ハイパフォーマンス
コンピューティング 77−23
(1999. 8. 3)

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

早川秀利† 近藤正章† 板倉憲一†
朴 泰祐† 佐藤三久††

本稿では，SMP結合された4台のIntel Pentium-II Xeonを1ノードとし，これを100base-TX Ethernetで結合したSMP-PCクラスタ *COSMO* における共有/分散メモリを考慮し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，同アーキテクチャ上で共有メモリに依存しないMPIによる分散メモリプログラミングを行なった場合について，全体的な性能と，プログラミングの容易さ等の点で比較検討する．その結果，ハイブリッドプログラミングは予想通り高い性能を引き出せるが，ノード内のバスボトルネック等の問題を十分考慮する必要があり，また同アーキテクチャにおけるMPIのみによるプログラミングでも，通信における順序性とプロセスマッピングを熟慮すれば，ある程度の性能が得られることが確認された．

Performance evaluation of hybrid parallel program on shared memory PC cluster

HIDETOSHI HAYAKAWA,† MASAAKI KONDO,†
KEN'ICHI ITAKURA,† TAISUKE BOKU† and MITSUHISA SATO††

In this paper, we describe the performance evaluation for HPC benchmarks on an SMP-PC cluster named *COSMO*. We compare two styles of programming: hybrid-programming with mixture of shared and distributed memory paradigms and message passing programming with MPI. As a result, although the hybrid programming achieves more high performance than MPI programming, we have to take care about bus-bottlenecking within a node which depends on the characteristics of the problem. For MPI programming, it is shown that a well-scheduled communication and process mapping onto nodes for optimized execution causes relatively good performance.

1. はじめに

近年，SMP (Symmetric Multi-Processor: 対称型マルチプロセッサ) 構成を用いたワークステーション (WS) やパーソナルコンピュータ (PC) が普及し，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，比較的安価で高性能なシステムとして構築可能である．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で結合したSMPクラスタ実験システムCOSMO (Cluster Of Symmetric Multi prOcessor) を使って，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．

† 筑波大学 電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba
†† 新情報処理開発機構
Real World Computing Partnership

一般に，並列プログラミングの手段として，共有メモリアーキテクチャではスレッドプログラミングが用いられ，分散メモリアーキテクチャではメッセージパッシングライブラリが用いられる．これに対してSMPクラスタは，ノード内の共有メモリアーキテクチャと，ノード間の分散メモリアーキテクチャの両方の性質を持つため，(1) メッセージパッシング統一型，(2) 共有メモリ統一型，(3) ハイブリッド型，の3通りのプログラミング方法が考えられる．

このうち，共有メモリ統一型ではDSM (分散共有メモリ) が必要となり，これにはハードウェアの支援によって行うものや，純粋にソフトウェアのみで行うものなどがあるが，前者はノード本体の低価格性に対しコストと開発の面で不利であり，後者は研究は進んでいるもののまだ実用化はされていない．

これに対し，メッセージパッシング統一型は，そもそもMPI等の豊富なプログラム資産を持ち，SMPクラスタのプログラミング法としては最も簡単である．

さらに，いくつかのMPI実装では，SMPノード上での通信が最適化されるようなバージョンが用意されている[1)2)]．ASCI Blue Mountainとして知られているSGI Origin 2000をネットワーク結合したシステムでも，多くのユーザがMPIのみを使ったプログラミングを行なっている．しかし，MPIのみを用いたプログラミングでは，SMPの性能を最大に引き出すことは困難であると考えられる．

定性的には，ハイブリッド型とメッセージパッシング統一型では，性能に関しては前者が，プログラミングの容易さに関しては後者が勝ると考えられる．そこで，我々はHPCの代表的ベンチマークで，両者を比較検討する．SMPノードにおけるHPCアプリケーションでは，ノード内のバス性能が重要になる．HPC問題ではプロセッサのキャッシュメモリは，その容量と問題サイズの関係から空間的局所性にのみ有効な場合が多く，システムの性能がメモリシステムを含むデータバスの能力に依存してしまう場合が多い．このため，ブロッキングアルゴリズム等によって，キャッシュの時間的局所性を活かして，バスへのアクセス回数を減らす工夫が必要となる．特に，Pentium系SMPシステムでは，バスの性能が低く，データの時間的局所性を引き出すことが重要である[3)]．アルゴリズムの本質によって，ブロック化ができない場合には，バスの競合を解消するために，ノード内での使用プロセッサ数を抑え，並列性を広くノード間に分散した方が効率が良い場合も考えられる．

本稿では，COSMOの概略とその性能，ハイブリッドプログラミングの概要，COSMO上でのハイブリッドプログラミングとメッセージパッシング統一型プログラミングの比較評価に関して報告する．

2. 実験環境

2.1 COSMOの仕様

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 PowerEdge 6300 (450NX chip-set, 主記憶512MB) である．全体では3ノード構成であり，ノード間は100base-TX Ethernet Switch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．

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2つである☆．

- **MPI:** MPIライブラリ (MPICH-1.1.2[1)]) がインストールされている．MPICHの下位の通信方法としてはCH_P4というTCP/IPの通信を行う方法を選択してある．(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 (Posix準拠のスレッドライブラリ)[5)]を使用できる．LinuxThreadsはカーネルレベルスレッドである．

ここで，ノード間及びノード内でのMPI通信に要する時間の目安として，ノード内でのスレッド間のデータ交換に要する時間との比較した．MPI通信では，2プロセス間でのピンポン転送を行ない，pthreadsでは，2つのスレッドが共有するバッファのデータを相互に更新するという処理を繰り返した．その結果，メッセージ長に関わらず，pthreadsでは約70MB/sec，ノード内MPI通信は約18MB/sec，ノード間MPI通信は約8.6MB/secのスループットがあることがわかった．ノード内でのMPI通信は，ノード間のそれに比べ高速であるが，やはり共有メモリを積極的に利用したスレッドプログラミングに比べ，大きく性能を低下させる可能性がある．

2.2 ハイブリッドプログラミング

ノード内ではpthreadsライブラリを用いた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い，ノード間ではMPIライブラリ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を行なう．プログラムの概念としては，MPIで通信し合う複数のノード上の各プロセスが，さらに各々のノード上でマルチスレッド化されていると理解すれば良い．

スレッド間でのMPI通信については，各スレッドが各々独自に他ノードのスレッドと通信を行なうスタイルも考えられる．しかし，このためにはMPI通信のリソース管理と通信ブロッキングをマルチスレッド環境下で安全に処理するthread-safeなMPIが必要である．さらに，このように通信を分散させることは処理の複雑化を招き，さらに通信性能が向上するとは考えにくい．まず，MPI通信の関数呼び出し回数が総合的に増え，オーバヘッドが増加する可能性が高くなり，またMPIをthread-safeにすることによるオーバヘッドの増加も考えられる．さらに，一般的にはデータを生成したスレッドがその送出を担当するが，いずれにせよデータはキャッシュから一旦メモリにフラッシュされ，NIAによってDMA転送されるため，キャッシュ内容を保持しているスレッドが通信を起動したからといって利点が生じるとは考え難い．よって，我々はMPI通信をマスタースレッドからのみ行なうというスタイルに統一する．

3. 性能評価

以下の2種類のベンチマークを用いて，ハイブリッドプログラムとメッセージパッシングプログラムを比較評価する．

☆ この他に，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[4)]のテスト版が利用可能であるが，本稿では取り挙げない．

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 とし，一部 2000 × 2000 も扱う．

CG NAS Parallel Benchmarks version 1 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局所性を引き出せる．アルゴリズムは外積型とした．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．

MPI プログラムの評価に際しては 1，2，3 の 3 通りのノード数に関し，各ノード数毎に 1 ノード内のプロセス数を 1，2，3，4 の 4 通りに変化させて実行時間を計測し，速度向上比を求めた．総プロセス数が，実際に稼働するプロセッサ数となる．MPI のプロセスは，使用するノードに対しサイクリックに割り当てる．以下では，分散メモリプログラムを full-MPI 版と略す．

ハイブリッド並列プログラムの評価に際してはノード数を 1，2，3 の 3 通りとし，各ノード数毎に 1 プロセス内のスレッド数を 1，2，3，4 の 4 通りに変化させて実行時間を計測し，速度向上比を求めた．ノード数とスレッド数の積が，実際に稼働するプロセッサ数となる．以下では，ハイブリッド並列プログラムを hybrid 版と略す．

3.1 Linpack における性能評価

full-MPI 版の評価

図 1に Linpack の行列サイズ 1000 × 1000 における full-MPI 版の速度向上比を示す．1 ノードの場合はプロセッサ数の増加に伴い性能が単調増加しているが，2 ノード及び 3 ノード時ではスケーラビリティが低く，かえって性能が低下している．ノード外への通信が入ることにより通信オーバヘッドが増すことが基本的要因と思われる．また，Linpack は各プロセスが自分の担当消去行を計算した後，全プロセスへ行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も理由と思われる．

hybrid 版の評価

図 2に Linpack の行列サイズ 1000 × 1000 における hybrid 版の速度向上比を示す．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．プロセッサ数の増加に伴って計算部分にかかる時間が

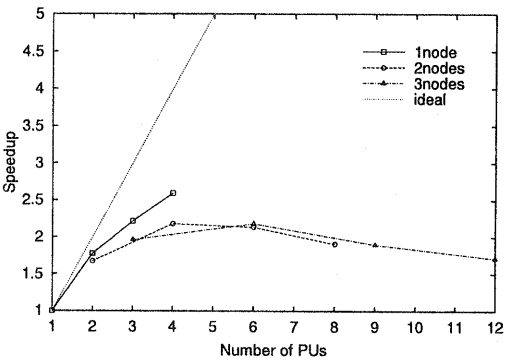

図 1 Linpack full-MPI 版：1000x1000 の速度向上比

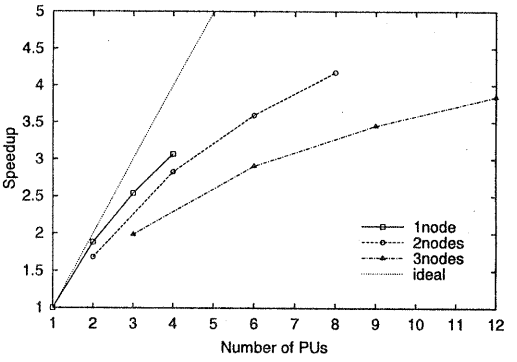

図 2 Linpack hybrid 版: 1000x1000 の速度向上比

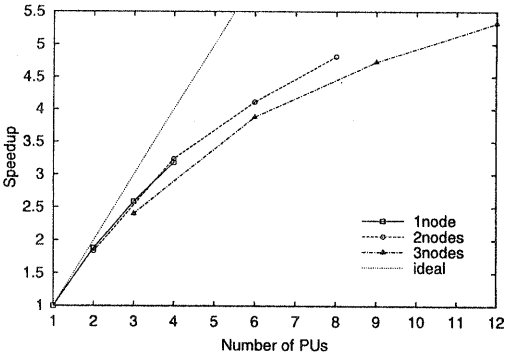

図 3 Linpack hybrid 版: 2000x2000 の速度向上比

短縮されることは full-MPI 版と同じであるが，消去行を全プロセスへブロードキャストする部分で，hybrid 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 版に比べ通信回数及び通信量が低減される．この影響から，全てのノード数でプロセッサ数の増加に伴って性能が向上する結果となった．

また，4 スレッド × 3 ノード = 12 プロセッサ の性能が，4 スレッド × 2 ノード = 8 プロセッサ の性能よ

り劣っている．これは，計算の粒度が十分大きくないためと考えられる．そこで，図3にLinpackの行列サイズ2000×2000におけるhybrid版の速度向上比を示した．予想通り，計算粒度が大きくなったことにより4スレッド×3ノード時の性能が4スレッド×2ノード時の性能を上回っている．

3.2 CGによる性能評価

full-MPI版の評価

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[6]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ている．

図4及び5に，CGのデータサイズClass-A，Class-Bにおけるfull-MPI版の速度向上比をそれぞれ示す．どちらの問題サイズに対しても，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い．この理由は，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思われる．

また，問題サイズが小さい程，各プロセッサの計算量が少なくなり，さらに割り当て領域サイズの減少によりキャッシュヒット率が高くなるため，メモリバスの混雑が低減するという相乗効果により，通信オーバヘッドがより顕在化することが予想される．そこで，プロセッサ内部処理時間のほとんどを占める行列ベクトル積(MVM)と，プロセッサ間通信時間のほとんどを占めるベクトルのコレクション(VC)に要する時間を求めた．1ノードの場合を表1に示す．

表1　CGにおける1ノード時の処理時間の内訳(sec.)

プロセッサ数	Class-A		Class-B	
	MVM	VC	MVM	VC
1	71.74	—	3959	—
2	47.60	1.52	2684	38.50
3	32.70	15.51	2437	134.50
4	32.25	30.02	2438	180.25

これからわかるように，問題サイズの小さいClass-Aのほうが，通信オーバヘッドが大きく効いており，それが原因でスケーラビリティが落ちていることがわかる．なお，どちらの場合も，プロセッサ数が2から3に増える際に，CVの時間が大幅に増加している．これに関しては調査中であるが，MPI関数におけるリダクション通信がノード内通信でどのように実装されているかが関係しているものと思われる．

full-MPI版のプログラムでは，通信の際，MPI_Bcast関数やMPI_Allreduce関数を使用した．しかし，その代わりにMPI_Send/MPI_Recv関数を多用し，SMPクラスタの構成に合わせて通信順序を工夫すれば性能

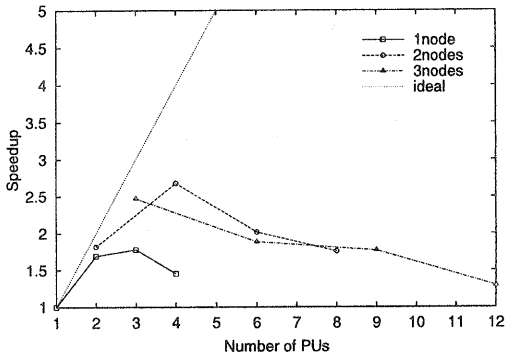

図4　CG full-MPI版: Class-Aの速度向上比

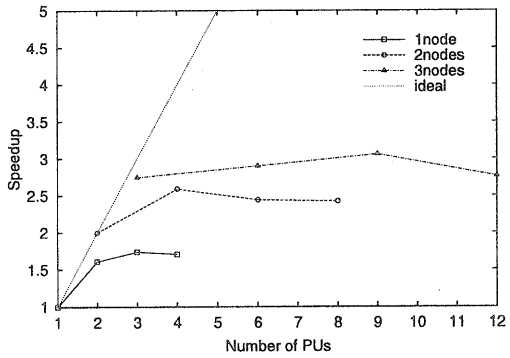

図5　CG full-MPI版: Class-Bの速度向上比

向上の余地があると考えられる．これについては3.3節に述べる．

hybrid版の評価

図6及び7に，CGのデータサイズClass-A，Class-Bにおけるhybrid版の速度向上比をそれぞれ示す．どちらの問題サイズに対してもfull-MPI版と比較して格段に性能が良いことがわかる．hybrid版ではノード数＝プロセス数であるため3ノード時でも高々3プロセス間の通信だけが必要となり，通信回数を大幅に抑えることができる．特にノード内での通信が無いことにより，ノード内プロセッサ数に対するスケーラビリティが大きく改善されている．

CGのClass-Bでは，Class-Aに比べ，キャッシュヒット率が低減すると思われる．1次元ブロッキング手法によるNPB CGでは，疎行列データに対するアクセスはメモリ番地を逐次スキャンする代わりに再利用性がなく，これに対し，乗数ベクトルへのアクセスはほぼランダムになるが高い再利用性がある．そして，疎行列データに関しては，どちらの問題サイズでも512KBの2次キャッシュ内に行列データを保持することは不可能である(12プロセッサの場合でも)が，ベクトルデータに関しては，Class-Aでは2次キャッシュ内に収まり，Class-Bでは収まらない．この差により，

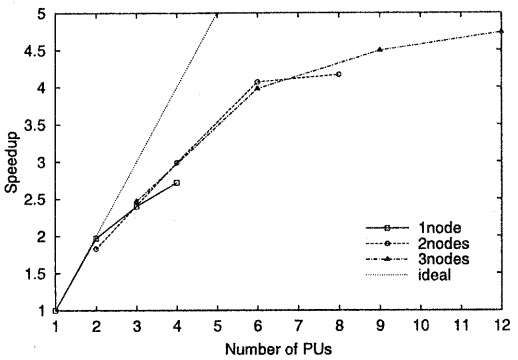

図6　CG hybrid 版: Class-A の速度向上比

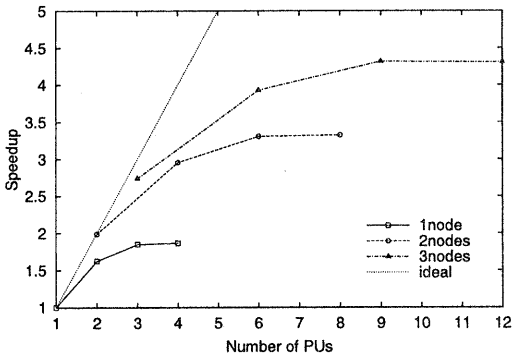

図7　CG hybrid 版: Class-B の速度向上比

結果的に Class-B のキャッシュヒット率は Class-A よりも落ちるため，ノード内プロセッサ数に対するスケーラビリティが低くなるのだと考えられる．また注目すべきは，例えば3プロセッサを用いる場合の，1ノードと3ノードの関係，あるいは4プロセッサを用いる場合の1ノードと2ノードの関係である．図6では，これらは各々，オーバラップしており，どの構成の場合でも，同じプロセッサ数を用いた場合の性能はほぼ等しい．これに対し，図7では，それぞれノード数が多い場合の方が性能が高く，これは6プロセッサの場合にも同様である．これもキャッシュヒット率と関係があると考えられる．同数のプロセッサを用いた場合，ノード内プロセッサを増やした時のバスボトルネックの影響が Class-B の方で顕著に現れるため，ノード数を増やした方が性能が内部処理時間に関しては有利である．さらに，プロセッサ内部処理時間に対するノード間通信のコストも，Class-B の方が Class-A よりも相対的に小さいため，このような結果になったと思われる．この結果を裏づけるように，ノード内プロセッサ数を増やした場合の性能の延び悩みに関しては，Class-B の方が顕著に現われている．

このように，同一プロセッサ数が利用可能な場合，ノード間にそれをどのように分配すべきかに関しては，

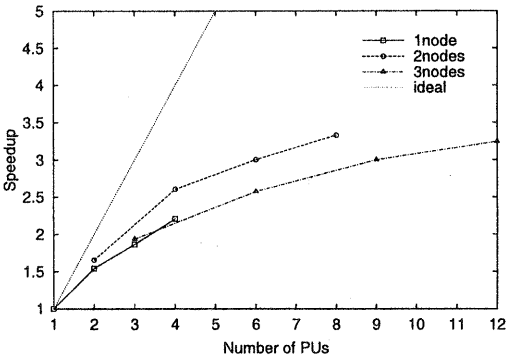

図8　CG full-MPI 版 (通信順序を最適化): Class-A

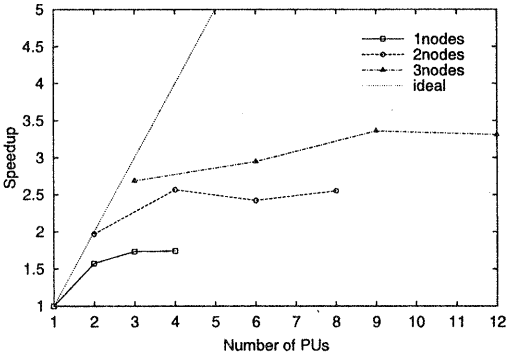

図9　CG full-MPI 版 (通信順序を最適化): Class-B

必ずしもノード内通信の低コスト性だけが指標にはならず，キャッシュヒット率に代表される，ノード内処理のスケーラビリティ低下要因を十分に考慮する必要があることがわかる．

3.3　通信順序を最適化した full-MPI 版の評価

これまでの full-MPI 版では，プロセスのノードへの割り当てをサイクリックに行ない，さらに通信に関しては，MPI によって提供されるリダクション通信等をそのまま用いてきた．しかし，ノード内の MPI 通信がノード間に比べ軽いことを考えると，リダクションのアルゴリズムや，通信順序を含めた SMP クラスタ向けの最適化が考えられる．図8及び9は，それぞれ Class-A 及び Class-B の full-MPI 版の CG において，プロセス間の通信順序を最適化した場合の速度向上比である．通信順序の最適化とは，上述の full-MPI 版のプログラムで使用した MPI_Bcast 関数や MPI_Allreduce 関数の代わりに MPI_Send 関数及び MPI_Recv 関数を使用し，プロセス間の通信順序を詳細に記述することにより，ノード内通信の際に起こるバス上でのブロッキングを低減させるように工夫するという意味である．

どちらの問題サイズに対しても，前述の full-MPI 版と比較して全体的に性能が改善されているが，hybrid

版の性能には及ばない．しかし，このような2レベルの通信コストを考慮することは重要である．例えば，SMP クラスタ用にチューニングされた MPI リダクション関数を用意することにより，かなりの性能改善が得られる可能性がある．

また，Linpack に関しては，今回行なったサイクリック分割による full-MPI 版は通信順序の点で不利である．各プロセスは割り当てられたブロックの更新データを計算し，これを全プロセスにブロードキャストする．MPI の標準のブロードキャストは，rank の低いプロセスから順番にデータを送信する．Linpack のアルゴリズムから考えると，現在消去された行に近い行(例えば直下の行)を担当するプロセス，になるべく早くデータを送ることにより，全体のパイプライン処理をスムーズに行なうことが可能となる．ところが，今回のようなサイクリック割り当てを行なっていると，直下行を担当するプロセスは常に他ノードに存在することになり，これが性能低下の要因となり得る．試みにこのようなデータ送信の順序性を考慮したプロセス割り当てを行なったところ，最大で約 1.5 倍程度の性能向上が確認された．

3.4 プログラミング容易性

一般にハイブリッドプログラミングは，複数のパラダイムを考慮しなければならず，メッセージパッシング統一型に比べ複雑になり易い．しかし，性能としては，ハイブリッドプログラミングは，メッセージパッシング統一型プログラミングに比べプロセス間通信の回数を抑えられることにより性能が高く，特にノード内 PU 数に対するスケーラビリティが高い．従って，計算量に比べ通信時間に占める比重が高いアプリケーションほど，両者の性能差は大きくなり，ハイブリッドプログラミングの相対性能が高くなることがわかった．

一方，メッセージパッシング統一型はその単純さに魅力があるが，SMP クラスタで用いる場合，プロセスのノードへの割り当てや，プロセス間通信の際の通信順序等に配慮しないと，その性能が大きく低下することが確認された．場合によっては，既存の MPI プログラムを SMP クラスタで実行させることにほとんど魅力が生じない可能性もあり得るが，逆に通信順序その他を意識したプログラムを書き始めると，ハイブリッド型ほどではないがかなり複雑な処理を要求され，結局ハイブリッド型に対する対プログラミングコスト性能比は良くない．この問題を解決する一つの鍵は，プロセスのノードへの割り当てと通信順序を考慮した，SMP クラスタ向けの MPI 通信ライブラリの提供ではないだろうか．

4. おわりに

最近の SMP クラスタ事情を反映し，SMP を意識した MPI 実装等により，ハイブリッドプログラミングとメッセージパッシング統一型プログラミングの性能差は縮まりつつある．しかし，現段階において SMP クラスタのアーキテクチャの利点を十分に引き出し高性能を追求するためには，やはりハイブリッドプログラミングは不可欠である．

MPI は MPI2 に示されるように，メッセージパッシングライブラリの標準として今後も利用されるであろう．一方 SMP プログラミングとしては今後は OpenMP が一つの標準になると期待されている．今回の実験を発展させ，今後は OpenMP+MPI のハイブリッドプログラミングにおける性能評価を行なっていく予定である．また，SMP あるいはそのクラスタにより特化された MPI 実装に関しても研究していく予定である．

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

2) LAM team / The Laboratory for Scientific Computing at the University of Notre Dame, "LAM / MPI Parallel Computing", http://www.mpi.nd.edu/lam/

3) 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 年 6 月

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.rwcp.or.jp/pdperf/Omni/home.ja.html

5) Xavier Leroy, "The LinuxThreads library", http://pauillac.inria.fr/ xleroy/linuxthreads/

6) 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 年 6 月.