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

2004－HPC－98 (4)
2004／4／13

MPIプログラムにおける大規模な実行履歴生成の検討

置田真生† 伊野文彦† 萩原兼一†

本稿では，大規模な実行履歴を生成するために，実行履歴の記憶量の増大に耐性をもつ生成手法を提案する．提案手法は，主記憶上の実行履歴の記録量を常に主記憶容量以下に保つため，プログラム実行中に実行履歴を主記憶からディスクへ待避する．提案手法の主な貢献は，主記憶容量を越える大規模な実行履歴の生成に伴うオーバヘッドを削減することである．提案手法を主記憶容量を越える実行履歴を生成するプログラムに対して適用した結果，オーバヘッドを約37分の1に削減し，大規模な実行履歴を効率よく生成できた．

Preliminary Study on Generating Large-Scale Trace Files for MPI Programs

Masao Okita,† Fumihiko Ino† and Kenichi Hagihara†

In this paper, we propose a robust method for generating large-scale trace. Our method removes trace from main memory to disk during program execution in order to maintain trace size on main memory below at memory capacity. The key contribution of our method is reduction of overheads for generating large-scale trace that exceeds memory capacity. The experimental result shows that our method reduces the overheads to 1/37 and generates large-scale trace efficiently.

1. はじめに

高水準通信仕様 MPI[1] を用いた並列プログラム（MPI プログラム）などのメッセージ通信型並列プログラムの性能を解析および改善するために，実行履歴を生成する計測ツールがある．これらのツールはプログラム実行時に性能に関するデータ（性能データ）を計測し実行履歴としてディスクに記録する．性能データの例として，処理時間および主記憶の参照値，キャッシュミス率などがある．

既存の計測ツールには gprof[2] や PAPI[3]，MPE[4] や Vampirtrace[5]，TAU[6] などが存在する．gprof および PAPI は計算処理の性能計測に，MPE は通信処理の性能計測に有用である．TAU および Vampirtrace は，通信および計算処理に関する多様なデータを計測し，視覚化および解析できる．また，TAU が生成する実行履歴は VAMPIR[5] など他のツールで利用できる．

これらの計測ツールを用いて実行履歴を生成するとき，性能摂動（Performance Perturbation）と呼ばれる計測オーバヘッドが発生する．このオーバヘッドは性能データの計測および実行履歴の記録に要する時間からなる．このうち実行履歴の記録において，多くのツールはプログラム実行中は主記憶上に蓄積し，プログラム終了後に主記憶上の実行履歴をディスクへ記録する．

実行履歴を主記憶に蓄積する計測手法は，実行履歴の記録量増大に対する耐性が低い．実行履歴の記録量が増大し，主記憶容量を越えるとメモリスワップ（スワップ）が発生する．スワップが発生すると，ディスクへのスワップアウトもしくはページアウトによって実行履歴の記録に伴うオーバヘッドが増大する．このオーバヘッドの増大が原因となり，プログラムの総実行時間の増大および実行状況の変化の2つの問題が生じる．総実行時間が増大すると，実行履歴の生成に要する時間が増大する．また，実行状況が変化すると，プログラムの本来の性能を実行履歴から得られないため，正確な性能ボトルネックを特定できない．

そこで我々は，大規模な実行履歴を生成するために，記憶量の増大に対して耐性をもつ手法を提案する．提案手法は，実行履歴の記録量を常に主記憶容量以下に保つため，プログラム実行中に実行履歴をディスクへ待避する．ディスクへの待避によってスワップの発生を回避することで，総実行時間の増大および実行状況

† 大阪大学大学院情報科学研究科コンピュータサイエンス専攻
Department of Computer Science, Graduate School of Information Science and Technology, Osaka University

の変化を防ぐ．ただし，ディスクへの待避に伴って新たなオーバヘッドが生じるため，このオーバヘッドがプロセス間で偏らないようにディスクへの待避を行う．

本稿では，まず2章で問題点および提案手法を示すための諸定義について説明し，大規模な実行履歴の生成における問題点を示す．次に，3章で問題点を解決するための提案手法について説明する．その後，4章で記録量増大への耐性およびキャッシュを利用した高速計算への影響を調べるために提案手法を評価した実験の結果を示す．

2. MPIプログラムにおける性能計測

2.1 諸定義

MPIプログラム S のソースコードを n 個の文のならび $s_1, s_2, ..., s_n$ と表記する．S の実行履歴を生成するためのプログラム S_i は $s_1, i_1, s_2, i_2, ..., s_n, i_n$ と表記できる[7]．ここで $i_1, i_2, .., i_n$ は性能データ取得および実行履歴生成のための文を示し，計測が不要な箇所では空である．

S_i を N 個のプロセスで並列実行したとき，プロセス p で得られる実行履歴をイベントのならび $e_{p,1}, e_{p,2}, ..., e_{p,m}$ と定義する．$e_{p,j}$ は，p における j 番目のイベントを表す．イベントは文の実行インスタンスを表し，実行時刻やメモリの参照値などを性能データとして情報に持つ．

S の性能解析は，S を実行したときの実行状況 E_p に対して行う．E_p を各イベントの処理時間のならびと定義し，E_p は式 (1) のように表す．

$$E_p = \{(w_{p,1}, t_{p,1}), ..., (w_{p,m}, t_{p,m})\} \quad (1)$$

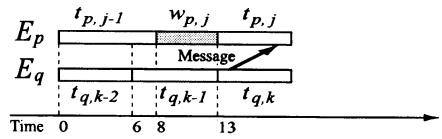
ここで，$(w_{p,j}, t_{p,j})$ は $e_{p,j}$ に対応し，$w_{p,j}$，$t_{p,j}$ は各々 $e_{p,j}$ の実行前に発生した通信待ち時間および $e_{p,j}$ の実行に要した時間を表す．$w_{p,j}$ は $e_{p,j}$ が計算命令ならば常に0秒であり，通信命令ならば通信待ち時間である．$t_{p,j}$ は $e_{p,j}$ に対応する文 s_k の実行時間である．

しかし，実行履歴に記録される実行状況は，S_i を実行したときの実行状況 E'_p である．E'_p は式 (2) のようになる．

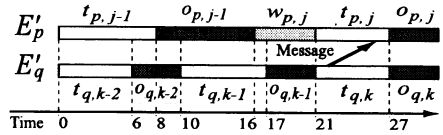
$$E'_p = \{(w_{p,1}, t_{p,1}, o_{p,1}), ..., (w_{p,m}, t_{p,m}, o_{p,m})\} \quad (2)$$

$o_{p,j}$ はイベント $e_{p,j}$ に対する計測オーバヘッドを表す．$o_{p,j}$ は文 i_k の処理時間であり，性能データの取得および記録に要する時間の2つからなる．

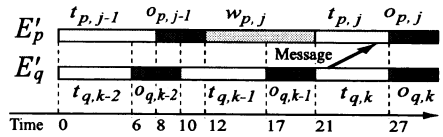
なお，E_p および E'_p を図示すると各々図1のようになる．図1では横軸は時間軸を，縦軸はプロセス軸を示す．時間軸沿いの数字は時刻を表す．図1はプロセス p および q が同期通信を行う例であり，$w_{p,j} + t_{p,j}$ および $t_{q,k}$ が各々受信イベント $e_{p,j}$ および送信イベント $e_{q,k}$ の処理時間を表す．$w_{p,j}$ 以外の待ち時間は全て0であるため，図1では省略する．



(a) 本来の実行状況



(b) 計測時の実行状況（実行状況の変化なし）



(c) 計測時の実行状況（実行状況の変化あり）

図1 計測オーバヘッドによる実行状況の変化

2.2 計測オーバヘッドによる実行状況の変化

MPIプログラムの特徴として，実行履歴に記録される実行状況が変化する可能性が挙げられる．実行状況の変化は次のように定義する．E'_p における $w_{p,j}$ および $t_{p,j}$ $(1 \leq j \leq m)$ の少なくとも1つの値が，E_p における値と異なるとき，実行状況は変化している．

実行状況が変化する原因は，E'_p における通信待ち時間が変化するためである．その例を図1(a) および (c) に示す．図1では，計測時 (c) における待ち時間 $w_{p,j}$ が本来の実行状況 (a) と比較して4増大している．この原因は，通信するイベントの組 $e_{p,j}$ および $e_{q,k}$ の開始時刻の差異が増大したためである．(a) における開始時刻の差異5に対して，(c) における差異は9に増加する．

開始時刻の差異が増大する原因は，プロセス間における計測オーバヘッドの偏りである．図1(c) における $e_{p,j}$ および $e_{q,k}$ の開始時刻は，各々(a) と比較して4および8遅延する．この遅延は，各々のイベントの開始前に生じたオーバヘッドの総和 $o_{p,j-1}$ および

$o_{q,k-2}+o_{q,k-1}$ に等しい．したがって，開始時刻の遅延は，開始前に生じたオーバヘッドの総和で決まる．プロセス間でオーバヘッドの総和が異なる場合，開始時刻の遅延も異なるため開始時刻の差異は変化する．一方，図 1(b) のように通信処理以前に発生したオーバヘッドの総和が等しい場合，開始時刻の差異は変化しない．したがって，$w_{p,j}$ も変化しない．

以上から，実行状況の変化を防ぐためには S_i の実行時に次の条件 C を満たせばよい．

条件 **C**：互いに通信するイベントの組 $e_{p,j}, e_{q,k}$ すべてに対して式 (3) が成り立つ．

$$\Sigma_{l=1}^{j-1} o_{p,l} = \Sigma_{l=1}^{k-1} o_{q,l} \qquad (3)$$

条件 C を満たすためには，S_i に挿入する計測のための文 $i_1,..,i_n$ を工夫する必要がある．発生する計測オーバヘッドを予測して，プロセス間で計測オーバヘッドの総和が等しくなるよう $i_1,..,i_n$ を決定する．

2.3 大規模な実行履歴の生成における問題

主記憶上に実行履歴を蓄積する場合，記録量が主記憶容量を越えるとスワップが発生する．このスワップが原因となって次の 2 つの問題が生じる．S_i の総実行時間の増大，および実行状況の変化である．

ある $e_{p,j}$ を記録領域に追加するときスワップが発生すると，$o_{p,j}$ の値は増大する．さらに，$j+1$ 番目以降のすべてのイベントにおいても，$e_{p,j}$ と同様にスワップが発生する．この理由は，主記憶上の実行履歴の記憶量がすでに主記憶容量を越えているためである．したがって，$o_{p,l}$ $(l=j..m)$ のすべてが増大する．$o_{p,l}$ の増大量は，本来の計測オーバヘッドおよびイベントの実行時間と比較して数十倍大きい．そのため，S_i の総実行時間は大幅に増大する．実際のプログラムにおける増大量は 4.1 節で示す．

また，スワップが発生すると実行状況が変化する可能性がある．スワップが発生するか否かはプロセスごとに決定される．この理由は，プロセスごとに主記憶上に蓄積する実行履歴の記録量が異なるためである．このため，あるプロセスでスワップが発生しても他のプロセスでは発生しない状況が存在する．この場合，プロセスごとに計測オーバヘッドの増大量が異なる．したがって，本来ならば条件 C を満たす S_i が，スワップが発生すると条件 C に反する可能性がある．条件 C に反する場合，E'_p における通信待ち時間が変化する．これは，プログラムの性能解析において，例えば性能ボトルネックを特定する場合などに問題となる．

```
// τ：中断時間
1: function trace_swapout() begin
2:   t_s := MPI_Wtime()
3:   write_trace() // 主記憶上の実行履歴をディスクへ記録
4:   clear_trace() // 主記憶上の実行履歴を消去
5:   t_e := MPI_Wtime()
6:   while t_e − t_s < τ begin
7:     t_e := MPI_Wtime()
8:   end
9: end
```

図 2　待避関数の実装

3. 提案手法

提案手法は，スワップの発生を防ぐことで，主記憶容量を越える大規模な実行履歴を効率的に生成する．一般に，主記憶へのは記録速度はディスクと比較して速いため，実行履歴は主記憶上に蓄積される．しかし，主記憶上に蓄積した実行履歴は実行中に参照されない．したがって，実行履歴を主記憶からディスクへ待避することが可能である．そこで提案手法では，実行履歴の記録量を常に主記憶容量以下に保つよう，プログラム実行中に実行履歴をディスクへ待避する．ただし，実行履歴の待避に伴うオーバヘッドが新たに発生するため，実行状況が変化しないよう適切なタイミングで待避する必要がある．

提案手法では，既存の S_i に待避のための関数 f を挿入したプログラム S_f を用いて実行履歴を生成する．f の挿入位置は，次の 2 点を満たすように決定する．

P1：実行履歴の記憶量が主記憶容量を越えない

P2：f の実行により生じるオーバヘッドが条件 C を満たす

P1 は，スワップの発生を防ぐために必要である．P1 を満たすために，実行履歴の記憶量が一定量に達するたびに f が実行されるよう挿入する．P2 は，f のオーバヘッドによる実行状況の変化を防ぐために必要である．P2 を満たすために，f の実行時間が一定になるよう f を実装し，全プロセスで f の実行回数が等しくなるよう f を挿入する．

P1 および P2 を満たす挿入位置の決定は，ソースコードに対して静的に決定する手法およびプログラム実行時に動的に決定する手法がある．動的に決定する手法では，挿入位置を実行中に決定することで新たなオーバヘッドが生じる．このオーバヘッドも含めて P2 を満たす挿入位置を決定する必要があるため，処理が煩雑になる．そこで提案手法では，プログラマが挿入位置を静的に決定し，ソースコード中に明示する．P1 および P2 を満たす挿入位置の例として，S_i 内の各

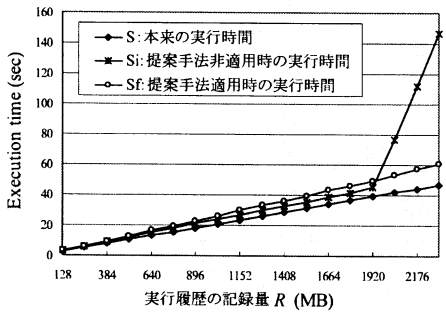


図 3 ピンポンプログラムの実行時間

MPI 集合通信の直後に挿入するパターンが挙げられる．他の挿入パターンの整理および検討は，今後の課題である．

待避関数 f の実装例を図 2 に示す．f を実行すると，まず MPI の時刻カウンタ MPI_Wtime() を用いて開始時刻を取得し（2 行目），記憶領域の実行履歴をディスクに記録する（3 行目）．次に，主記憶上の実行履歴を消去する（4 行目）．P1 を満たしている場合，3〜4 行目の処理によって主記憶上に存在する実行履歴の記録量は常に主記憶容量以下になるため，スワップは発生しない．その後，時間 τ が経過するまで空ループを続けて待機する（5〜8 行目）．待機する理由は，f の実行時間をプロセス間で一定にすることで P2 を満たすためである．プロセスごとに主記憶上に持つ実行履歴の記録量が異なる場合でも，τ を十分大きくすることで f の実行時間は一定になる．

なお，提案手法の欠点として，キャッシュミスの発生が挙げられる．プログラム実行中に実行履歴をディスクへ待避するため，本来生じないはずのキャッシュミスが生じる可能性がある．このキャッシュミスによる計算性能の低下については 4.2 節で述べる．

4. 実　　験

本章では，以下の 2 つの適用実験の結果を示す．

実験 1：記録量増大への耐性を確認する．2 つのプロセス間で送受信を繰り返すプログラムを対象に，実行履歴の記録量が主記憶容量を越えたときの計測オーバヘッドの増大量を示す．また，提案手法が計測オーバヘッドの増大を回避できることを確認する．

実験 2：実行履歴の待避がキャッシュを利用した高速計算に及ぼす影響を調べる．キャッシュを利用して超線形性能を示す行列積演算プログラムに提案手法を適用し，性能の変化を調べる．

表 1 ピンポンプログラムの実行時間および実行時間の差異

記録量 (MB)	実行時間（秒）			S との差異	
	S	S_i	S_f	S_i	S_f
128	2.65	3.11	3.09	0.46	0.44
1920	39.06	45.36	49.41	6.30	10.35
2304	46.94	146.99	60.77	100.05	14.83

なお，MPI プログラムの実行には，16 台の PC をミリネットネットワーク[8] で相互接続したクラスタを用いた．各 PC は Pentium III 1GHz の CPU（2 次キャッシュ 256KB）および 2GB の主記憶を搭載している．MPI 実装には MPICH-SCore[4] を用いた．

4.1 記録量増大への耐性の確認

2 台のプロセス p および q で，送受信を繰り返すピンポンプログラムを実行する．まず，p が MPI 送信命令 MPI_Send() を実行してデータを送信し，q が MPI 受信命令 MPI_Recv() を実行して受信する．その後 q は受信したデータをただちに p へ送信し，p がこれを受信する．以上の処理を 1 回のピンポンとして繰り返す．

元のプログラム S に対して，計測プログラム S_i では各 MPI_Recv() の直後に計測のための文を挿入した．MPI_Recv() の実行時刻および通信データの内容を実行履歴に記録する．また，提案手法を適用したプログラム S_f では，ピンポンを 4096 回繰り返すたびに待避関数 f を実行するよう f を挿入した．この挿入箇所は P1 および P2 を満たしている．通信データサイズを 128KB に固定すると，実行履歴の記憶量 R（MB）が 512MB に達するたびに f が実行される．なお，提案手法における待避時間 τ は 1 秒とした．この時間は，ディスクへの書き込みに要する時間より十分大きい．

図 3 にピンポン回数を 1024 回から 18432 回まで，すなわち R の値を 128MB から 2304MB まで変化させたときの S および S_i，S_f の総実行時間を示す．図 3 は横軸が生成した実行履歴の記憶量 R，縦軸がプログラムの総実行時間を表している．また，表 1 に $R = 128$ および 1920，2304 における総実行時間および差異を示す．S_i および S_f に対する S の実行時間との差異は，各々 S_i および S_f の実行において発生した計測オーバヘッドの総和を表す．

S_i の実行時間について，図 3 および表 1 から，次の 2 点がわかる．1 点は実行時間が R に対して線形に増大すること，もう 1 点は $R > 1920$ において実行時間の増大率が増加することである．S_i の実行時間の内訳は，S の実行時間および計測オーバヘッドである．S の実行時間は，図 3 が示すように R に比例し

```
// M：行列サイズ
// A, B：M × M 行列
// L：一度にブロードキャストする列数
// F：実行履歴の待避を行う頻度
 1: foreach j := 1 to M begin
 2:   if j mod L = 0 then begin
 3:     MPI_Bcast(...)  // B の j 列から j + L 列までを放送
 4:     measure_and_trace(...) // 実行履歴の生成
 5:     if j mod F = 0 then trace_swapout() // 待避関数 f
 6:   end
 7:   foreach i := 1 to M/N
 8:     foreach k := 1 to M
 9:       C(i, j) = C(i, j) + A(i, k) · B(k, j)
10: end
```

図 4　行列積演算プログラムの概要

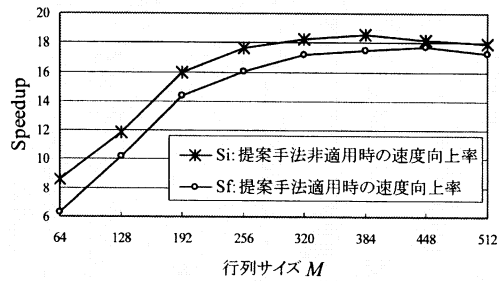


図 5　行列積計算を 16 台で実行した場合の速度向上率

て増大する．また，計測オーバヘッドは，実行履歴を主記憶に蓄積するための memcpy() の実行時間であるため，R に比例して増大する．したがって，R に比例して S_i の実行時間は増大する．

$R > 1920$ における S_i の実行時間の増大率の増加は，スワップの発生が原因である．$R = 1920$ を越えると，OS および他のプロセスが使用する主記憶量との合計が 2GB を越えるため，スワップが発生する．スワップが発生すると，memcpy() の実行時間が増大するため計測オーバヘッドが増大する．表 1 から実行履歴 1MB の生成ごとに発生する計測オーバヘッドを計算すると，$R \leq 1920$ においては $(6.30 - 0.46)/(1920 - 128) = 3.26 \times 10^{-3}$ である．$R > 1920$ においても同様に計算すると $(100.05 - 6.30)/(2304 - 1920) = 2.44 \times 10^{-1}$ である．したがって，スワップの発生によって，計測オーバヘッドは約 74.8 倍に増加する．

S_f の実行時間について，図 3 から次の 2 点がわかる．1 点は，$R > 1920$ においても実行時間の増大率が変化しないこと，もう 1 点は $R = 1920$ を境目に S_i の実行時間との大小関係が逆転することである．提案手法では，プログラムの実行中に実行履歴をディスクへ待避するため，スワップが発生しない．そのため，S_i の計測オーバヘッドは R の値に関わらず $(14.83 - 0.44)/(2304 - 128) = 6.61 \times 10^{-3}$ で一定である．したがって，S_i と異なり $R > 1920$ においても実行時間の増大率は変化しない．

また，S_f の実行時間は $R \leq 1920$ においては S_i と比較して大きい．この理由は，待避関数 f の実行時間が計測オーバヘッドに含まれるためである．しかし，f の実行による計測オーバヘッドは，スワップの発生によるオーバヘッドと比較すると小さい．したがって，$R > 1920$ においては，S_f の実行時間は S_i と比較して小さくなる．$R > 1920$ における計測オーバヘッドを比較すると，S_f の計測オーバヘッドは S_i の約 36.9 分の 1 である．したがって，記録量が主記憶容量を越えるような大規模な実行履歴の生成において，提案手法は効率がよい．

4.2　キャッシュを利用した高速計算への影響

$M \times M$ 行列 A および B を入力とし，行列積 $A \cdot B = C$ を演算するプログラムを並列実行する．並列化にあたり，行列 A および B を各々行方向および列方向にブロック分割し各プロセスに分散する．実装の概要を図 4 に示す．MPI 集合通信 MPI_Bcast() を用いて B のサブブロックを L 列ずつ放送し（2〜3 行目），各プロセスが保持する A のサブブロックとの積を演算する（7〜9 行目）．

元のプログラム S に対して，S_i では各 MPI_Bcast() の直後に計測のための文を挿入した（図 4 の 4 行目）．MPI_Bcast() の終了時刻を実行履歴に記録する．また，提案手法を適用したプログラム S_f では，MPI_Bcast() を実行した直後に待避関数 f を挿入する（図 4 の 5 行目）．この挿入箇所は P1 および P2 を満たす．今回の実験では，MPI_Bcast() を 16 回実行するたびに実行履歴を待避するよう，$F = 16$ とした．16 回実行したときの実行履歴の記録量は 128B と小さいため，本来ならば 16 回ごとに実行履歴を待避する必要はない．しかし，今回の実験では実行履歴の待避を頻繁に行ったときの影響を調べるため，$F = 16$ とした．なお，待避時間 τ は 4.1 と同様に 1 秒とした．

図 5 に S_i および S_f をプロセス数 $N = 16$ で実行したときの速度向上率（Speedup）を示す．速度向上率は，使用したプロセス台数に対して得られる並列効果を表し，$N = 1$ の場合と比較した実行時間の倍率である．図 5 では，横軸および縦軸が各々行列サイズ M および速度向上率を示す．

図 5 から次の 2 点がわかる．1 点は S_i および S_f ともに M が大きい場合には超線形の速度向上率を示す

こと，もう1点は M が同じならば S_f の速度向上率は S_i と比較して小さいことである．

図5が示すように，S_i および S_f の速度向上率は各々 $M \geq 192$ および $M \geq 256$ において16を上回る．実行したプロセス数 N 以上の速度向上率を示しており，超線形の性能を得た．この理由は，3つの行列（A および B，C）のデータサイズの総和が単一CPUの2次キャッシュのサイズ256KBを越えているためである．$M \geq 148$ において3つの行列のデータサイズが256KBを越える．256KBを越えると，$N = 1$ のときすべてのデータが2次キャッシュに載らないため計算速度は遅い．一方，$N = 16$ のときはデータを複数のCPUに分散するため，各々のもつデータはすべてキャッシュに載る．また，$M = 64$ のときの速度向上率は S_i および S_f ともに N の半分以下である．この原因は，行列サイズが小さく計算量が少ないためである．計算量が少ないと通信処理がボトルネックとなってプログラムの速度向上率は悪化する．

M の値が同じとき，S_f の速度向上率は S_i と比較して小さい．特に $M = 192$ のときは，S_i は超線形性能を示すが，S_f は N 以下の速度向上率を示す．この理由は，実行履歴の待避処理がキャッシュの内容を更新するためと考えられる．キャッシュの内容を更新すると，待避処理後に本来発生しないキャッシュミスが発生し，計算速度が遅くなる．実際にキャッシュミスが発生したか否かの確認，および発生した場合のペナルティの大きさの調査は今後の課題である．

このように，提案手法はキャッシュを利用した高速計算の性能を低下させる．しかし，性能が低下した場合でも，M が十分大きければ超線形性能を得られた．

5. まとめ

本稿では，MPIプログラムを対象に，主記憶容量を越える大規模な実行履歴を効率的に生成する手法を提案した．提案手法は，実行履歴の記録量を常に主記憶容量以下に保つため，プログラム実行中に主記憶上の実行履歴をディスクへ待避する．提案手法はメモリスワップの発生を回避し，実行時間の増大および実行状況の変化を防ぐ．適用実験において，提案手法は実行履歴の記憶量が主記憶容量を越えるときの計測オーバヘッドを約36.9分の1に削減できた．また，提案手法はキャッシュを利用した高速計算の性能を低下させるが，計算量が十分大きい場合には超線形性能を得られることを示した．

今後の課題としては，待避処理の挿入箇所の検討および提案手法がキャッシュを利用した高速計算の性能を低下させる原因の解析が挙げられる．

謝辞　本研究の一部は，科学研究費補助金基盤研究(C)(2)(14580374)，若手研究(B)(15700030)およびNECネットワーク開発研究本部の補助による．

参考文献

1) Message Passing Interface Forum: MPI: A Message-Passing Interface Standard, *Int'l J. Supercomputer Applications and High Performance Computing*, Vol. 8, No. 3/4, pp. 159–416 (1994).
2) Graham, S. L., Kessler, P. B. and McKusick, M. K.: gprof: a Call Graph Execution Profiler, *Proc. SIGPLAN Symp. Compiler Construction (SCC'82)*, pp. 120–126 (1982).
3) London, K., Dongarra, J., Moore, S., Mucci, P., Seymour, K. and Spencer, T.: End-user Tools for Application Performance Analysis Using Hardware Counters, *Proc. 14th Int'l Conf. Parallel and Distributed Computing Systems (PDCS'01)* (2001).
4) O'Carroll, F., Tezuka, H., Hori, A. and Ishikawa, Y.: The Design and Implementation of Zero Copy MPI Using Commodity Hardware with a High Performance Network, *Proc. 12th ACM Int'l Conf. Supercomputing (ICS'98)*, pp. 243–250 (1998).
5) Nagel, W. E., Arnold, A., Weber, M., Hoppe, H.-C. and Solchenbach, K.: VAMPIR: Visualization and Analysis of MPI Resources, *The J. Supercomputing*, Vol. 12, No. 1, pp. 69–80 (1996).
6) Shende, S. and Malony, A. D.: Integration and application of TAU in parallel Java environments, *Concurrency and Computation: Practice and Experience*, Vol. 15, No. 3/5, pp. 29–39 (2003).
7) Malony, A.D. and Reed, D.A.: Models for Performance Perturbation Analysis, *Proc. Workshop on Parallel and Distributed Debugging (WPDD'91)*, pp. 15–25 (1991).
8) Boden, N.J., Cohen, D., Felderman, R.E., Kulawik, A.E., Seitz, C.L., Seizovic, J.N. and Su, W.-K.: Myrinet: A Gigabit-per-Second Local-Area Network, *IEEE Micro*, Vol. 15, No. 1, pp. 29–36 (1995).