メモリバスアクセス率に基づくSMP-PCクラスタの性能評価

板倉憲一[†1] 早川秀利[†2,☆] 近藤正章[†2,☆☆]
吉川茂洋[†2] 朴泰祐[†2]
佐藤三久[†3] 田中良夫[†4]

本論文では，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*の性能評価を典型的なHPCベンチマークによって行う．さらに，SMP-PCクラスタの性能に大きな影響を与えるノード内のメモリバスアクセス率を測定し，性能との関係を調べる．また，メモリアーキテクチャを考慮したハイブリッドプログラミングとMPIによる分散メモリプログラミングの2つのプログラミング方法において，性能とメモリバスアクセス率を実測し，プログラミング方法と性能の関係を明確にする．結果として，各プログラミング方法によるベンチマークプログラムのスケーラビリティの限界となった原因を示すことができた．

Performance Evaluation of SMP-PC Cluster Based on Memory Bus Access Ratio

Ken'ichi Itakura,[†1] Hidetoshi Hayakawa,[†2,☆]
Masaaki Kondo,[†2,☆☆] Shigehiro Yoshikawa,[†2] Taisuke Boku,[†2]
Mitsuhisa Sato[†3] and Yoshio Tanaka[†4]

In this paper, we describe the performance evaluation for HPC benchmarks on an SMP-PC cluster named *COSMO* and measure the memory bus access ratio which is influential with the performance on the SMP-PC cluster. 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, we show the bottleneck of the performance scalability for each benchmark program and programming style.

1. はじめに

歴史的な経緯から，現在のHPCの分野の並列プログラムにはメッセージパッシング方式が多く用いられている．これは，実用となった並列計算機の多くが分散メモリ型のアーキテクチャであること，シンプルな実装で並列化のオーバヘッドが読みやすく性能チューニングが容易であったこと等が理由としてあげられる．このような，メッセージパッシング方式への移行は，初期のNAS Parallel Benchmarks[1]が数値計算の問題とその解法のみが示され，特定のプログラムコードが指定されていなかったのに対して，Version 2[2]では，事実上標準となったMPI[3]によるプログラムコードが指定されたベンチマークとなったことからも伺える．

これに対して，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．これは，単に小規模な並列計算機というだけでなく，たとえば米国のASCIプロジェクトにおいてもクラスタ化した超並列計算機はHPCの有効なプラットフォームであること

†1 筑波大学計算物理学研究センター
Center for Computational Physics, University of Tsukuba
†2 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba
☆ 現在，富士ソフトABC株式会社
Presently with FUJISOFT ABC Inc.
☆☆ 現在，東京大学先端科学技術研究センター
Presently with Research Center for Advanced Science and Technology, University of Tokyo
†3 新情報処理開発機構
Real World Computing Partnership
†4 電子技術総合研究所
Electrotechnical Laboratory

が示されている．

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 SMP クラスタでは，スレッドプログラミングの中でメッセージパッシングライブラリを用いるハイブリッドなプログラミングが最も性能を引き出すのに有効であると考えられる．これに対し，メッセージパッシングの API として MPI は広く用いられ，その豊富なプログラム資産がある．また，ASCI Blue Mountain[4] 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っている．

定性的には，性能に関してはハイブリッド型が有利であり，プログラミングの容易さではメッセージパッシング統一型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 アプリケーションにおいて，プロセッサのキャッシュメモリは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，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[5]．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．

文献 5) では，SMP-PC 上でのマルチスレッドによる並列プログラミング性能に対し簡単なバスアクセスのベンチマークにより定性的傾向を示していた．しかし，より高度な性能解析およびチューニングのためには，プログラムの細部にわたるメモリバスボトルネックの定量的解析が必要になると考えられる．そこで，本論文では，ハイブリッドプログラミングと MPI のみのプログラミングの両者を対象に，メモリバスアクセス率の定量的評価を基にした性能評価を行い，両者の性能差の原因を明らかにすることを目的とする．

2. 実験環境

2.1 COSMO の仕様

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．

COSMO の各ノードは Intel Pentium-II Xeon（450 MHz，16 KB 4way L1 データキャッシュ，512 KB 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 512 MB）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 で接続されている．また，OS は，SMP 対応の Linux 2.2.10 を利用している．

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[6]）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）[7] 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，通信オーバヘッドの目安を示すために，ノード間およびノード内での MPI 通信とノード内でのスレッド間のデータ交換に要する時間を比較した（**図 1**）．MPI 通信では 2 プロセス間でのピンポン転送を行い，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．測定結果より，HPC アプリケーションに多く見られる比較的長いメッセージ長（数 KB 以上）において，Pthreads では約 70 MB/sec，ノード内 MPI 通信は約 18 MB/sec，ノード間 MPI 通信は約 8.6 MB/sec のスループットがあることが分かった．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，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

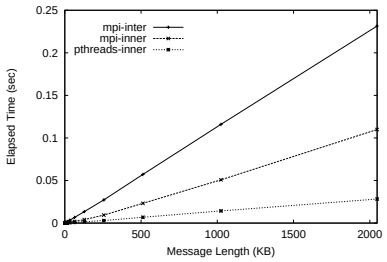

図 1 基本データ転送性能
Fig. 1 Basic data transfer performance.

に比べると，性能を低下させる要因となる．

2.2 ハイブリッドプログラミング

ハイブリッドプログラミングにおいては，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，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よい．

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 通信ライブラリが必要である．また，通信を分散させることは処理の複雑化を招く．さらに，個々のスレッドが MPI 通信を呼び出す場合は，代表スレッドどうしでデータをまとめて通信する場合に比べて粒度の細かい通信となる．このために，関数呼び出し回数が増え，オーバヘッドが増加する可能性が高い．さらに，データはキャッシュからいったんメモリにフラッシュされた後に NIA によって DMA 転送されるために，キャッシュにデータ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うというスタイルに統一する．

2.3 バスアクセス率

Intel の Pentium-II Xeon には，プロセッサのパフォーマンスを測定するためのカウンタが用意されている．このカウンタを使用し，プロセッサのデータ系のメモリ参照回数（DATA_MEM_REFS）とすべてのバストランザクション回数（BUS_TRAN_ANY）を測定する．そして，この値を基にバスアクセス率を BUS_TRAN_ANY/DATA_MEM_REFS として定義する．I/O を含むすべてのバスアクセスの回数がプログラムが実行時間に大きな影響を及ぼすのに対し，データ系のメモリ参照回数はプログラムのソースレベルから予想できる値であり，アルゴリズムが同一であれば一定であると考えられる．このために，このバスアクセス率によって同一のアルゴリズムに対するプログラミング方法の評価が行える．

さらに，プロセッサの動作周波数の単位で測定可能なクロックカウンタの値を同時に測定し，単位時間あたりのメモリ参照回数やキャッシュミスによるバスのアクセス回数を得ることができる．これらの値を指標として各アプリケーションとそのプログラミング方法を比較する．

3. 性能評価

3.1 アプリケーションプログラム

本論文では，以下の 2 種類のベンチマークを用いて性能を比較評価する．

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[8]．行列サイズは 1000×1000，2000×2000，3000×3000 の 3 種類を用いる．

Kernel CG NAS Parallel Benchmarks version 1[1] の Kernel CG．正値対称な大規模疎行列の最小固有値を CG（Conjugate Gradient）法によって求める問題．問題サイズは Class-A と Class-B を用いる．

Linpack は外積型のアルゴリズムを使用し，キャッシュブロッキングを施すことによって，データ参照の時間的局所性の高いプログラムの作成が可能となる．一方 Kernel CG では，反復計算中の最も処理時間のかかる疎行列とベクトルの積の計算において，大規模なワーキングセットとなる疎行列の各要素は各々一度だけ読み出される．このため，Kernel CG のプログラムではデータ参照の時間的局所性が低い．このように，データの局所性の有無に特徴のある 2 つのアプリケーションプログラムを評価対象として選んだ．

なお，NPB Kernel CG はプロセッサ数が多いときに疎行列を 2 次元に block-block 分割するマッピングによってプロセス間通信時間を短縮できる[9]が，今回はプロセッサ数が最大 16 までなので疎行列を単純に 1 次元 block 分割してマッピングする方法を選択した．

MPI プログラムの評価に際しては 1，2，3，4 の 4 通りのノード数に関して，各ノード数ごとに 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 1，2，3，4 の 4 通りとして，各ノード数ごとに MPI の 1 プロセスを起動する．そして各プロセスのスレッド数を 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．

表 1 各ベンチマークの逐次処理時間
Table 1 Total processing time by the sequential version.

ベンチマーク	処理時間（sec）
Linpack 1000 × 1000	9.15
Linpack 2000 × 2000	79.13
Linpack 3000 × 3000	289.54
NPB kernel CG Class A	82.76
NPB kernel CG Class B	4146.94

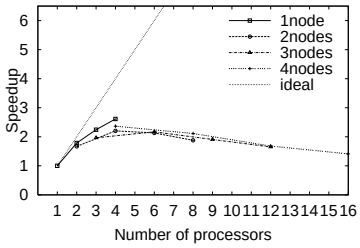

図 2 速度向上比（Linpack 1000 × 1000，full-MPI 版）
Fig. 2 Speedup ratio (Linpack 1000 × 1000, full-MPI).

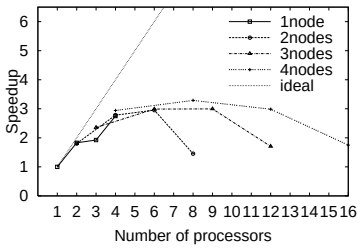

図 3 速度向上比（Linpack 2000 × 2000，full-MPI 版）
Fig. 3 Speedup ratio (Linpack 2000 × 2000, full-MPI).

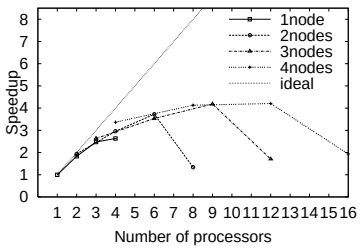

図 4 速度向上比（Linpack 3000 × 3000，full-MPI 版）
Fig. 4 Speedup ratio (Linpack 3000 × 3000, full-MPI).

性能評価に使用したプログラムは，MPI+Pthreads で記述されており，どのような構成にも対応できるようになっている．たとえば，すべての MPI プロセスを 1 スレッドで動かせば full-MPI となり，1 つの MPI プロセスで複数スレッドを動かすことも可能である．この場合はハイブリッドではない単なるマルチスレッドであり，1 ノードでのみ実行可能である．性能評価は 1 プロセス・1 スレッド，すなわち逐次処理に対する並列処理の速度向上比として表す．基準となる逐次処理の時間を**表 1** に示す．

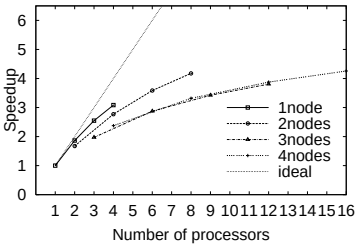

図 5 速度向上比（Linpack 1000 × 1000，hybrid 版）
Fig. 5 Speedup ratio (Linpack 1000 × 1000, hybrid).

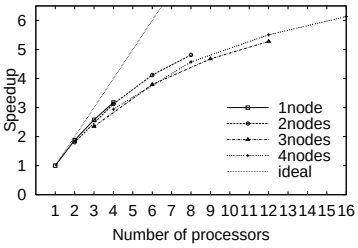

図 6 速度向上比（Linpack 2000 × 2000，hybrid 版）
Fig. 6 Speedup ratio (Linpack 2000 × 2000, hybrid).

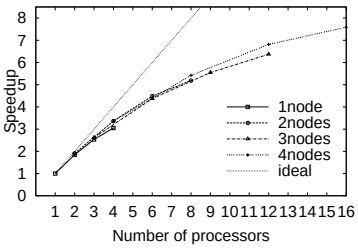

図 7 速度向上比（Linpack 3000 × 3000，hybrid 版）
Fig. 7 Speedup ratio (Linpack 3000 × 3000, hybrid).

3.2 Linpack

まず，**図 2〜4** に，Linpack の full-MPI 版の速度向上比を示す．この結果より，full-MPI 版の Linpack はプロセッサ台数の増加に対して性能のスケーラビリティが悪いことが分かる．特に，2000×2000，3000×3000 のノード内で 4 プロセッサ使用している場合には極端な性能低下が見られるが，この理由はメモリバスボトルネックである．実際のバスアクセス率を測定した結果を**表 2** に示す．この表から，確かにノード内で 4 プロセッサ使用時に高いバスアクセス率が起こっていることが分かる．これは，MPICH の実装では計算用のプロセスのほかに通信管理用のプロセスを起動するために，計算プロセスがマイグレーションを起こしてキャッシュミスが増加しているのが原因である．実際

表 2 各ベンチマークの全体的なメモリバスアクセス率（%）
Table 2 Memory bus access ratio (%).

#n	#p	Linpack						Kernel CG			
		1000 × 1000		2000 × 2000		3000 × 3000		Kernel A		Kernel B	
		full	hybrid	full	hybrid	full	hybrid	full	hybrid	full	hybrid
1	1	1.24	1.24	1.36	1.37	1.56	2.64	2.50	2.50	5.49	5.76
1	2	1.22	1.17	1.55	1.34	1.68	2.51	2.58	2.53	6.01	5.58
1	3	1.23	1.07	1.35	1.30	1.80	2.57	2.67	2.55	5.64	5.58
1	4	1.22	0.99	8.82	1.28	2.04	2.43	2.38	2.57	5.79	5.63
2	2	1.21	1.21	1.39	1.39	1.59	2.10	2.50	2.51	5.79	5.67
2	4	1.29	0.87	1.57	1.18	2.03	1.72	2.11	2.53	5.59	5.65
2	6	1.12	0.64	1.34	0.99	1.68	1.51	2.37	2.55	5.60	5.75
2	8	1.29	0.49	8.32	0.87	5.18	1.31	1.92	2.57	5.72	5.67
3	3	1.18	1.18	1.38	1.39	1.59	2.32	2.42	2.42	5.49	5.56
3	6	1.07	0.66	1.53	0.98	1.92	1.65	2.11	2.48	5.41	5.58
3	9	1.01	0.43	1.33	0.74	1.59	1.20	1.75	2.52	5.56	5.71
3	12	1.31	0.28	7.76	0.60	5.81	1.13	1.51	2.54	5.27	5.68
4	4	1.15	1.15	1.39	1.39	1.59	2.09	2.37	2.34	5.40	5.55
4	8	1.04	0.58	1.50	0.88	1.91	1.44	1.68	2.41	5.52	5.44
4	12	0.96	0.33	1.28	0.65	1.57	1.06	1.36	2.50	5.01	5.53
4	16	1.27	0.22	7.66	0.52	7.32	0.88	1.17	2.51	5.05	5.57

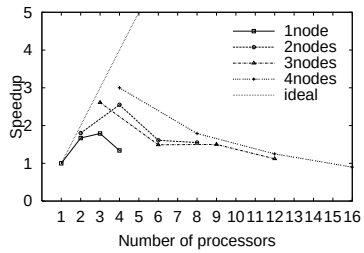

図 8 速度向上比（Kernel CG Class A，full-MPI 版）
Fig. 8 Speedup ratio (Kernel CG Class A, full-MPI).

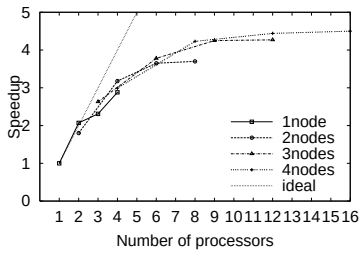

図 10 速度向上比（Kernel CG Class A，hybrid 版）
Fig. 10 Speedup ratio (Kernel CG Class A, hybrid).

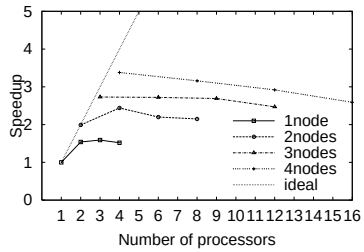

図 9 速度向上比（Kernel CG Class B，full-MPI 版）
Fig. 9 Speedup ratio (Kernel CG Class B, full-MPI).

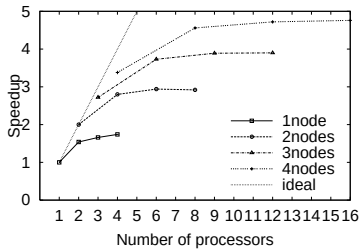

図 11 速度向上比（Kernel CG Class B，hybrid 版）
Fig. 11 Speedup ratio (Kernel CG Class B, hybrid).

にバスアクセス率を測定するときにプロセスを実行しているプロセッサ ID を取得しているが，ノード内 3 プロセッサ使用まではマイグレーションがほとんどが起こらず，4 プロセッサ使用時には多く起こることを確認した．

次に，**図 5〜7** に，Linpack の hybrid 版の速度向上比を示す．この結果より，hybrid 版は，full-MPI 版に比べすべての場合において性能が向上することが確認できる．また，hybrid 版は良いスケーラビリティを示すことが分かる．バスアクセス率（表 2）を full-MPI 版と比較すると明らかにバスアクセス率が低下しており，特にノード内のプロセッサ台数が多いときにこの差が大きく，性能向上の要因となっている．さらに，2000×2000，3000×3000 の場合には，使用するノード数・スレッド数の構成にかかわらず総プロセッサ数にのみ性能が依存している．これは，ノード間通信の粒度が大きくボトルネックとならないためである．また，使用する通信パターンがブロードキャストという

表 3 バスアクセス率の内訳（Kernel CG Class A，full-MPI 版）
Table 3 Breakdown of the memory bus access ratio (Kernel CG Class A, full-MPI).

		Matrix-Vector Multipy				Others			
#n	#p	Mem_Ref	Bus_Acs	(%)	(clock)	Mem_Ref	Bus_Acs	(%)	(clock)
1	1	28462087	729861	2.56	1.28e+08	577879	13171	2.28	2.39e+06
1	2	14236918	365803	2.57	7.10e+07	344862	9335	2.71	2.95e+06
1	3	9506279	249466	2.62	5.08e+07	229737	6727	2.93	3.98e+06
1	4	7173299	197409	2.75	4.29e+07	207364	5588	2.69	4.13e+06
2	2	14241957	370726	2.60	6.38e+07	580160	9050	1.56	5.91e+06
2	4	7172343	190972	2.66	3.64e+07	208502	5968	2.86	6.38e+06
2	6	4421538	129118	2.92	2.09e+07	157120	4486	2.86	7.50e+06
2	8	3581450	96185	2.69	2.07e+07	160993	4727	2.94	9.27e+06
3	3	9506724	249729	2.63	4.26e+07	503315	6286	1.25	1.03e+07
3	6	4357655	140088	3.21	1.76e+07	157288	5329	3.39	1.22e+07
3	9	2926311	92180	3.15	1.36e+07	139590	5575	3.99	1.37e+07
3	12	2380130	69766	2.93	1.35e+07	291840	5177	1.77	1.94e+07
4	4	7173671	187617	2.62	3.21e+07	493792	4026	0.82	9.67e+06
4	8	3581195	97888	2.73	1.82e+07	161605	4026	2.49	1.25e+07
4	12	2379825	66210	2.78	1.20e+07	296611	4290	1.45	1.53e+07
4	16	1622629	51973	3.20	6.87e+06	159146	4212	2.65	1.94e+07

表 4 バスアクセス率の内訳（Kernel CG Class B，full-MPI 版）
Table 4 Breakdown of the memory bus access ratio (Kernel CG Class B, full-MPI).

		Matrix-Vector Multipy				Others			
#n	#p	Mem_Ref	Bus_Acs	(%)	(clock)	Mem_Ref	Bus_Acs	(%)	(clock)
1	1	217411129	12329328	5.67	1.32e+09	3155977	208277	6.60	1.90e+07
1	2	108534836	6244786	5.75	7.69e+08	59531685	113971	0.19	6.58e+08
1	3	72452337	4381321	6.05	4.49e+08	33264787	83417	0.25	9.05e+08
1	4	54597506	3196964	5.86	4.04e+08	15782824	70873	0.45	6.38e+08
2	2	108516227	6080380	5.60	6.22e+08	3269588	108341	3.31	6.90e+07
2	4	54593137	3156696	5.78	3.31e+08	23009786	69847	0.30	3.70e+08
2	6	34498771	1937285	5.62	1.69e+08	11110635	62438	0.56	3.84e+08
2	8	26523137	1620650	6.11	1.45e+08	9100195	61429	0.68	4.85e+08
3	3	72421965	4050354	5.59	4.36e+08	37117281	78697	0.21	5.03+08
3	6	34439272	1944501	5.65	1.78e+08	11802915	64834	0.55	5.95+08
3	9	23553090	1444872	6.13	1.28e+08	7969862	58378	0.73	4.55+08
3	12	18126227	1064695	5.87	1.06e+08	6548357	61059	0.93	7.56+08
4	4	54585484	3031815	5.55	3.26e+08	24281169	67600	0.28	3.84+08
4	8	26516946	1560676	5.89	1.42e+08	9019284	59026	0.65	5.03+08
4	12	18149765	1058502	5.83	1.10e+08	6760390	58911	0.87	6.41+08
4	16	13264764	808926	6.10	7.16e+07	4352216	56916	1.31	6.05+08

一方向通信なので，受信したプロセスは他のプロセスの状態に関係なく処理が続行できる．これが並列性を活かせる点も要因にもなっている．

3.3 Kernel CG

まず，**図 8**，**9** に Kernel CG の full-MPI 版の速度向上比を示す．この結果から，ノード内で 1 プロセッサのみを使った場合のスケーラビリティは良いことが分かる．しかし，ノード内のプロセッサ数を増やすと性能低下を起こしている．バスアクセス率（表 2）を Linpack と比較すると，キャッシュミスによる多くのバスアクセスを必要することが分かる．このために，ノード内のプロセッサ数を増やしたときにバスボトルネックが生じ性能低下を引き起こしている．

次に，**図 10**，**11** に Kernel CG の hybrid 版の速度向上比を示す．明らかに full-MPI 版よりも良い性能でありスケーラビリティも改善されている．しかし，バスアクセス率（表 2）を full-MPI 版と hybrid 版で比較しても有意な差は認められない．そこで，Kernel CG のバスアクセス率をさらに細かく測定した．その結果を**表 3〜6** に示す．

ここでは CG 法の反復計算部分を疎行列・ベクトル積とそれ以外（DAXPY 計算，通信をともなう内積計算，ベクトルの Collection 通信）に分けてバスアクセス率と時間を計測した．表 3〜6 より，高いバス

表 5 バスアクセス率の内訳（Kernel CG Class A，hybrid 版）
Table 5 Breakdown of the memory bus access ratio (Kernel CG Class A, hybrid).

		Matrix-Vector Multipy				Others			
#n	#p	Mem_Ref	Bus_Acs	(%)	(clock)	Mem_Ref	Bus_Acs	(%)	(clock)
1	1	28460917	736951	2.59	1.27e+08	576844	13779	2.39	2.44e+06
1	2	14232546	370832	2.61	7.15e+07	518816	17698	3.41	2.50e+06
1	3	9514002	247629	2.60	5.12e+07	582358	18411	3.16	2.78e+06
1	4	7184877	185197	2.58	4.08e+07	527440	17641	3.34	2.51e+06
2	2	14234266	358660	2.52	6.40e+07	583333	8053	1.38	4.93e+06
2	4	7168162	186226	2.60	3.63e+07	547786	11155	2.04	5.20e+06
2	6	4789887	124906	2.61	2.33e+07	559591	11733	2.10	5.05e+06
2	8	3584029	98530	2.75	3.33e+07	544475	12196	2.24	5.31e+06
3	3	9503668	264023	2.78	4.34e+07	498939	7119	1.43	1.03+e07
3	6	4793677	127066	2.65	2.19e+07	512565	8713	1.70	1.07+e07
3	9	3188619	86118	2.70	1.62e+07	501014	9892	1.97	1.05+e07
3	12	2378422	68591	2.88	1.45e+07	516759	9707	1.88	1.04+e07
4	4	7172305	185560	2.59	3.20e+07	490105	5527	1.13	9.99+e06
4	8	3584432	96312	2.69	1.68e+07	482527	8491	1.76	9.96+e06
4	12	2370679	63869	2.69	1.22e+07	484567	8098	1.67	9.93+e06
4	16	1771385	49412	2.79	1.10e+07	493489	8450	1.71	1.02+e07

表 6 バスアクセス率の内訳（Kernel CG Class B，hybrid 版）
Table 6 Breakdown of the memory bus access ratio (Kernel CG Class B, hybrid).

		Matrix-Vector Multipy				Others			
#n	#p	Mem_Ref	Bus_Acs	(%)	(clock)	Mem_Ref	Bus_Acs	(%)	(clock)
1	1	217412707	12501741	5.75	1.31e+09	3156953	207042	6.56	1.90e+07
1	2	108746200	6372841	5.86	8.04e+08	2826366	217798	7.71	1.86e+07
1	3	72475713	4072072	5.62	6.33e+08	3149855	225583	7.16	1.98e+07
1	4	54714297	3078329	5.63	6.00e+08	2830522	226968	8.02	1.92e+07
2	2	108566655	6111340	5.63	6.30e+08	61575387	106567	0.17	6.74e+08
2	4	54513309	3063132	5.62	3.85e+08	39954818	116813	0.29	4.40e+08
2	6	36241337	2031391	5.61	3.16e+08	33181527	120663	0.36	3.66e+08
2	8	27260609	1529695	5.61	2.99e+08	31668732	121547	0.38	3.40e+08
3	3	72435986	4076056	5.63	4.37e+08	36609720	77993	0.21	4.98e+08
3	6	36286096	2034490	5.61	2.62e+08	22447846	87074	0.39	3.22e+08
3	9	24160124	1353170	5.60	2.09e+08	19724424	93952	0.48	2.71e+08
3	12	18134223	1016271	5.60	2.00e+08	19421917	95921	0.49	2.66e+08
4	4	54568112	3066328	5.62	3.28e+08	24272702	67525	0.28	3.82e+08
4	8	27200891	1523909	5.60	1.99e+08	17051197	78247	0.46	2.72e+08
4	12	18121732	1016397	5.61	1.57e+08	15314768	82280	0.54	2.45e+08
4	16	13577297	763572	5.62	1.50e+08	12989860	87336	0.67	2.19e+08

アクセス率は疎行列・ベクトル積で生じていることが分かる．これは，大規模数値計算においてキャッシュメモリが有効に働かない典型的な例である．ただ，この部分は途中に同期処理を含まずに並列化が可能なので，使用するプロセッサ数の増加によりバスのアクセス回数が減り，処理時間も減少することが分かる．さらに，full-MPI 版と hybrid 版での違いもほとんど見られない．

これに対し full-MPI 版では，ノード内のプロセッサ台数の増加に従い演算以外の通信の処理時間が増加しボトルネックとなっている．バスのアクセス回数やアクセス率では逆に hybrid 版の方が高い値を示しているが処理時間は少ない．これは，ノード内の通信方法が full-MPI 版は TCP/IP レベルのカーネルを介した通信を行っているのに対して，hybrid 版ではユーザレベルの共有メモリを介した通信を行っているためである．図 1 に示したとおり，ノード内での共有メモリのアクセスは TCP/IP レベルの通信よりも高速であるために結果的には通信部分のバスのアクセス回数やアクセス率は問題になっていない．

Kernel CG は主な計算である疎行列・ベクトル積の部分は高い並列性があり高速化できるが，プロセッサ数の増加にともなうベクトル Collection の時間がオーバヘッドとなってしまう．hybrid 版ではノード内

での Collection 通信のオーバヘッドが不要となり，この結果全体のスケーラビリティが向上した．

4. メモリバスバンド幅の改善

今回性能評価を行った Linpack では，16 台のプロセッサを用いても約 4〜7.5 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，Kernel 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．

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 のインタリーブが可能であるが，現在はインタリーブを行っていない．そこで，他のノードのメモリを 1 ノードに集中的に装着し，メモリのインタリーブ機能を有効にした場合の実験を行った．**表 7** 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．測定には，64 MB の整数配列データの連続読み出し（read），64 MB の整数配列データの連続書き出し（write），64 MB の整数配列間のコピー（copy）のプログラムを用いた．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分かった．さらに，Pthreads 版の Linpack および Kernel 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 Kernel CG において性能向上が見られた．**図 12** に Kernel CG のデータサイズ Class-B のメモリインタリーブなし（no IL.）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 ではインタリーブなしに比べて約 1.6 倍の性能向上が得られることが分かった．

今後，COSMO の全ノードのメモリを 4-way に拡張する予定であり，これによって，メモリバスアクセス率が減らない場合においてもメモリアクセスレイテンシが減少し，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．

表 7 メモリバスのバンド幅（MB/s）
Table 7 Memory bus bandwidth (MB/s).

read

スレッド数	no IL.	2-way IL.	4-way 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	no IL.	2-way IL.	4-way 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	no IL.	2-way IL.	4-way 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

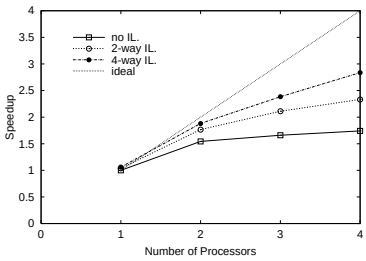

図 12 メモリの way 数の変更による速度向上率の比較（Kernel CG Class B）
Fig. 12 # of memory way vs. speed up ratio (Kernel CG Class B).

5. 関 連 研 究

Mucci らが開発している PAPI[10] は，我々が使用したパフォーマンスカウンタのような機構を様々なプラットフォームの上で利用にするための API を提供する．このカウンタは性能評価だけでなく，コンパイラの最適化やデバッグに用いることも可能である．PAPI は特に SMP を意識して作られており，様々なプログラミング方法が考えられる SMP クラスタの特性の定量的な解析に有効な手段となる．

また，本論文ではすべて MPI で行うプログラミングや MPI と Pthread のハイブリッドプログラミング

によって SMP クラスタを評価した．これに対して，SMP クラスタ上でソフトウェアによる分散共有メモリ（DSM）を行う研究もさかんに行われれている．Qin らは 4 CPU を 1 ノードとした SMP クラスタ上での共有メモリの評価をシミュレーションによって行っている[11]．Jiang らは 4 CPU の Intel PentiumPro SMP を 1 ノードとし，高速な Myrinet ネットワークで 16 ノードを結合したクラスタの上で共有仮想メモリを実装している[12]．このようなソフトウェア DSM においても，ノード内のバスボトルネックは深刻な問題であり，本論文で行ったような手法で定量的に解析することが必要である．

6. ま と め

SMP-PC クラスタではしばしばメモリバスボトルネックが問題となる．本論文では，HPC ベンチマークプログラムのバスアクセス率を実測することにより，このバスボトルネックがどのように性能に影響するかを定量的に評価した．

評価結果から，1 プロセッサのバスアクセス率が 1%程度の場合にはバスボトルネックは生じずノード内のプロセッサ数の増加が良いスケーラビリティを示すが，バスアクセス率が 2.5%や 5%を超えるような場合にはバスボトルネックによってノード内のスケーラビリティは低下することが分かった．また，ハイブリッドプログラミングは，個々のプロセッサが MPI 通信をする必要がないためにバスアクセス率を下げるとともにカーネルレベルの通信が軽減され高速化できることが分かった．

最近の SMP クラスタの事情を反映し，SMP を意識した MPI 実装等がなされ，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[13] 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[14] が 1 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 と 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 日頃，性能評価技術に関し議論していただいている 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．なお，本研究の一部は文部省科学研究費補助基盤研究（C）課題番号 12680327 による．

参 考 文 献

1) Bailey, D., Barszcz, E., Barton, J., Browning, D., Carter, R., Fatoohi, R., Fineberg, S., Frederickson, P., Lasinski, T., Schreiber, R. and Simon, H.: The NAS Parallel Benchmarks, Technical Report, RNR-94-007, NAS (Mar. 1994).
2) Bailey, D., Harris, T., Saphir, W., van der Wijngaart, R., Woo, A. and Varrow, M.: The NAS Parallel Benchmarks 2.0, Technical Report, NAS-95-020, NAS (Dec. 1995).
3) Message Passing Interface Forum: MPI: A Message-Passing Interface Standard, *Proc. ACM International Conference on Supercomputing '93*, pp.878–883 (1993).
4) Accelerated Strategic Computing Initiative (ASCI) at Los Alamos NATIONAL LABORATORY (LANL): Blue Mountain at LANL, `http://www.lanl.gov/asci/bluemtn/bluemtn.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，情報処理学会論文誌，Vol.40, No.5, pp.2215–2224 (1999).
6) Gropp, W. and Lusk, E.: Installation guide for mpich, a portable implementatation of MPI, Technical Report, CS-94-230, Computer Science Dept. University of Tennessee (1994).
7) Leroy, X.: The LinuxThreads library, `http://pauillac.inria.fr/~xleroy/linuxthreads/`.
8) Dongarra, J., Bunch, J., Moler, C. and Stewart, G.W.: LINPACK User's Guide, SIAM, Philadelphia, PA (1979).
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 Kernel CG の評価，情報処理学会論文誌，Vol.39, No.6, pp.1757–1765 (1998).
10) Mucci, P.J., Browne, S., Deane, C. and Ho, G.: PAPI: A Portable Interface to Hardware Performance Counters, *Department of Defense HPCMP Users Group Conference* (June 1999).
11) Qin, X. and Baer, J.-L.: Optimizing Software Cache-coherent Cluster Architectures, *Proc. IEEE Supercomputing '98* (Nov. 1998).
12) Jiang, D., O'Kelley, B., Yu, X., Yu, S., Kumar,

S., Bilas, A. and Singh, J.P.: Application Scaling under Shared Virtual Memory on a Cluster of SMP, *Proc. ACM International Conference on Supercomputing '99* (June 1999).
13) MPI2, `http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html`.
14) OpneMP, `http://www.openmp.org/`.

(平成 12 年 2 月 9 日受付)
(平成 12 年 6 月 2 日採録)

板倉 憲一（正会員）
昭和 44 年生．平成 5 年筑波大学第三学群情報学類卒業．平成 11 年同大学院工学研究科博士課程修了．博士（工学）．平成 11 年筑波大学計算物理学研究センターリサーチ・アソシエイト．並列計算機アーキテクチャ，並列入出力・可視化機構の研究に従事．IEEE 会員．

早川 秀利
昭和 51 年生．平成 10 年筑波大学第三学群情報学類卒業．平成 12 年同大学大学院理工学研究科修士課程修了．同年，富士ソフト ABC（株）に入社，システムエンジニア．リアルタイム系ソフトウエアシステムの開発に従事．並列処理等に興味を持つ．

近藤 正章（学生会員）
昭和 50 年生．平成 10 年筑波大学第三学群情報学類卒業．平成 12 年同大学大学院工学研究科博士前期課程修了．現在，東京大学大学院工学系研究科博士後期課程在学中．計算機アーキテクチャ，ハイパフォーマンスコンピューティングの研究に従事．

吉川 茂洋（学生会員）
昭和 51 年生．平成 11 年度筑波大学第三学群情報学類卒業．現在，同大学大学院システム情報工学研究科に在学中．共有メモリ・クラスタにおける高性能計算に関する研究に従事．

朴 泰祐（正会員）
昭和 59 年慶應義塾大学工学部電気工学科卒業．平成 2 年同大学大学院理工学研究科電気工学専攻後期博士課程修了．工学博士．昭和 63 年慶應義塾大学理工学部物理学科助手．平成 4 年筑波大学電子・情報工学系講師，平成 7 年同助教授，現在に至る．超並列処理ネットワーク，超並列計算機アーキテクチャ，ハイパフォーマンスコンピューティング，並列処理システム性能評価の研究に従事．電子情報通信学会，日本応用数理学会，IEEE 各会員．

佐藤 三久（正会員）
昭和 34 年生．昭和 57 年東京大学理学部情報科学科卒業．昭和 61 年同大学院理学系研究科博士課程中退．同年新技術事業団後藤磁束量子情報プロジェクトに参加．平成 3 年，通産省電子技術総合研究所入所．平成 8 年より，新情報処理開発機構つくば研究センターに出向．現在，同機構並列分散システムパフォーマンス研究室室長．理学博士．並列処理アーキテクチャ，言語およびコンパイラ，計算機性能評価技術，グローバルコンピューティング等の研究に従事．日本応用数理学会会員．

田中 良夫（正会員）
昭和 40 年生．平成 7 年慶応義塾大学大学院理工学研究科後期博士課程単位取得退学．平成 8 年技術研究組合新情報処理開発機構入所．平成 12 年通産省電子技術総合研究所入所．現在に至る．博士（工学）．クラスタを中心とした並列システム上での高性能計算，グローバルコンピューティングシステム，Lisp 処理系（主に並列ガーベッジコレクション）に関する研究に従事．IC '99 論文賞．ACM 会員．