社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2009－ARC－182 (31)
2009－HPC－119 (31)
2009／2／28

NUMA 並列型クラスタ上での効率的なスケジューリング

小西祐介[†] 野村哲弘[†]
松葉浩也[††] 石川裕[†]

ジョブのメモリおよびネットワーク負荷の統計情報があらかじめ取得されているジョブ群において、NUMA 並列型クラスタ上での効率的なバッチジョブスケジューリング手法を提案する。本提案手法では、単純なメモリや I/O のリクエスト数をスケジューリング指標とする代わりに、同時実行したメモリ、I/O アクセスを行うベンチマークの速度低下を指標として用いる。NAS Parallel Benchmarks タスクセットでは、従来手法より良いスケジューリングを行うことができることを確認した。

Efficient Scheduling for Cluster of NUMA Parallel Computers

Yusuke Konishi,[†] Akihiro Nomura,[†] Hiroya Matsuba,[††]
and Yutaka Ishikawa[†]

We propose a new algorithm to schedule jobs on a cluster of NUMA machines. Instead of using simple memory or I/O request rate as the scheduling metrics, we use degradation of performance rate in memory and I/O benchmarks concurrently executed with the task as an indicator. We confirm the proposed scheduler using the NAS Parallel Benchmarks.

1. はじめに

近年のコモディティCPU のマルチコア化に対応し、コモディティCPU を用いたクラスタシステムでもマルチコアシステムが主流である。例えば、2008 年 6 月に筑波大、東大、京大に導入された T2K オープンスーパーコンピュータでは、1 ソケットあたり 4 コアの AMD 社製 Opeteron CPU を 4 ソケット搭載したノードを構成要素としている[1)]。

クラスタのセンター運用では、Torque[2)]、OpenPBS[3)]、GridEngine[4)] などのオープンソースで提供されるバッチジョブシステムや商用バッチジョブシステムが使われる。これらバッチジョブシステムでは、一般に、ノード単位あるいはコア単位でのジョブ割り当てが行われる。例えば、T2K オープンスーパーコンピュータのようなハードウェア環境において Torque を使用した場合、8 ノードかつノード毎に 8CPU を要求するジョブと 8 ノードかつノード毎に 8CPU を要求するジョブは、同一 8 ノードに 2 つのジョブを実行させることも出来るし、別々の 8 ノードあるいは同一の 8 ノードに逐次にジョブを割り当てることも出来る。前者を共有ジョブ割り当てポリシ、後者を排他ジョブ割り当てポリシと呼ぶことにする。

上記割り当てポリシはシステム設定時に決まる。共有ジョブ割り当てポリシの場合、ノード上のコアがアイドル状態になることを避けることが出来るが、複数のジョブが同時に実行されて、トータルスループットが上がるとは限らない。一方、排他ジョブ割り当てポリシでは、システム全体でコアの使用率が下がる。

バッチジョブシステムの待ち行列中に存在するジョブのメモリおよび通信 I/O 使用状況が予測できたとき、これらジョブを組み合わせて、同じノード群を共有させて実行した方が、排他ジョブ割り当てによる逐次ジョブ実行よりもスループットが向上する可能性がある。本論文では、NUMA 並列型クラスタ上で、ジョブが持つメモリおよびネットワーク負荷の統計情報から共有ジョブ割り当て可能ジョブ集合を導き、ジョブスケジューリングする手法を提案する。

本論文では、関連研究を述べた後、まず、NAS Parallel Benchmarks (NPB) を用いた実験を通じ、排他ジョブ割り当てポリシにおけるベンチマーク性能と共有ジョブ割り当てポリシにおけるベンチマーク性能を測定する。共有ジョブ割り当てポリシにおいては、NPB の 5 本のベンチマークの中から、2 本のベンチマークの全ての組み合わせに対して、同時に実行したときの実行時間を測定する。この結果、組合せによって、排他ジョブ割り当てポリシによる逐次ジョブ実行よりも、スループットが向上することを示す。

次に、効率良いジョブの組合せを自動選択すること

† 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, University of Tokyo
†† 東京大学情報基盤センター
Information Technology Center, University of Tokyo

を可能にするための指標を次のように求める。まず、ジョブの資源アクセス率として主記憶アクセス率と通信I/Oアクセス率を指標として、NPBの5本の統計情報を個別に取得する。この統計情報からは、効率良いジョブの組合せを自動選択できないことを示す。

そこで、ジョブの統計情報を個別に取得するのではなく、主記憶アクセスだけを行うmemprobeジョブや、I/Oアクセスだけを行うcomprobeジョブを同時実行し、これの性能低下率を求める。これは、統計情報を取得したいジョブが、他のタスクの主記憶アクセスや、I/O操作に対してどの程度の外乱を与える事になるのかを取得していることになる。本統計情報を用いると、NPBの5本のジョブに対して、適切な組合せを求めることが出来ることを示す。

2. 関連研究と本研究の特徴

様々な環境において共有資源の競合から来る速度低下を回避するスケジューラが提案されてきた。Snavelyら[5]はSMT (Simultaneous Multi-Threaded) プロセッサ上で同時に実行するタスクを選択するスケジューラSOSを実装した。SMTプロセッサ上で、タスクはメモリやキャッシュ、パイプラインユニットなどを共有している。SOSでは同時に実行するタスクの組み合わせを複数通り試し、その中からパフォーマンスカウンタが良い値を示したものを選択する事で性能向上を図っている。

Antonopoulosら[6]はSMPのマルチプロセッサ上で共有バスの使用率を均等に近づけるスケジューラを提案した。また、McGregorら[7]は、SMTのマルチプロセッサ上でbus transaction数、stall cycle数、cache miss数を指標として用いるスケジューラを比較し、bus transaction数を元にスケジューラを構築するのが有効だと主張している。Koukisら[8),9)]はSMPクラスタ上で、メモリバンド幅とネットワークバンド幅の使用率を均等に近づけるスケジューラを提案した。

我々が知る限り、既存研究ではNUMA並列型クラスタにおけるスケジュールの研究は存在しない。

3. 実験環境

本論文では、全ての実験を4台からなるNUMAマシンクラスタ上で行った。各マシンは2つのAMD Dual-core Opteron 2212プロセッサを備えており、メモリはDDR2-667 6GB、ネットワークインータフェースとしてMyrinet 10G (1.25GBps) を使用している。CPUコアは各々が1MBのL2キャッシュを持っていて、共有はしていない。

OSはLinux 2.6.18に、パフォーマンスカウンタを取得できるよう独自の変更を加えたものを用いている。MPIはmpich2-mx 1.0.7で、ラッパーを通す事でMPI関数の内部で経過した時間などを取得できるようにしている。

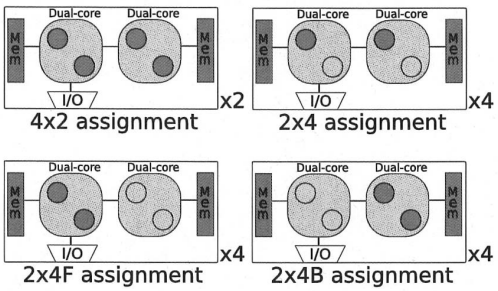

図1 タスクの割り当て方

NUMA環境では同じノードのメモリをアクセスする場合より、異なるノードのメモリをアクセスする場合に速度が低下する。メモリの割り当て方により性能が変わる問題を避けるため、実験においてはnumactlコマンドを介してプロセスが配置されたCPUに最も近いメモリにプログラム/データを配置する。

4. 予備実験

4.1 実験の仮定

本実験では、4コア4ノードのクラスタ資源において、16プロセスを必要とするタスク群の割り当て方法についての予備実験である。評価指標はトータルスループットとする。ベンチマークは、NAS Parallel Benchmarks[10]のEP, FT, CG, LU, MGの5本を用いた。問題サイズはCで、並列数は8である。

バッチジョブキューには、5種類のベンチマークが多数投入され、ジョブスケジューラは、この中から、2つを選択し同時実行する場合を仮定する。

4.2 単独実行時の性能

排他ジョブ割り当て方式において、4コア4ノードクラスタに対すし8プロセスタスクを割り当てるには、4コア2ノードを割り当て方と2コア4ノードを割り当てる方法がある。単一タスクに対してCPU使用率が上がる4コア2ノードの割り当て方を基準に議論していく。1つのタスクに4コア2ノードを単独で割り当てた場合の実行時間を表1に示す。

表1 4コア2ノード単独実行時間

ベンチ	実行時間 (sec)
EP	77.22
CG	83.97
FT	122.50
LU	418.43
MG	31.05

4.3 同時2タスク実行時の性能

8プロセスのタスクを4ノードクラスタ上で実行する場合の配置方法は、主に4通り存在する。その配置

表 2　2 タスク同時実行時の単体実行時に対する必要時間の変化 (%)

ベンチ	配置	同時実行				
		EP	CG	FT	LU	MG
EP	4x2	-0.30	-1.16	1.22	-0.20	0.45
	2x4	-0.92	-2.00	-0.66	-1.02	-0.39
	2x4F	-2.39	-3.25	-1.79	-3.10	-0.72
	2x4B	-0.69	-0.26	0.05	-0.06	0.52
CG	4x2	0.43	0.04	0.11	-0.07	-0.50
	2x4	-3.37	0.05	8.63	2.83	5.03
	2x4F	-2.78	-0.26	3.49	-1.95	-0.32
	2x4B	-0.96	0.84	3.76	0.06	1.73
FT	4x2	-0.29	0.11	0.56	-0.13	-0.25
	2x4	-10.15	-5.87	3.39	-1.56	-1.90
	2x4F	-5.70	-2.49	-1.37	-3.67	-3.43
	2x4B	-3.60	-1.41	-1.45	-2.11	-1.41
LU	4x2	0.08	-0.03	-0.03	-0.09	0.05
	2x4	-9.85	-4.91	-0.32	0.69	2.12
	2x4F	-0.85	0.96	2.19	-0.04	1.21
	2x4B	-0.14	0.46	0.44	0.35	2.39
MG	4x2	-0.22	-0.08	0.25	0.31	-0.25
	2x4	-15.06	-9.08	-4.44	-3.55	-1.13
	2x4F	-1.17	-0.07	2.81	0.61	1.12
	2x4B	2.50	2.30	1.56	0.05	1.30

方法を図 1 に示す。図において、**4x2** は 1 ノードに 4 プロセスを配置し、2 ノードを用いる配置である。**2x4**、**2x4F**、**2x4B** は、1 ノードに 2 プロセスを配置し、4 ノードを用いる配置である。このような配置方法は、各プロセッサに 1 プロセスずつ配置する方法、I/O に近いプロセッサに 2 プロセス配置する方法、I/O から遠いプロセッサに 2 プロセス配置する方法の 3 パターンが存在する。**2x4**、**2x4F**、**2x4B** は、この 3 パターンを意味している。ベンチマーク群から 2 つのタスクを選んで 4 ノード上で実行する場合、片方のプログラムを **4x2** で実行するとき、もう片方のプログラムは別の 2 ノード上の **4x2** で実行され、**2x4**、**2x4F**、**2x4B** で実行するときは、もう片方のプログラムは同じ 4 ノード上の **2x4**、**2x4B**、**2x4F** で実行される。

5 種類のベンチマークから 2 つを組み合わせ、図 1 に示した 4 通りの配置で同時実行した。表 1 の通り、使用したベンチマークは実行時間が異なるため、片方のベンチマークが終了した場合、そのベンチマークを始めから実行しなおすことで測定を行っている。

本実験によるプログラムの実行時間の結果から、表 1 で示した 4 コア 2 ノード上で単独実行したときの実行時間に比べて、2 つ同時に実行したときの実行時間が、どのくらい時間が短縮したかの比率を表 2 に示す。本実験の実測値は、付録の表 9 に掲載する。

表 2 より、MG と EP を 2 コア 4 ノードによる共有ジョブ割り当てすることにより、4 コア 2 ノードによる MG 実行時間に比べて 15.06%の短縮、4 コア 2 ノードによる EP 実行時間に比べて 0.39%の短縮となる。もし、MG と EP の 2 つのタスク群が十分にバッチジョブキューに存在すれば、2 コア 4 ノードによる共有ジョブ割り当てのトータルスループットは、4 コア 2 ノードで排他ジョブ実行するときのトータルスループットより、7.73%の時間短縮になることがわかる。

一方、組合せによっては、実行時間が遅くなる場合がある。最悪組合せは、2 つの FT を 2 コア 4 ノードによる共有ジョブ割り当てとする場合である。この場合、トータルスループットは、3.39%増加することになる。

表 2 の結果から、EP、CG、FT、LU、MG のタスクを 2 つずつ、計 10 個のタスクを実行するというワークロードを考えた場合、表 3 に示す通り、EP と MG を **2x4** で 2 組、CG と LU を **2x4** で 2 組、FT と FT を **2x4F/2x4B** の配置で実行することで最大の 3.79%の性能向上が見込めることがわかる。

表 3　実験結果による組合せ選択

2x4 EP-MG, 2x4 CG-LU, 2x4F/B FT-FT
2x4 EP-MG, 2x4 CG-LU

本実験から、排他ジョブ割当て方式よりもジョブ共有割当て方式の方がトータルスループット性能が良くなるスケジューリングがあることが分かった。次節では、最適な 2 つのタスクの組合せを選択する基準として、タスクの計算資源に対するアクセス統計情報 (メモリや通信) を考える。

4.4　単独実行統計情報

NPB5 本のベンチマークのメモリ要求数、I/O 要求数を表 4 に示す。メモリ要求密度は、EP, FT, CG, LU, MG の順に増えていることが分かる。ここで、DRAM Accesses の情報は、Opteron プロセッサのパフォーマンスカウンタ[11] である DRAM Accesses のレジスタ情報であり、各メモリに対するアクセス数を示す。CPU のキャッシュにヒットするデータアクセスは含まれない。I/O to MEM は、I/O 接続側のプロセッサのパフォーマンスカウンタである IO Requests to Local Memory と、IO Requests to Remote Memory の和である。どちらの値も最大となるマシン上での値を採用している。

表 4　1ms あたりの平均要求数

ベンチマーク	DRAM Accesses	I/O to MEM
EP	767	0
FT	19548	2385
CG	23469	1069
LU	33307	72
MG	40915	381

本統計情報のみを用いて、5 種類 2 つずつ計 10 のタスクを同時実行するときの組み合わせを、メモリアクセス率の大きいもの、小さいものをペアにするように決めた結果を以下に示す。

2x4 EP-MG, 2x4 FT-LU, 2x4 CG-CG
2x4 EP-MG, 2x4 FT-LU

実際の最適組合せ結果と比較すると、EP と MG の

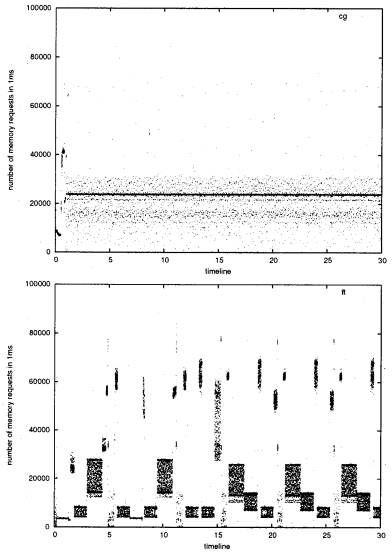

図 2 メモリアクセス数の推移、上が CG で下が FT

組合せ以外は、組合せが異なっている。アプリケーションのメモリアクセス数と I/O アクセス数という単純な統計情報だけでは、最適スケジューリングできないことが分かる。

4.5 メモリアクセス密度のばらつき

実験環境で用いた Opteron では、コントロールレジスタに適切な値を入れることにより、CPU 内のパフォーマンスカウンタを特権モードでなくても読み出す事ができる。また、DRAM アクセス数などの特定の情報は、プロセッサ内のどのコアからでも同じ値を取得することができる。よって、2 コア以上あるプロセッサので測定対象のプロセスを実行し、他のコアでタイムスタンプカウンタを読みながら定期的にパフォーマンスカウンタを記録することで測定対象に与える影響を最小限に抑えながら非常に細かい粒度で情報の取得が可能になる。

2x4 の配置における CG と FT のタスク開始後 30 秒のメモリ要求の推移を図 2 に示す。この図では 1ms に 1 回サンプリングを行っている。CG のメモリ要求数は 24000 回程の一定要求数であるが、FT では粗密の差が非常に大きく、最大で 70000 回程になっている。すなわち、表 4 では、CG の方が FT よりもメモリアクセス数が大きく観測されているようにみえているが、実際には、FT の方が周期的に CG の 3 倍程度のメモリアクセス数が生じていることが分かる。このような細かい周期でメモリアクセスの密度がばらつくタスクの場合、メモリ要求数の平均を見るだけでは、他のタスクに与える影響を見誤ることがわかる。

5. 予測手法

5.1 手　　法

メモリアクセス頻度はばらつくため、メモリアクセスが原因として起こる速度低下によりスケジューリングを行うには、パフォーマンスカウンタを見るだけでは難しい。また、NPB を含む多くのクラスタ上で動作する並列アプリケーションは、プロセス内での計算とプロセス間の通信を交互に行う。このため、通信についてもパフォーマンスカウンタによる通信に伴う I/O アクセス数だけで、通信による実行時間低下率を予想してスケジューリングするのは難しい。

そこで、我々はタスクが他のタスクのメモリアクセスならびに通信に与える影響をより良く見積もるために、以下の手法を提案する。

メモリアクセスに関しては、メモリに連続して書き込みを行う memprobe と呼ばれるプログラムを導入する。memprobe プログラムは L2 キャッシュより十分大きな領域のメモリにキャッシュライン間隔で書き込みを行い、頻繁なメモリアクセスを引き起こすプログラムである。memprobe プログラムを 2x4 の配置で実行し、測定タスクを同時に実行する。測定タスクのメモリアクセスや通信の影響で、memprobe プログラムの実行時間は、単体で実行したときに比べ遅くなる。memprobe プログラムの性能低下率を求める。

通信に関しては、ノード間で全体全通信を行う comprobe と呼ばれるプログラムを導入する。comprobe では、各ノードに 1 つのプロセスを作り、全体全通信を繰り返す。同時に、測定タスクを同時に実行する。測定タスクの通信の影響で、comprobe の実行時間は、単体で実行したときに比べ遅くなる。comprobe プログラムの性能低下率を求める。

memprobe の性能低下率と comprobe の性能低下率の 2 つの性能指標から、次のようなアルゴリズムで同時実行する組合せを決める。

(1) memprobe の性能低下率から、性能低下率の低いタスクと高いタスクを同時実行候補とする。
(2) comprobe の性能低下率の高いタスクと同時に実行するタスクの通信時間の割合が高い場合には、その組合せでは必ずしも性能が高いとは限らない。このような選択は同時候補から除外する。
(3) 組合せが決まっていないタスク群を使って、再度、(1) (2) を適用する。

5.2 テ ス ト

まず、ネットワークの影響を排除した状態下でこの手法が、パフォーマンスカウンタを用いた推論より良い結果を示すということを確認する。そのために、ほとんど通信を行わないように変更を加えた CG, FT, LU, MG(以下 NCCG, NCFT, NCLU, NCMG、ま

とめて無通信 NPB と呼ぶ) を用意した。

表 5 は、変更を行っていない 5 種類と、変更を加えた 4 種類のプログラムと同時実行時の memprobe プログラムの性能低下率である。なお、EP は主計算時に通信が発生しないプログラムなので、新たな測定は行っていない。本結果から、EP を除く性能低下率は、CG、FT、LU、MG の順番で悪くなっている (性能低下率が高くなっている)。

表 5 memprobe の性能低下率

	性能低下率		性能低下率
EP	3.1%		
CG	20.1%	NCCG	21.9%
FT	26.2%	NCFT	27.1%
LU	29.9%	NCLU	30.6%
MG	38.2%	NCMG	40.4%

本結果から提案手法の 1 番目の規則を適応したときの組合せを以下に示す。

2x4 EP-MG, 2x4 CG-LU, 2x4 FT-FT
2x4 EP-MG, 2x4 CG-LU

表 6 は、comprobe の性能低下率を示している。本表左列「性能低下率 (I/O) 側」は、comprobe を I/O ポートに近いコアで実行したときの結果であり、右列は、comprobe を I/O ポートから遠いコアで実行したときの結果である。LU は、comprobe を実行するコアの位置によって性能に差が出ているが、5 つの中で最も高い性能低下率を示しているのは、FT であることがわかる。FT と組み合わせたタスクは通信による性能低下が著しくなることを意味している。上記組み合わせ候補では、FT2 つが残っているため、この組み合わせは変更されない。

本組合せ結果は、表 3 で示した最適解と同じ組合せ結果となることが分かる。

表 6 comprobe の性能低下率

	性能低下率 (I/O 側)	性能低下率
LU	1.9 %	11.9 %
CG	4.0 %	6.2 %
MG	4.3 %	6.5 %
FT	11.2 %	16.8 %

6. 議　　論

提案手法で使用してた memprobe の速度低下率が、正しくタスクの速度低下率と同じになっているかを確認する。表 7 は無通信 NPB を 2x4 の配置で同時実行した場合の実行時間である。本表の行および列の並びの順番は、表 5 における性能低下率が少ない順に並べている。全ての組合せで、表 5 の性能低下で示された通りの順番で、速度低下が引き起こされていることが分かる。

すなわち、memprobe が示す速度低下率は、通信を行なわないタスクに対しては、従来手法より良い指標を示している。

表 7 無通信 NPB の同時実行時の実行時間 (s)

	NCCG	NCFT	NCLU	NCMG
NCCG	78.55	78.99	81.44	82.18
NCFT	103.60	104.12	107.47	109.20
NCLU	391.65	398.45	419.29	432.60
NCMG	27.91	28.30	30.04	30.22

NPB の 10 のタスクの組み合わせ選択では、comprobe の結果を使わなかったが、comprobe が示す速度低下率が、通信を行うタスクに対して、正しい指標を示しているか確認する。表 8 に、2x4 で NPB を単独で実行したときに比べて MPI 関数内でどれだけの経過時間が増えたかを示す。表は、comprobe の性能低下率の低いタスクを左から右に、あるいは、上から下に並べた。下線部分の LU と LU、MG と LU、FT と LU の組み合わせが予測と異なる部分である。LU は comprobe の使用するコアの配置によって大きな性能低下が生じる。LU と LU、MG と LU、FT と LU の組み合わせで、LU、MG、FT がそれぞれ通信性能劣化が大きいのは、LU による通信性能劣化の大きい状況下で計測されたからだと予想される。

表 8 MPI 関数内経過時間

	同時実行 (sec)			
ベンチ	LU	CG	MG	FT
LU	4.20	0.50	3.47	9.30
CG	0.71	0.84	0.87	6.661
MG	0.23	0.13	0.28	1.01
FT	1.20	0.7	0.7	9.13

10 タスクのワークロードにおける最適組み合わせと今回提案手法により導かれた組み合わせは、2 つの FT の配置方法のみが異なるという結果となった。調べたところ、FT は一定周期で比較的大きな通信を行うプロセスで、FT2 つを 2x4 で実行すると、密度の高いメモリアクセスや通信を行う部分が重なり、周期をずらした場合と比較し、11%遅くなるということが分かった。その為、FT2 つを 2x4 で動かすという判断は平均的には問題が無いものと思われる。

7. ま と め

ジョブのメモリおよびネットワーク負荷の統計情報があらかじめ取得されているジョブ群において、NUMA 並列型クラスタ上で、複数のジョブを同時に実行することによりトータルスループットを向上させるジョブスケジューリング手法を提案した。本論文で提案した手法では、あるジョブの計算資源に対する負荷指標を得るために、そのジョブと同時に動かした、メモリ、I/O アクセスを行うベンチマークの性能を用いることが大きな特徴である。負荷指標は、メモリおよび I/O アクセスベンチマークを単体で実行したときと、そ

のジョブと同時に実行したときの性能低下率とする。NAS Parallel Benchmarks から EP、CG、FT、LU、MG の 5 種類のベンチマークによるジョブスケジューリングにおいて、最適にジョブをスケジュールすることが出来ることを示した。

本手法は、あらかじめジョブの統計情報が入手できていて、ジョブキューには相応のジョブが待機しているという仮定をおいているだけでなく、いくつかの制限がある。ジョブ実行時間が一様でない場合に、空き資源を有効利用出来なくなる可能性がある。また、2 つの同時実行のみを考慮している点である。本手法は、プロダクションラン用に使われているクラスタ上で、ジョブの種類が決まっていて、大量のジョブが投入される状況において利用できる可能性がある。

参 考 文 献

1) T2k open supercomputer alliance. `http://www.open-supercomputer.org/`.
2) Torque resource manager. `http://www.clusterresources.com/pages/products/torque-resource-manager.php`.
3) Openpbs. `http://www.openpbs.org/`.
4) Grid engine project. `http://gridengine.sunsource.net/`.
5) Allan Snavely and Dean M. Tullsen. Symbiotic jobscheduling for a simultaneous mutlithreading processor. *ACM SIGPLAN Notices.*, 35(11):234–244, 2000.
6) ChristosD. Antonopoulos, DimitriosS. Nikolopoulos, and TheodoreS. Papatheodorou. Scheduling Algorithms with Bus Bandwidth Considerations for SMPs. *International Conference on Parallel Processing*, 2003.
7) R.L. McGregor, C.D. Antonopoulos, and D.S. Nikolopoulos. Scheduling Algorithms for Effective Thread Pairing on Hybrid Multiprocessors. *19th IEEE Parallel and Distributed Processing Symposium*, April 2005.
8) E.Koukis and N.Koziris. Memory bandwidth aware scheduling for SMP cluster nodes. *13th Euromicro Conference on Parallel, Distributed, and Network-Based Processing*, pages 187–196, Feb. 2005.
9) Evangelos Koukis and Nectarios Koziris. Memory and Network Bandwidth Aware Scheduling of Multiprogrammed Workloads on Clusters of SMPs. *International Conference on Parallel and Distributed Systems*, 1:345–354, 2006.
10) NASA Ames Research Center. Nas parallel benchmarks. `http://www.nas.nasa.gov/Resources/Software/npb.html`, Oct 1997.
11) Advanced MicroDevices Inc. Bios and kernel developer's guide for amd athlon 64 and amd opteron prosessors rev 3.30 no.26094, Feb 2006.

付　　録

表 9　2 タスク同時実行時の実行時間

ベンチ	配置	同時実行 EP	CG	FT	LU	MG
EP	4x2	77.13	76.47	78.30	77.21	77.71
	2x4	76.65	75.82	76.85	76.57	77.06
	2x4F	75.52	74.85	75.98	74.97	76.81
	2x4B	76.83	77.17	77.40	77.32	77.77
CG	4x2	84.45	84.12	84.18	84.03	83.67
	2x4	81.26	84.13	91.34	86.46	88.32
	2x4F	81.75	83.87	87.02	82.45	83.82
	2x4B	83.28	84.80	87.25	84.14	85.54
FT	4x2	122.46	122.95	123.51	122.66	122.50
	2x4	110.35	115.61	126.98	120.90	120.48
	2x4F	115.82	119.75	121.14	118.31	118.60
	2x4B	118.39	121.08	121.03	120.23	121.09
LU	4x2	419.85	419.38	419.38	419.13	419.72
	2x4	378.16	398.91	418.15	422.39	428.38
	2x4F	415.93	423.51	428.67	419.33	424.55
	2x4B	418.91	421.41	421.32	420.96	429.50
MG	4x2	31.04	31.08	31.19	31.20	31.03
	2x4	26.43	28.28	29.73	30.00	30.76
	2x4F	30.75	31.09	31.98	31.30	31.46
	2x4B	31.89	31.83	31.59	31.13	31.51

表 10　2 タスク同時実行時の MPI 関数内での経過時間

ベンチ	配置	同時実行 EP	CG	FT	LU	MG
EP	4x2	2.55	2.37	3.13	2.93	3.45
	2x4	2.36	1.59	2.37	2.07	2.76
	2x4F	1.63	0.98	2.01	1.32	1.02
	2x4B	2.01	1.98	2.03	2.09	2.53
CG	4x2	4.53	4.43	4.53	4.50	4.29
	2x4	4.82	5.11	10.88	4.98	5.14
	2x4F	3.52	4.31	7.93	3.73	3.86
	2x4B	5.15	5.53	8.22	5.34	5.37
FT	4x2	16.49	16.17	16.69	16.34	16.14
	2x4	12.29	12.70	21.13	13.50	13.29
	2x4F	10.99	11.16	12.27	11.08	11.20
	2x4B	12.96	13.59	14.14	13.30	13.36
LU	4x2	15.74	15.51	15.76	16.80	16.93
	2x4	14.25	13.76	22.56	17.46	16.73
	2x4F	16.24	15.76	17.93	16.33	18.78
	2x4B	17.51	17.33	18.62	17.23	17.74
MG	4x2	0.84	0.99	1.01	0.84	0.95
	2x4	0.71	0.74	1.62	0.84	0.89
	2x4F	0.67	0.72	1.34	0.67	0.69
	2x4B	0.77	0.82	1.46	0.87	0.77

表 11　実行時間に対する MPI 関数内の経過時間の割合 (%)

ベンチ	配置	同時実行 EP	CG	FT	LU	MG
EP	4x2	3.31	3.10	3.99	3.80	4.44
	2x4	3.08	2.09	3.08	2.70	3.58
	2x4F	2.16	1.31	2.65	1.76	1.33
	2x4B	2.61	2.56	2.62	2.71	3.25
CG	4x2	5.36	5.26	5.38	5.36	5.13
	2x4	5.93	6.07	11.91	5.76	5.82
	2x4F	4.30	5.14	9.11	4.52	4.60
	2x4B	6.18	6.52	9.43	6.35	6.28
FT	4x2	13.46	13.15	13.51	13.32	13.17
	2x4	11.14	10.99	16.64	11.16	11.03
	2x4F	9.49	9.32	10.13	9.37	9.44
	2x4B	10.95	11.23	11.69	11.06	11.03
LU	4x2	3.75	3.70	3.76	4.01	4.03
	2x4	3.77	3.45	5.40	4.13	3.91
	2x4F	3.90	3.72	4.18	3.89	4.42
	2x4B	4.18	4.11	4.42	4.09	4.13
MG	4x2	2.70	3.18	3.24	2.70	3.05
	2x4	2.68	2.62	5.45	2.79	2.90
	2x4F	2.18	2.30	4.20	2.14	2.20
	2x4B	2.42	2.59	4.63	2.80	2.45