッド数	noway-IL.	2way-IL.	4way-IL.
1	263.0	266.0	266.1
2	419.2	440.4	443.2
3	419.7	497.0	527.1
4	417.3	511.9	530.8

write

スレッド数	noway-IL.	2way-IL.	4way-IL.
1	145.3	208.0	213.5
2	145.5	220.2	280.8
3	144.2	186.5	269.2
4	144.2	206.1	259.5

copy

スレッド数	noway-IL.	2way-IL.	4way-IL.
1	134.2	132.6	162.9
2	101.2	148.3	176.2
3	99.0	130.4	170.4
4	96.8	126.0	146.5

※ IL. = interleave

一般にメモリバスバンド幅を向上させる方法として，メモリのインターリーブを有効にすることが考えられる．COSMO の SMP ノードで用いている 450NX チップセットでは，2-way，および 4-way のインターリーブが可能であるが，現在は no-way インターリーブである．そこで，他のノードのメモリを 1 ノードに集中的に装着することにより，メモリのインターリーブ機能を有効にした場合の実験を行なった．表 1は，スレッド数をパラメータとしてメモリバスバンド幅を測定した結果である．表のバスバンド幅は全スレッドのバンド幅の合計を示している．read についてはあまり性能改善は見られないが，write と copy についてはメモリの way 数を増やした場合に 1.2～1.7 倍の性能向上が得られている．特に複数スレッドを用いた場合のバスバンド幅の性能が改善されることがわかった．さらに，Pthreads 版の Linpack，および CG のプログラムを用いて，2-way および 4-way における性能を測定したところ，特にメモリアクセス率の高い CG において性能向上が見られた．図 10に CG のデータサイズ Class-B おける no-way での実行時間を基準にして，メモリの way 数を 2-way, 4-way と変えた場合の速度向上率を示す．使用するプロセッサ数を増やしていくと性能が大きく改善され，4 プロセッサ使用時に 4-way

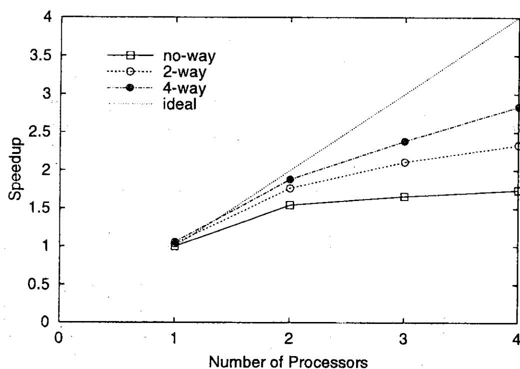

図 10　CG Class-B: メモリの way 数の変更による速度向上率の比較 (1 ノードを使用)

では no-way の約 1.6 倍の性能向上が得られることがわかった. なお, no-way 時のグラフの傾向が図 9 とやや異なっているのは, 測定時のコンパイル環境等の違いによるものである. また, Pthreads と OpenMP の性能差がそれほどないことから, OpenMP を用いた場合でも同様の性能向上が得られるはずである. 今後, COSMO の全ノードのメモリを no-way から 4-way に拡張する予定であり, その場合, この単一ノードにおける性能向上に応じた複数ノードでの性能向上が期待できる.

6. まとめと今後の課題

本稿では, OpenMP と MPI を組み合わせたハイブリッドプログラミングを行ない, マルチスレッドプログラミングと MPI によるハイブリッドプログラムと性能を比較して, ほぼ同じ性能向上が得られることを確認した. 特に Linpack では, MPI 版のプログラムに OpenMP の指示文を 1 行追加しただけであり, プログラミングはかなり容易になっている. CG においても, OpenMP を用いた場合はインクリメンタルに並列化していくことが可能なため, 並列化は比較的容易であった. 一方, OpenMP はスレッドへのデータのマッピングがユーザによって制御できないので, プログラムによっては十分なチューニングが行なえない場合があることがわかった.

また, OpenMP と MPI は共有/分散メモリプログラミングの API の標準として, 今後多くの計算機上で利用可能になると予想される. したがって, OpenMP と MPI を組み合わせることにより, 可搬性の高いハイブリッドプログラムの開発が可能になったといえる.

一方, SMP クラスタで HPC を行なう場合, メモリバスボトルネックは避けては通れない問題となっている. アルゴリズムを工夫するすることにより, データのより高い時間的局所性を引き出すこと, およびメモリバスの性能の改善を図ることは, SMP クラスタでは特に重要な課題である.

謝辞　本研究を行なうにあたり, 御助言・御討論頂いた新情報処理開発機構並列分散パフォーマンス研究室の関係者各位及び TEA グループのメンバー諸氏に感謝いたします. TEA グループは, 研究技術組合新情報処理開発機構・電子技術総合研究所・筑波大学を中心とする性能評価に関する研究グループである.

参 考 文 献

1) 早川 秀利, 近藤 正章, 板倉 憲一, 朴 泰祐, 佐藤 三久: "共有メモリ PC クラスタにおけるハイブリッド並列プログラムの性能評価", 情報処理学会研究報告, 99-HPC-77-23, pp131-136, 1999.

2) http://www.openmp.org/

3) http://www.myri.com/

4) 新情報処理開発機構並列分散パフォーマンス研究室, "RWCP OpenMP compiler project", http://pdplab.trc.rwcp.or.jp/Omni/home.ja.html

5) 田中 良夫, 松田 元彦, 安藤 誠, 久保田 和人, 佐藤 三久, "COMPaS: Pentium Pro を用いた SMP クラスタとその評価", 並列処理シンポジウム JSPP'98 論文集, pp.343-350, 1998.

6) Argonne National Laboratory, Mississippi State University, "MPICH - A Portable Implementation of MPI", http://www-unix.mcs.anl.gov/mpi/mpich/

7) Xavier Leroy, "The LinuxThreads library", http://pauillac.inria.fr/ xleroy/linux threads/

8) Numerical Aerospace Simulation Facility (NAS) at NASA Ames Research Center , "The NAS Parallel Benchmarks", http://www.nas.nasa.gov/Software/NPB/.

9) 板倉憲一, 松原正純, 朴泰祐, 中村宏, 中澤喜三郎, "超並列計算機 CP-PACS における NPB Kernal CG の評価", 情報処理学会論文誌, vol.39, No.6, pp.1757-1765, 1998.