計算機アーキテクチャ 132−13
システムソフトウェアとオペレーティング・システム 80−13
ハイパフォーマンスコンピューティング 75−13
（1999. 3. 4）

並列システム性能の視覚的解析とその評価

益口摩紀[†1] 建部修見[†2] 佐藤三久[†3]
関口智嗣[†2] 長嶋雲兵[†4]

本研究では 低コストかつ高精度なMPIプロファイリングツールを設計・構築し，実際に作成したツールを用いてアルファワークステーションクラスタ etlwiz の性能評価を行った．性能測定にはクロックカウンタを用いている．これにより，etlwiz では 100 μsec. 単位の精度が得られた．また大域時間を導入することで全プロセスのタイミングをはかることができ，並列システムの性能解析や通信オーバーヘッドの測定に役立つ．etlwiz のブロッキング通信において，メッセージサイズを 4KB として受信が先に発行された状態で，送信が発行されてから受信が終了するまでの経過時間は 394 μsec，その時のスループットは 約 10.14 MB/s であった．

Performance analysis of parallel computers and parallel programs using clock-level profiling system

MAKI MASUGUCHI,[†1] OSAMU TATEBE,[†2] MITSUHISA SATO,[†3]
SATOSHI SEKIGUCHI[†2] and UMPEI NAGASHIMA[†4]

We develop a lightweight profiling tool for MPI programs using a clock counter of the CPU and precisely evaluate etlwiz, a dedicated cluster of Alpha WSs. This profiling system also supports a clock adjustment to generate a global clock in the order of 100 μsec on the etlwiz. Using the global clock, precise timings of whole parallel programs can be obtained, that can be helpful to measure performance of parallel machines and calculate communication overhead of parallel programs. In the case of blocking communication on the etlwiz, our profiling tool shows a 4 Kbyte latency of 394 μsec. from the beginning of `MPI_Send()` of a sender to the end of `MPI_Recv()` of a receiver under the condition that `MPI_Recv()` is issued in advance, and a throughput of 10.14 Mbyte/sec.

1. はじめに

科学技術計算の分野における高速計算能力の要求に応えるべく，コンピュータシステムのCPU性能の向上は最近著しく，それに対するネットワーク性能の考察を行なうことは非常に重要である．メッセージ通信の手段としては，高いポータビリティを備えたメッセージパッシングライブラリ MPICH[1] が広く用いられている．

一方，多様化が進むコンピュータシステムの中でも，最近では高速な処理を安価に実現する手段としてワークステーションクラスタに目が向けられるようになってきた．既存の資源によりユーザのニーズにあったシステムを構築することが可能であり，CPU性能の高い最新のプロセッサを用いることによりワークステーションクラスタ上での高速な並列処理が実現されている．

そこで本研究では並列システムの特徴抽出を支援するためのMPIプロファイリングツールを設計・構築し，実際に作成したツールを用いて AlphaStation のクラスタ etlwiz の性能評価を行なうことを目的とする．

更にベンチマークプログラムとして Lattice QCD の実アプリケーションである QCDMPI[2] を用い，アプリケーション実行時の各要素プロセスにおける挙動を可視化することにより並列システム性能の視覚的解析を行なう．QCDMPI は，プログラム上で演算と通信が全く分離しており，プロセッサ数やデータ分割方法に依存しないポータブルなコードである．このため，計算機の演算性能・通信性能を各々独立に評価するのに適しており，ベクトル型並列計算機を含めたベンチマークデータが数多く公表されている．QCDMPI ではデータ分割の次元を変えることでプログラムの挙動，ネットワーク性能の考察を行なう．

†1 お茶の水女子大学
Ochanomizu University
†2 電子技術総合研究所
Electrotechnical Laboratory
†3 新情報処理開発機構
Real World Computing Partnership
†4 物質工学工業技術研究所
National Institute of Materials and Chemical Research

表1 etlwizの仕様

プロセッサ(PE数)	クロック周波数	メモリ	キャッシュ	ネットワーク	バンド幅
Alpha21164(32PE)	333 MHz	128MB	8KB,96KB,2MB	100Base/TX ether switch	100Mbps

2. 基本性能

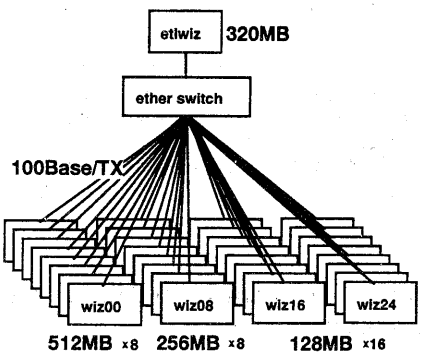

図1 etlwizの構成

図1にetlwizの構成を示す. etlwizは32台構成で, それぞれのプロセッサにおけるメモリは, 512 MB × 8台, 256 MB × 8台, 128 MB × 16台のような三段構成である. これは並列処理を実行する上で, 並列度に依存しない一定のメモリ量を確保するためである. つまり, 32 PEでの実行時に利用可能なメモリ量128 MB × 32と8PEを用いた場合に利用可能なメモリ量512 MB × 8は相等しい. また, ネットワークは100Base/TXをether switchで接続した構造となっており, ether switchのバックプレーンは1.2 Gbpsである.

3. クロックカウンタ

現在の高性能マイクロプロセッサには, クロックレベルのカウンタを持つものが多い[3]. すべての処理に対する実行単位で処理時刻を得ることができるため, クロックレベルの精密なカウンタをレジスタとして使用することは詳細な性能測定に非常に役立つ. 本研究で評価の対象としたetlwizはrpcc命令によりユーザモードで内部のクロックカウンタを読みだすことができる.

etlwizにおいてrpcc命令により読み出されるクロックの返り値は32bitのunsigned int型である. クロック1周期当たりの経過時間は約12.89 sec.である. また, Cプログラム上でrpcc命令の呼び出した場合のオーバーヘッドは約0.13 μ sec.であることが調査された.

4. プロファイル

作成したMPIプロファイリングツールの機能について述べる. ここでは, プロファイリングの実行コストを必要最小限に抑えることを最重要視して設計したlevel 0について記述する. level 0におけるプロファイリングデータは, プログラム実行において発行されるMPI手続きの関数名とそれに対するタイムスタンプである. MPICH version 1.1.0にて提供されるプロファイル・インターフェースによりコード化されたプロファイリングツールと, level 0において, MPIの各手続き呼び出し単位のプロファイリングにかかる実行コストを比較した. `MPI_Send()`, `MPI_Recv()` に対するプロファイルのオーバーヘッドは, 従来のツールではそれぞれ4 μsec., 8 μsec.であるのに対し, level 0では1 μsec., 2 μsec.であった. 1 wordの送受信に関する `MPI_Send()`, `MPI_Recv()` の実行時間はそれぞれ142 μsec., 63 μsec.であった. level 0の実行コストは従来のツールに対し約1/4に押えられていることがわかる.

タイムスタンプは各要素プロセッサに備えられたカウンタの計測値をもとに刻まれる. しかしそれらのカウンタは原点が揃っていないため, 各プロセッサ間で共通の時間を得るためにカウンタの時間差を精密に調整し, 大域時間を導入した. 時間差測定はピンポン隣接転送により行った. PE0とPE1間の測定方法を**図2**に示す. メッセージ通信の往路と復路に要する経過時間は等しいと仮定する. まず, PE0からPE1へメッセージを送信し, PE1は受信後直ちにメッセージを返す. このときのPE0における送信命令発行時と受信完了時の中間点と, PE1の受信完了時のカウンタの計測値を比較することにより, 両者のプロセスのカウンタの時間差を得る. データの正当性をはかるため, 図2に示すように, 正方向にピンポン転送した場合とその逆方向に転送した場合との時間差のデータを比較したところ, 両者の値はほぼ同等の数値に収まることが確認された. 更にデータの精度を計るため, このピンポン隣接転送を10回繰り返し, 各試行で得られた結果を比較し計測誤差を調査した. **図3**に例としてPE0とPE1におけるカウンタの時間差を10回測定したときの測定誤差を示す. このグラフは, 2回目の試行で得られたPE0とPE1とのカウンタの時間差「10.99505」を基準値0として, 1回目～10回目の各試行における測定値の計測誤差を示している. 1回目の試行においては, 通信が安定していないために測定値が大きく振れている. しかし2回目以降の試行による測定値に着目すると, 計測誤差は$\pm$ 50 μsec.範囲内に収まっており, 100 μsec.単位の精度が得られている. この計測誤差を踏まえると, ミリ秒あるいはマイクロ秒レベルの精度が期待される.

5. 可視化と性能解析

主要なMPIの手続き関数のオーバーヘッドを調査す

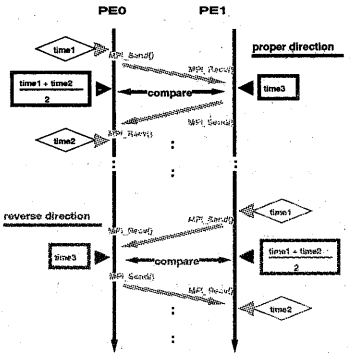

図2 測定方法

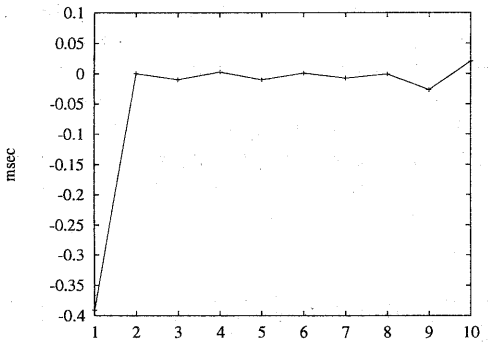

図3 10回の時間差測定における計測誤差

る．プロファイリング情報の可視化ツールである Upshot[4] を用いて視覚的解析を行う．Upshot は MPICH と共に配布されている．実験には4PEを使用し，MPI手続き関数を発行するタイミングを変化させた．メッセージサイズは 1word, 1MB, 4MB の3通りに設定した．

5.1 1対1通信

`MPI_Send()`, `MPI_Recv()` のオーバーヘッドを調査した．PE1 から PE2 への1対1ブロッキング通信を実行し，それぞれの通信命令を発行するタイミングを (A), (B) の2通りに設定し測定を行なった．(A) では `MPI_Recv()` の発行を `MPI_Send()` より 1sec. 遅れて行ない，(B) では逆に `MPI_Send()` の呼び出しを遅らせた．

`MPI_Send()`, `MPI_Recv()` の呼び出しにかかる経過時間を表 2，3 に示す．数値は各試行の最小値である．また diff. の値は `MPI_Send()` の終了時から `MPI_Recv()` の終了時までの時間である．(A) の 1word のケースでは，`MPI_Send()` よりも `MPI_Recv()` の方が早く終了するのでデータは載せていない．

図4，**図5** のどちらにおいても送信呼び出しは対応する受信の実行が開始されるまで完了しないことがわかる．図4 では PE2 において受信呼び出しを発行した時には PE1 から送信されたメッセージはすぐにバッファにコピーされる状態にあるので，待ち時間なく受信を完了することが出来る．よって 図4 の `MPI_Recv()` の経過時間がそのまま `MPI_Recv()` のオーバーヘッドであると考えられる．逆に，`MPI_Send()` のオーバーヘッドは 図5 の経過時間により表される．

表2 ブロッキング通信 (A) の経過時間

	1word	1MB	4MB
MPI_Send	65 μsec	1.02530 sec	1.10210 sec
MPI_Recv	88 μsec	0.02519 sec	0.10239 sec
diff.	-	2223 μsec	0.00809 sec

表3 ブロッキング通信 (B) の経過時間

	1word	1MB	4MB
MPI_Send	113 μsec	0.02352 sec	0.09580 sec
MPI_Recv	1.00050 sec	1.02374 sec	1.09602 sec
diff.	130 μsec	2261 μsec	0.00758 sec

図5 において，`MPI_Send()` を呼び出してから `MPI_Recv()` の終了するまでの経過時間により，理想的なブロッキング通信のオーバーヘッドをはかることができる．メッセージサイズを 1KB, 4KB として実験を行なったところ，ブロッキング通信のオーバーヘッドはそれぞれ 268 μsec, 394 μsec となった．これよりスループットを求めると，メッセージサイズ 1KB のとき 約 3.73 MB/s, 4KB のとき 約 10.14 MB/s であった．

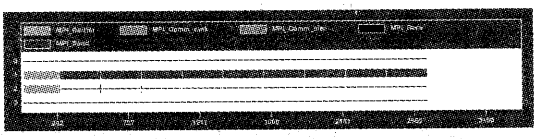

図4 ブロッキング通信 (A) (Message size : 1MB)

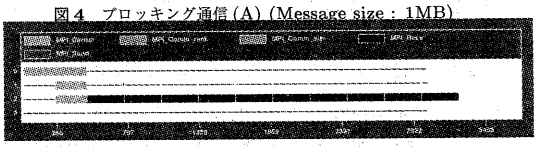

図5 ブロッキング通信 (B) (Message size : 1MB)

5.2 集団通信

以下に3つの集団通信の手続き関数のオーバーヘッドを調査した結果を示す．PE0 ～ PE3 において通信命令を発行するタイミングは (C), (D) の2通りの状態を想定して測定した．(C) は PE0 ～ PE3 においてプログラム上で同じタイミングで手続き関数を呼び出す．(D) は PE0 において手続き関数を呼び出すタイミングを他のプロセスより 1sec. 遅らせたものである．PE1 ～ PE2 における関数呼びだしのタイミングは一定とする．

5.2.1 バリア同期

図6 に`MPI_Barrier()` を 11 回ループさせた結果を示す．バリア同期はグループの全プロセスが `MPI_Barrier()` を呼び出すまで呼びだし側をブロックする．よって `MPI_Barrier()` のオーバーヘッドはグループ内のプロセスの中で最後に呼び出しを完了したプロセスの経過時間となる．図6 より，最初に `MPI_Barrier()` を発行したプロセスが最も経過時間を要することがわかる．表4 は，各ループにおいて経過時間が最大となったプロセスとその計測結果である．11

回のループのうち, 最も経過時間が最小になったのは最後のループにおける 458 μsec. であった. この時の経過時間が `MPI_Barrier()` のオーバーヘッド と言える. 図6 を見ると, 11 回目のループにおいてMPI_Barrier() の開始時間が最も揃っていることがわかる.

また, PE0 において手続き関数を呼び出すタイミングを他のプロセスより 1sec. 遅らせたとき PE0 における経過時間の最小値は 323 μsec. であった. 実装依存であるが, 全プロセスが理想的なタイミングで `MPI_Barrier()` を呼び出すとき, オーバーヘッドはこの程度の値まで抑えることができると予測される.

表4 バリア同期 (C) の最大経過時間

Loop	PE	time
1	PE0	1.75337 sec
2	PE2	2295 μsec
3	PE1	560 μsec
4	PE3	510 μsec
5	PE0	467 μsec
6	PE3	466 μsec
7	PE0	474 μsec
8	PE3	550 μsec
9	PE0	548 μsec
10	PE3	514 μsec
11	PE0	458 μsec

図6 バリア同期 (C)

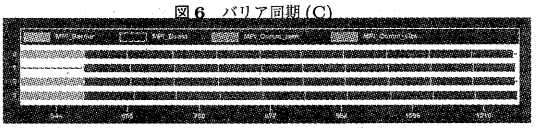

図7 ブロードキャスト (C) (Message size : 1MB)

5.2.2 ブロードキャスト

図7 に PE0 をルートプロセスとし, メッセージサイズを 1MB としたときの `MPI_Bcast()` の振舞いを可視化した結果を示す. PE0 をルートプロセスとし, メッセージサイズは 1MB とした.

`MPI_Bcast()` はルートに指定されたプロセスからグループ内の全プロセスへメッセージをブロードキャストする. 全プロセスによる呼び出しが完了した時点でルートプロセスの通信バッファのメッセージが全プロセスにコピーされている. バリア同期と同様, 計測時間が最大となったプロセスの経過時間が `MPI_Bcast()` のオーバーヘッドとなる. 図7 の場合には 0.198551 sec. となった. メッセージサイズ 4MB のときは 0.80368 sec. であり, メッセージサイズに対しほぼ線形に増加する.

5.2.3 リデュース

メッセージサイズを 1MB として実験を行なったときの `MPI_Reduce()` の振舞いを調査した結果を 図8 , 図9 に示す. PE0 をルートプロセスとした. (D) は PE0 において手続き関数を呼び出すタイミングを他のプロセスより 1sec. 遅らせたものである.

プログラム上で同時に PE0 ~ PE3 の呼び出しを行なう 図8 を見ると `MPI_Reduce()` の実行の挙動は, PE0, PE2 と PE1, PE3 において異なる. これはメッセージをツリー状に収集しているためだと考えられる. PE0 の呼び出すタイミングを他のプロセスより 1sec. 遅らせた 図9 において, 各呼びだしの完了は PE3, PE1, PE2, PE0 の順になっている. PE3 では PE0 において `MPI_Reduce()` が発行される前に呼び出しを完了するが, 一方 PE1 は PE0 の待ち時間が生じることがわかる.

`MPI_Reduce()` はグループ内のプロセスの入力バッファのメッセージに対し演算を施してルートプロセスの出力バッファに返す. よって全プロセスのオーバーヘッドはルートプロセスの経過時間に包隠される. 図9 のように, PE0 における関数呼び出しのタイミングを他のプロセスより 1sec. 遅らせたときの PE0 の経過時間は, メッセージサイズ 1word のとき 105μsec, 1MB のとき 0.29454sec, 4MB のとき 1.23048sec となった.

図8 リデュース (C) (Message size : 1MB)

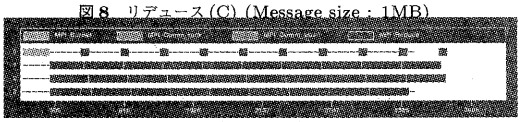

図9 リデュース (D) (Message size : 1MB)

6. etlwiz の性能評価

ここでは, 計算機の演算性能・通信性能を各々独立に評価するのに適した実アプリケーションである QCDMPI を用いた評価を行なう.[5),6)] QCDMPI は, Fortran での記述, MPI を用いたメッセージ通信, SPMD モデルに基づいており, 計算に必要な CPU 時間は格子空間のサイズに比例する.

6.1 QCDMPI のアルゴリズム

4 次元の格子空間 $N = ng1 \times ng2 \times ng3 \times ng4$ をプロセッサ数 $P = np1 \times np2 \times np3 \times np4$ で分割する. 各プロセッサは一斉にそれぞれ割り当てられた空間内の処理を行なう. 各プロセッサに割り当てられる空間サイズは, $\frac{ng1}{np1} \times \frac{ng2}{np2} \times \frac{ng3}{np3} \times \frac{ng4}{np4}$ となり, 計算するリンクの数は空間サイズに次元数 4 を掛けたものとなる.

プログラムのアルゴリズムは以下の通りである.

[step1： **generation**] 各々のプロセッサ対し割り当てられた格子空間内における初期値を設定する.
[step2： **update1**] 行列の加算・乗算を行なう. ここで, 並列実行の場合各次元毎に隣接通信が生じる.
[step3： **update2**] 経路積分を行なう.
[step4： **gather**] 結果を集約する. step2 ～ step4 を反復することにより物理量 (plaquette energy) の近似値を得る[7),8)]. 但し, 反復回数は事前に与える.

step2 において効率を考慮して可能な限り同時処理を行なう. 1 つの次元分割に対し, 1 反復 12 回の通信を必要とする. 例えば 4 次元分割の通信回数は 12 回 × 4 次元となる. 低次元分割の場合, 通信の頻度は低いが, 1 通信当たりの転送量は増す. 高次元分割により転送量を軽減できるが, 通信回数は増加する. 計算機の特性に合う次元分割の選択により, 実行時間の短縮化が望める. ここでは, 最適な分割方法の検討により, 演算 / 通信性能の評価を行なう. プロセス数 (16PE) と問題サイズを一定 (問題サイズ: 16^4) に揃え, 各次元の通信におけ特性の差が反映されるようなに問題を設定した. PE 数 16 とした時の各次元分割における分割方法を表 5に示す.

表 5　QCDMPI における次元分割方法 (16PE)

分割方法		分割方法	
1 次元	16 × 1 × 1 × 1	2 次元	4 × 4 × 1 × 1
3 次元	4 × 2 × 2 × 1	4 次元	2 × 2 × 2 × 2

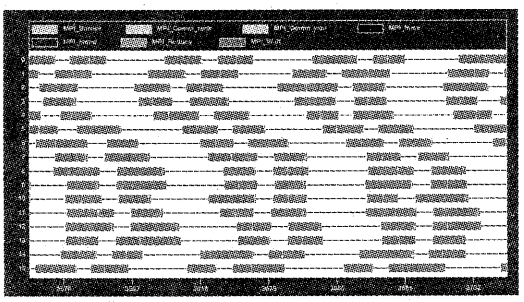

図 10　QCDMPI の 1 次元分割結果

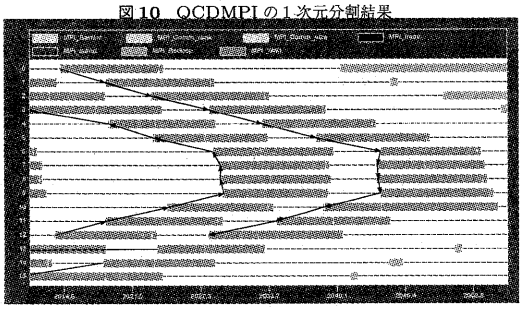

図 11　QCDMPI の 1 次元分割結果 (拡大図)

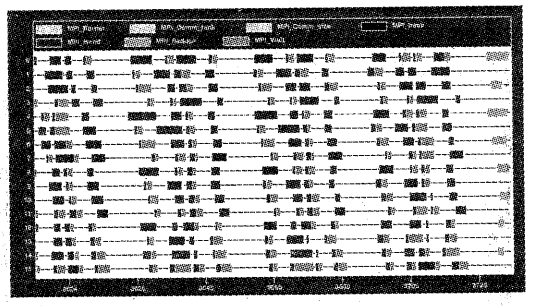

図 12　QCDMPI の 2 次元分割結果

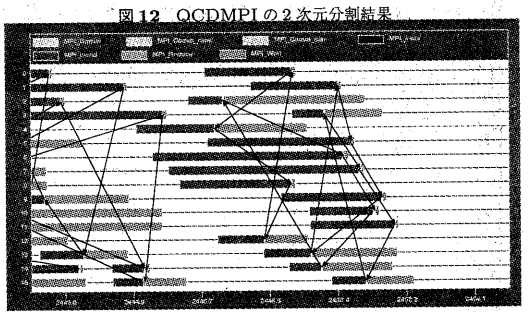

図 13　QCDMPI の 2 次元分割結果 (拡大図)

6.2　プロファイリングツールによる評価

図 10 ～ **図 17** に各次元分割における通信状態を示す. それぞれの実行の一部とさらにその拡大図を示す. 拡大図には **MPI_Isend()** の呼びだし完了時から **MPI_Irecv()** の呼びだし完了時に向けて矢印を施した. ノンブロッキング通信であるため, 時間軸に逆行する場合もあり得る.

全体を見ると, 高次元分割になるほど各通信時間が低減しており, 転送量が減少していることがわかる. 一方, 通信の頻度は高次元分割の方が高い.

拡大図を見ていくと, 1, 2 次元分割では全体又はグループで通信がなされているのに対し, 3, 4 次元分割では 1 対 1 のエクスチェンジが主である.

図 13 , 15 , 17より, 矢印が時間軸に逆行するとき, つまり **MPI_Isend()** より前に **MPI_Irecv()** が発行されているときには, 送信側において, その直後の **MPI_Wait()** の経過時間が比較的短いことがわかる. また **図 15** , 17 のエクスチェンジにおいては, **MPI_Isend()** の発行されるタイミングに関わらず, 常にランクの高いプロセッサの方が **MPI_Isend()** の経過時間が早い.

全体として, 1 次元分割のときプログラム中で同じシフトのパターンを繰り返しており, 非常に規則的な通信を実行していることがわかる. これらは PE6-9 による遅延のために穏やかな同期がとれているためだと考えられる. また, 4 次元分割のとき各通信の開始時点が揃っており比較的効率の良い通信が実行されている.

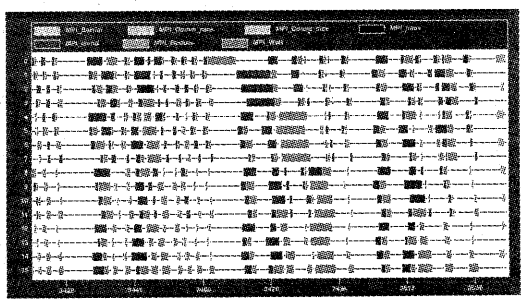

図14 QCDMPIの3次元分割結果

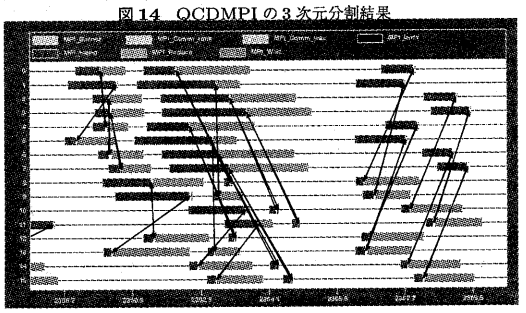

図15 QCDMPIの3次元分割結果(拡大図)

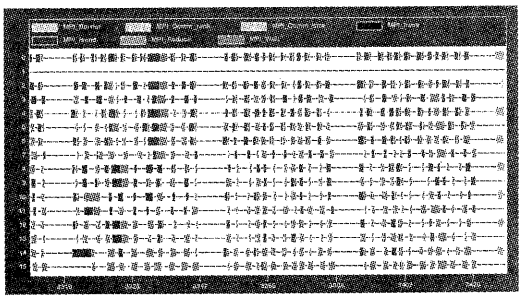

図16 QCDMPIの4次元分割結果

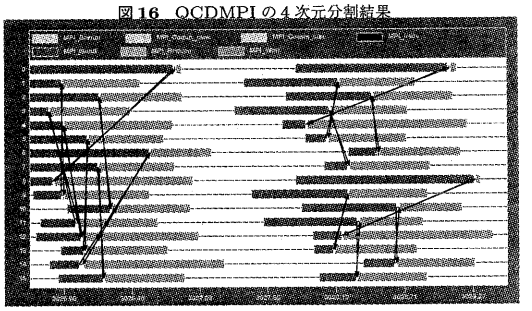

図17 QCDMPIの4次元分割結果(拡大図)

7. まとめ

本研究では,並列システムの特徴抽出を支援するためのMPIプロファイリングツールを設計・構築し,実際に作成したツールを用いて並列システムの性能評価を行った.作成したプロファイリングツールでは収集するプロファイリング情報を必要最小限に押え,実行コストの軽いプロファイリングを実行できる環境を提供している.また,作成したツールでは,高精度なプロファイル情報を提供するために,性能測定にはクロックカウンタを用いた.これによりミリ秒,或いはマイクロ秒単位の精度を得ることができた.さらに,要素プロセッサの持つそれぞれのカウンタの時間差を精密に調整して,大域時間により各プロセッサの実行状態を比較することで高精度かつ信頼性の高い性能情報を提供することが可能となった.

またプロファイリングツールを用いて実際にワークステーションクラスタの性能情報を収集し,幾つかの主要なMPIの手続き関数のオーバーヘッドを調査した.更にQCDMPIの実行に関してプロファイリングを行いetlwizの性能評価を実施した.実行状態の可視化により,プログラムのアルゴリズムの解析を行なわなくとも通信効率の優劣を容易に把握することができた.MPIプロファイリングツールを用いることにより,これまで困難とされていた詳細なパフォーマンスの分析を比較的容易かつ精密に行なうことが可能となった.

謝辞 QCDMPIを御提供頂いた帝塚山大学の日置慎治助教授と,議論の場として,TEA研究グループの皆様に深く感謝致します.

なお,本研究は工業技術院国際特定共同研究「ハイパフォーマンスコンピューティングシステム性能評価技術の研究」ならびに電子技術総合研究所とお茶の水女子大学の共同研究に基づくものである.

参考文献

1) Message Passing Interface Forum. "MPI-2: Extensions to the Message-Passing Interface". July 1997. http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html
2) 日置慎治, http://insam.sci.hiroshima-u.ac.jp/QCDMPI
3) http://www.rwcp.or.jp/lab/pdperf/timer-collection/
4) Virginia Herrarte and Ewing Lusk. Studying parallel program behavior with upshot. Technical Report ANL-91/15, Argonne National Laboratory, Argonne, IL 60439, 1991
5) 益口摩紀, 長嶋雲兵, 関口智嗣, 佐藤三久. "量子色力学プログラムを用いた並列計算機の性能評価". 情報処理学会研究報告 HPC-67-8, pp.43-48. 1997.
6) 益口摩紀, 建部修見, 関口智嗣, 佐藤三久, 長嶋雲兵. "アルファワークステーションのクラスタ etlwizの性能評価". 情報処理学会研究報告 HPC-70-11 pp.61-66. 1998.
7) 原康夫. 量子色力学とは何か, 丸善 (1986).
8) 南部陽一郎. クォーク 素粒子物理の最前線, 講談社 (1981).