メッセージ通信プログラムにおける性能解析のための大規模な実行履歴の生成

置田真生[†,☆] 伊野文彦[†] 萩原兼一[†]

本稿では，大規模な並列プログラムの性能を正確に解析するために，実メモリの容量を超えて大規模な実行履歴を生成する手法を提案する．提案手法の特長は，正確な性能計測の障害となる性能摂動を小さく保ったまま，大規模なプログラムの実行履歴を得られる点にある．提案手法は，プログラムの実行中にバリア同期を利用して実行履歴をメモリからディスクへ退避し，メモリスワップを回避することで性能摂動の増大を防ぐ．この際，退避のためのオーバヘッドを全プロセスで均等にし，実行の復元を容易にする．ユーザは，実行履歴から退避のためのオーバヘッドを除去することでプログラムの実行を容易に復元でき，プログラムの性能を解析できる．提案手法を用いて実メモリの容量を超える実行履歴を生成した結果，提案手法を用いない場合と比較して性能摂動による実行時間の増大を 4 分の 1 以下に削減できた．また，性能摂動を含まない本来の実行時間を約 1.8%の誤差で復元できた．

Large-scale Trace Generation for Performance Analysis of Message Passing Parallel Programs

Masao Okita,[†,☆] Fumihiko Ino[†] and Kenichi Hagihara[†]

In this paper, we propose a robust instrumentation method that is capable of generating large traces beyond the capacity of physical memory, aiming at analyzing the performance of large-scale parallel programs. The key contribution of our method is that the generation of large traces with decreasing performance perturbation. In order to prevent increasing perturbation caused by memory swap, our method pushes trace from the main memory to a disk during program execution. The overheads for our method are equalized among all processes, making it easy to reconstruct the original execution. The experimental results show that our method reduces overheads for generating large-scale trace to 1/4. It also allows us to reconstruct the original execution time within an error of 1.8%.

1. はじめに

メッセージ通信仕様 MPI [1] を用いた並列プログラム（MPI 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[2)~5)] 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記録する．多くのツールは，実行履歴の生成オーバヘッドを削減するために，プログラム実行中は実行履歴をメモリ上に蓄積し，プログラム終了後にメモリ上の実行履歴をディスクへ記録する．この実行履歴を解析することで，性能ボトルネックの特定および性能予測，可視化などを実現する．

一般に，実行履歴を生成するとき，性能データをメモリへ蓄積するための文（履歴生成文）を対象プログラムに挿入する．この履歴生成文を実行することが原因で，挿入前と比較してプログラムの実行時間や通信の受信順序が変化する．このことを性能摂動（Performance Perturbation）[6] と呼ぶ．したがって，プログラムの本来の性能を解析するためには，実行履歴から性能摂動を除去して本来の実行を復元する必要がある．

性能摂動は，その除去の容易さの観点から，直接摂動および間接摂動に分類できる．直接摂動は，履歴生成文そのものの実行による処理時間の増大を表す．ゆえに，その発生箇所はソースコードから静的に特定でき，その大きさも実行時に計測できる．したがって，直接摂動の除去は容易である．一方，間接摂動は，履歴生成文以外の文の実行で発生する．たとえば，直接摂動が通信タイミングを遅らせることで発生する，通

† 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
☆ 現在，株式会社アクセス
Presently with Accés Co., Ltd.

信待ち時間や受信順序の変化があげられる．このように，間接摂動は連鎖的に発生するため，その発生箇所および大きさを実行履歴から特定することが難しく，除去は容易でない．

大規模な実行履歴を生成する場合，実行履歴をメモリに蓄積する計測ツールでは，間接摂動の発生が顕著になる．この原因は，実行履歴のサイズが実メモリの容量を超えるとメモリスワップ（スワップ）が発生するためである．スワップが発生するタイミングはプロセスごとに異なるため，以降の通信タイミングが変化し，間接摂動が発生する．

そこで我々は，スワップが原因で発生する間接摂動を回避しつつ大規模な実行履歴を生成する手法を提案する．提案手法は，プログラム実行中，バリア同期の直後に実行履歴をメモリからディスクへ退避する．退避するタイミングを全プロセスで合わせ，かつそのためのオーバヘッドを均等にすることで，通信タイミングの変化を回避する．また，これを実現するため，全プロセスが退避に要する時間を共有する手法を考案し，実装した．提案手法を用いることで，ユーザは退避のためのオーバヘッドを除去するだけでプログラム本来の性能を復元できる．

提案手法が対象とするプログラムは，各プロセスが非同期な時刻で動作する環境で実行する，プログラムおよび実行履歴が必要とするメモリ量が実メモリの容量を超えるプログラムである．実行履歴には，通信および計算に関する性能情報を記録する．このようなプログラムの例として，プログラム自体が大量のメモリを使用するために実行履歴を蓄積するための空き容量が少ないもの，また計算に対する詳細な実行履歴を生成するために単位時間あたりの実行履歴生成量が多いものがあげられる．

以降では，まず 2 章で性能摂動を定義し，大規模な実行履歴を生成するときの問題をまとめる．次に 3 章で提案手法について述べ，4 章で適用実験の結果を示し，提案手法の有用性について議論する．その後，5 章で関連研究をあげる．最後に，6 章で本稿をまとめる．

2. 性能解析のための実行履歴

本章では，並列プログラムの実行をモデル化し，性能摂動を定義する．また，大規模な実行履歴の生成における問題を示す．

2.1 並列実行のモデル化

MPI プログラムを並列実行したとき，その実行 G を，頂点の集合 V および辺の集合 E からなる有向非循環グラフとして表す．

$$G = (V, E) \tag{1}$$

ここで，頂点および辺は，各々実行中に発生したイベントおよびイベント間の HB 関係（happened-before relation [7]）に対応する．イベントは，任意のプロセスがプログラムの一連の文を実行したときに 1 個発生する．MPI 関数などの通信関数の呼び出しから完了までに対応するイベントを通信イベント，それ以外を計算イベントと呼ぶ．また，HB 関係は，イベント a および b 間の依存関係を表し，b を実行する前に a の実行が完了したことを表す．以降では，p における i 番目のイベントを $v_{p,i}$ と表す．

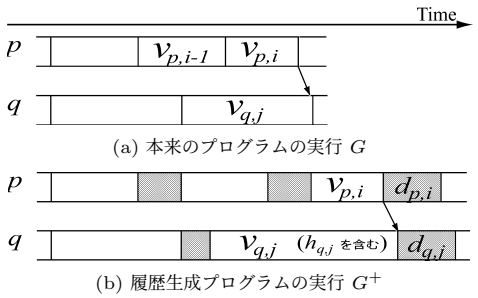


図 1 並列プログラム実行のモデル化
Fig. 1 Execution model for MPI programs.

図 1 (a) は G をガント図として図示したものである．縦軸はプロセスを表し，横軸は時間を表す．$v_{p,i}$ および $v_{q,j}$ 間の矢印で通信を表す．

プログラムの実行時間 $T(G)$ は，G のクリティカルパス[8] 上のイベントの集合 V_{CP} を用いて表現できる．

$$T(G) = \sum_{v_{p,i} \in V_{CP}} |v_{p,i}| \tag{2}$$

ここで，$|v_{p,i}|$ は $v_{p,i}$ の実行開始から終了までに要した処理時間である．

2.2 直接摂動と間接摂動

本来のプログラムに履歴生成文を挿入したものを履歴生成プログラムと呼ぶ．履歴生成プログラムの実行 G^+ では，本来のプログラムのイベントに加え，履歴生成文のイベントが発生する．

ここで，$v_{p,i}$ に対する履歴生成文のイベントを直接摂動 $d_{p,i}$ と呼ぶ（図 1 (b)）．直接摂動の大きさ $|d_{p,i}|$ は $d_{p,i}$ の処理時間で与える．$|d_{p,i}|$ の大半は実行履歴をメモリに蓄積するためのメモリコピーに要する時間である．

一方，間接摂動は独立したイベントとして発生せず，$v_{p,i}$ に含まれる．本稿では，$v_{p,i}$ に発生する間接摂動 $h_{p,i}$ の大きさを，$v_{p,i}$ の処理時間の変化で与える．G^+ の実行における $v_{p,i}$ の処理時間を $|v_{p,i}|^+$ と表すと，

$|h_{p,i}|$ は式 (3) で表現できる．

$$|h_{p,i}| = |v_{p,i}|^{+} - |v_{p,i}| \tag{3}$$

図 1 (b) に間接摂動の例を示す．$v_{p,i}$ および $v_{q,j}$ の実行前に発生した直接摂動の和が異なるため，通信タイミングがずれ $|v_{q,j}|$ が本来の実行より増大している．この増大した部分が $h_{q,j}$ に相当する．

ただし，G^{+} の実行時には $|h_{q,j}|$ を直接計測できない．そのため，$h_{q,j}$ を除去して $T(G)$ を復元する際，通信モデルなどを用いて $|h_{q,j}|$ を予測する必要がある．図 1 (b) の例であれば，$v_{p,i}$ の送信時刻を復元したのち，その時刻を基に $v_{q,j}$ の受信時刻を予測することで $|h_{q,j}|$ を計算できる．

以上から，G^{+} は次の式 (4) で表現できる．

$$G^{+} = (V \cup D, E) \tag{4}$$

ここで，D は直接摂動の集合を表す．なお，ここでは D の挿入によって HB 関係 E が変化しないと仮定している．この仮定は，送信元が非決定的な受信命令を含まない MPI プログラムに対して一般に成り立つ．

履歴生成プログラムの実行時間 $T(G^{+})$ は式 (5) で表現できる．

$$T(G^{+}) = \sum_{v_{p,i} \in V_{CP}} (|v_{p,i}|^{+} + |d_{p,i}|) \tag{5}$$

したがって，クリティカルパス上の直接摂動の合計および間接摂動の合計をそれぞれ DP および HP と表すと，定義より $T(G)$ は次の式 (6) のように計算すればよい．

$$T(G) = T(G^{+}) - (DP + HP) \tag{6}$$

しかし，すでに述べたように，HP を直接計測することは難しい．そこで，各 $|h_{p,i}|$ が $|v_{p,i}|$ に対して十分小さいと仮定できる場合には，$HP = 0$ と見なして次の式 (7) で $T(G)$ を近似的に求める．

$$T(G) \fallingdotseq T(G^{+}) - DP \tag{7}$$

2.3 大規模な実行履歴生成における問題

スワップが発生すると，以降の通信タイミングが変化するために間接摂動が発生する．通信タイミングが変化する理由は次の 2 つである．まず，プロセスごとにメモリ上に蓄積する実行履歴のサイズが異なるため，スワップが発生するタイミングも異なる．次に，スワップに要する時間もプロセス間で異なる．これは，記録するデータ量が等しい場合でも，ディスクへ記録する時間が一定でなく，実行するごとに変化するためである．たとえば，IDE で接続された最高転送速度 33 MB/s のハードディスクに，512 MB のデータを記録するための時間は最小 25 秒から最大で 36 秒まで変化した．スワップによって発生する間接摂動の例を**図 2** (b) に示す．p および q 間では，通信 $v_{p,i} \rightarrow v_{q,j}$ の前後でスワップの発生タイミングが異なるため，本来の実行状況図 2 (a) と比較して $h_{q,j}$ が発生している．また，p および r 間では，スワップに要する時間が異なるため，$h_{r,k}$ が発生している．

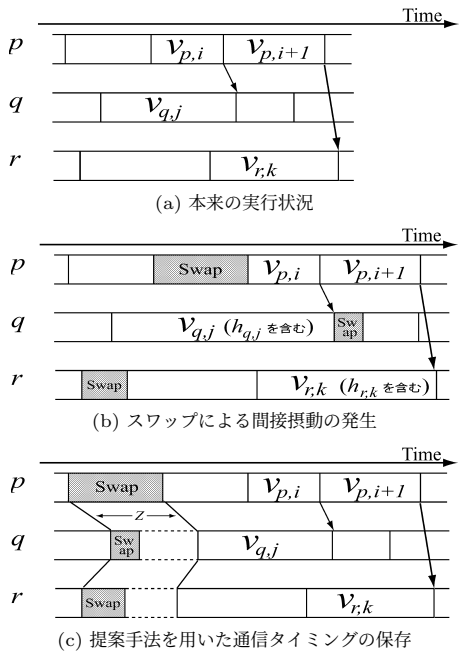


図 2 スワップによる間接摂動の発生とその回避
Fig. 2 Preventing indirect perturbations caused by swap.

すでに述べたように，間接摂動を除去する際には，その大きさを通信モデルなどを用いて予測する必要がある．しかし，あるイベントの間接摂動の大きさは，最悪の場合クリティカルパス上のすべてのイベントに依存するため，予測に要する計算量は多い．したがって，大規模な実行履歴に対する実行の復元は非常に煩雑になる．また，スワップに要する時間はイベントの処理時間より数十倍長く，発生する間接摂動も大きいために，式 (7) のように近似できない．

3. 提 案 手 法

本章では，まず提案手法の目的について述べる．その後，提案手法の実装としてプログラムの中断および中断時間の決定について述べる．

3.1 提案手法の目的

スワップを未然に防ぐためにメモリ上に蓄積された実行履歴をファイルへ記録することを，実行履歴の退避と呼ぶ．提案手法は，明示的に実行履歴を退避し，その際に発生する間接摂動を小さく抑えることで実行の復元を容易にすることを目指す．

この目的のために，提案手法は，履歴生成プログラ

ムに実行履歴の退避のための文（履歴退避文）を挿入して実行する．このプログラムの実行 G' は，履歴退避文の実行イベントの集合を F とすると，次の式 (8) で表現できる．ただし，式 (4) と同様に，F の挿入によって HB 関係は変化しないと仮定している．

$$G' = (V \cup D \cup F, E) \tag{8}$$

履歴退避文を実行することで，G^+ と比較して，あらたな直接摂動および間接摂動が G' に発生する．クリィカルパス上のこれらの合計をそれぞれ DP' および HP' とすると，式 (6) と同様に考えて，次の式 (9) のように $T(G')$ から $T(G^+)$ を復元できる．

$$T(G^+) = T(G') - (DP' + HP') \tag{9}$$

3.2 提案手法の条件

G' から G^+ への復元を容易にするために，提案手法は次の条件を満たす必要がある．

- 履歴退避文によって発生する間接摂動を抑える

この条件を満たすことで $HP' = 0$ と見なせるため，式 (7) と同様に考えて，G' から DP' を除去するだけで G^+ を容易に復元できる．

この条件を満たすために，実行履歴を退避する際のプロセス間の通信タイミングを保存し，間接摂動を小さく抑える必要がある．そこで提案手法では，全プロセスがいっせいにプログラムの実行を中断し，実行履歴を退避，その後実行を再開する．この際，図 2 (c) のように全プロセスにおいて中断から再開までの中断時間 Z を均等にすることで，プロセス間の相対的な通信タイミングを保存できる．実行履歴を退避するタイミングを合わせ，かつ Z を均等にすることで，間接摂動の発生を回避できる．

全プロセスがいっせいに中断できる箇所としては，全プロセスが同期するイベントがあげられる．提案手法は，対象プログラムが集合通信を含むことを前提とし，集合通信を実行した直後にプログラムを中断する．

なお，一般に実行履歴に基づく性能解析は繰り返し行うため，実行履歴の生成に要する時間 $T(G')$ は短いことが望ましい．そのため，提案手法のオーバヘッドである中断時間 Z を可能な限り短く決定する必要がある．

そこで，提案手法では，プログラムの実行中，実行履歴の退避に要する時間が最も大きいプロセスを基準に中断時間 Z を動的に決定し，全プロセスに通知する．この際，メモリ上の実行履歴のサイズが一定の閾値 S より小さい場合には実行履歴をディスクに記録する必要はないと判断し，中断後すぐに再開する．

一方，プログラム実行前に中断時間を予測して静的に決定する手法も考えられるが，次の 2 つの問題が生じる．まず，プログラマが実行環境ごとに適切な値を予測して設定する必要がある．次に，実行履歴の退避に要する時間が予測を超える可能性がある．この際，プロセス間で中断時間が不均等になるため，間接摂動が発生する．静的な手法と提案手法の詳細な比較は 4.3 節で後述する．

```
// b：メモリ上の実行履歴のサイズ
// S：退避するか否かを判断する閾値
 1: function suspension (b) {
 2:    t := PMPI_Wtime();   // 中断開始
 3:    f := (b > S);
 4:    PMPI_Allreduce(f,σ,OR);
 5:    if (σ = true)
         trace_output();   // 実行履歴をディスクへ記録
 6:    Z := suspension_time(t);   // 中断時間の決定
 7:    while ( (PMPI_Wtime() - t) < Z ) {
 8:       // busy wait
 9:    }
10: }   // 中断終了
```

図 3 実行履歴の退避の概要
Fig. 3 Pseudo algorithm of proposed method.

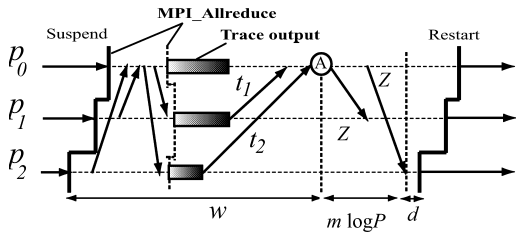


図 4 実行履歴の退避の処理の流れ
Fig. 4 Procedure of proposed method.

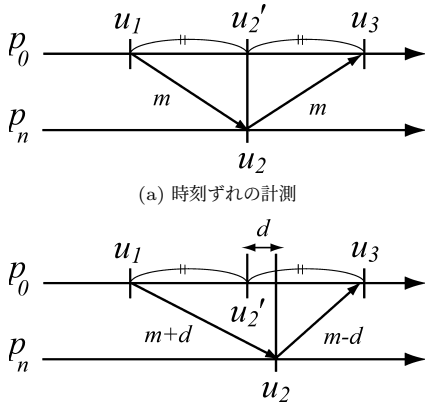
3.3 提案手法の実装

履歴退避文の概要を**図 3** に示す．この手法では，まず中断開始時刻を計測し（2 行目），次にメモリ上の実行履歴が閾値を超えているか判定する（3 行目）．判定結果を全対全通信を用いて全プロセスで共有し（4 行目），いずれかのプロセスが閾値を超えていた場合には実行履歴をディスクへ記録する（5 行目）．その後，開始時刻を基に中断時間を決定し（6 行目），中断時間が経過するまで空の while ループを実行して待機する（7 行目）．なお，MPI_Wtime を用いた時間計測の誤差は 2 マイクロ秒程度であり，図 3 の処理全体に対して十分小さく，無視できる．

実行履歴をディスクへ記録する場合（$b > S$）の提案手法の処理の流れを**図 4** に示す．縦軸および横軸は各々プロセスおよび時間を表す．各プロセスの横向きの矢印および点線は各々プログラム本来の処理およびプログラムの中断を表す．提案手法では，各プロセスが実行履歴をディスクへ記録したのち，図 4 の A の

```
// P：プロセス数
// C_0,..,C_P：あらかじめ計測した時刻のずれ
// m：通信命令の平均処理時間
// d：時刻ずれの計測誤差の最大値
 1: function suspension_time (t) {
 2:   PMPI_Gather(t,p_0);
 3:   if (p_0) then
 4:     for i := 0 to P − 1 do t_i := t_i + C_i;
 5:     w = PMPI_Wtime() − minimum(t_0,t_1,...,t_{P−1});
 6:     Z := w + m log_2 P + d;
 7:   end
 8:   broadcast(Z,p_0);
 9:   return Z;
10: }
```

図 5 中断時間 Z を決定するアルゴリズム
Fig. 5 Algorithm for computing suspention time.



(a) 時刻ずれの計測

(b) 計測誤差の最大値 d

図 6 時刻ずれの計測と計測誤差
Fig. 6 Measurement of clock gaps.

時点で関数 suspension_time（**図 5**）を呼び出して中断時間を決定する．なお，ディスクへの記録をともなわない場合の処理の流れは，図 4 から trace_output を省略したものとなる．

なお，実行履歴をディスクへ記録する際には，MPI の I/O 関数（MPI_File_write）を用いる．各プロセスに接続されたローカルなディスクに，メモリ上に蓄積された実行履歴をプロセスごとに独立に記録する．この際，記録するファイルの空き容量が不足すると，新たなファイルを作成して記録を続行する．

3.4 中断時間の決定

中断時間 Z を決定するアルゴリズムを，図 5 を用いて詳細に説明する．

まず，各プロセスの中断開始時刻 $t_1, ..., t_{P-1}$ を p_0 に集め，最も早い中断開始時刻から現在までの時刻を w とする（2 から 5 行目）．この際，中断開始時刻を比較するために，全プロセスで共有されたグローバルな時刻が必要である．提案手法では p_0 の時刻を仮想的にグローバルな時刻と見なす．p_0 と他のプロセスとの時刻のずれをあらかじめ計測し，必要に応じて他のプロセスの時刻を p_0 の時刻に変換する（4 行目）．

時刻のずれの計測には，一般によく用いられる手法[9] を利用する．**図 6** (a) にその手順を示す．この手法では，p_0 から p_n への送信に要する時間と p_n から p_0 への送信に要する時間が等しいと仮定して時刻のずれを計測する．この仮定のもとでは，時刻 u_1 および u_3 の中間である u_2' と u_2 はグローバルな時間軸上で一致する．したがって，p_0 および p_n のローカルな時刻のずれ C_n は，$C_n = u_2' - u_2$ で計算できる．ただし，実際には固定長のデータに対する通信時間は一定でないため，この計測結果に誤差が生じる．平均通信時間を m とし，通信時間が変動する幅を d とすると，時刻のずれの計測誤差は最大 d となる（図 6 (b)）．なお，時刻のずれの計測とともに，平均通信時間 m と計測誤差の最大値 d を求める．時刻のずれを計測するために行った通信（プロセス数 $\times 2$ 回）の平均通信時間を m とし，最大もしくは最小の通信時間と m との差を d とする．

次に，p_0 は w を基に中断時間 Z を決定する（6 行目）．この際，決定した Z を他のプロセスへ通知する時間を予測し，Z に含める必要がある．仮に，通知に要する時間を Z に含めない場合，最も早く中断を開始したプロセス q が Z を受信したときすでに中断終了時刻を経過している可能性がある．この場合，q の中断時間は通知に要した時間だけ他のプロセスと比較して長くなるため，中断時間の均一化に失敗する．他のプロセスへの通知には 1 対 1 通信を用いて二分木型のブロードキャストを行う（8 行目）．したがって，平均通信時間 m を用いて，ブロードキャストに要する時間の上限は $m \log_2 P$ と予測できる．さらに，時刻ずれの計測誤差の最大値 d を加算して，Z の計算式は次のようになる．

$$Z = w + m \log_2 P + d$$

注意すべき点として，プロセス間の時刻のずれがプログラムの実行中に増大することがあげられる．時刻のずれが増大すると，中断時間が変動する．図 4 を例に説明する．時刻ずれの計測時と比較して，p_2 の時刻が 1 秒進んでいると仮定する．この場合，p_2 は本来の時刻 t_2 より 1 秒早い時刻 t_2' を p_0 に送信する．p_0 は t_2' を基に w を決定するため，w の値は本来より 1 秒増大する．この結果，中断時間が増大し，提案手法のオーバヘッドが増大する．

このオーバヘッドの増大を解決するためには，実行

中に時刻のずれを再計測する必要がある．提案手法では，プログラムを中断するごとに時刻のずれを計測する．時刻のずれを再計測する場合としない場合の詳細な比較は 4.3 節で示す．

4. 評　　価

以下に示す 5 つの観点から，提案手法を評価する．

- 提案手法のオーバヘッド
- 実行時間の復元精度
- 中断時間を決定する手法の比較
- 提案手法の実用性
- 提案手法の適用可能性

実験では，MPI 実装として MPICH-SCore[10] を用いた．また，MPI プログラムの実行には，32 台の PC を通信バンド幅 2 Gb/s の Myrinet ネットワーク[11] で相互接続したクラスタを用いた．各 PC は，Pentium III 1 GHz のプロセッサを持ち，2 GB の実メモリを装備している．なお，各 PC におけるスワップファイルのサイズは 2 GB である．

4.1 提案手法のオーバヘッド

この実験に用いたプログラムは，3 次元画像間の位置を対応づける位置合わせプログラムである．このプログラムは 2 つの画像を入力とする．一方の画像を特定の方向へ移動もしくは変形させ，他方と対応づけた類似度を評価し，類似度が高くなる方向へと変形を繰り返す．実験では，画像中の 1 画素ごとの対応づけを計算する関数に履歴生成文を挿入し，計算の途中結果を実行履歴に記録した．生成した実行履歴は，それを基に変形の繰返しを再現できるため，効率の良い変形方向の検討に有用である．なお，今回生成した実行履歴のサイズは 256 MB であった．さらに，実行履歴のサイズが増大した場合の実行時間への影響を調べるために，実行履歴にダミーデータを加えてそのサイズを増大させた．この際，実行履歴の生成頻度は一定とし，一度に蓄積する情報の量を変化させることで実行履歴のサイズを変化させた．なお，提案手法を用いる場合には，プログラム内の MPI_Barrier の直後に履歴退避文を挿入した．プログラム実行時の MPI_Barrier の実行回数は 520 回であった．

図 7 に，実行履歴のサイズを変化させたときのプログラムの実行時間を示す．ここで，プロセス数は 32 個であり，退避の閾値を $S = 512$ MB とした．なお，このプログラムが本来の処理に使用するメモリ量は 680 MB であり，その実行時間は 58 秒である．

提案手法を用いない場合，実行履歴のサイズが $M \leq$ 1,100 のとき，実行時間はほぼ一定である．しかし，この値を超えると急激に実行時間が増大し，$M = 1{,}800$ においては，本来の 7 倍以上の 459 秒を要した．この原因はスワップの発生にある．スワップが発生する状況下では，実行履歴を実メモリに蓄積するための時間が通常と比較して 50 倍以上に増大していた．

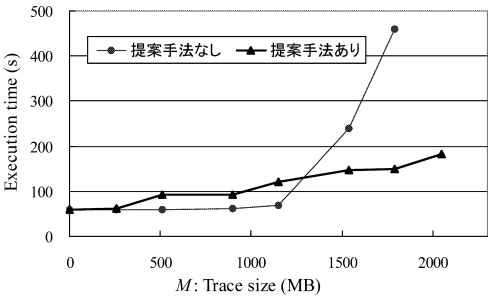


図 7 実行履歴のサイズとその生成に要する実行時間
Fig. 7 Trace size and time spent for generation.

一方，提案手法では，実行履歴のサイズが増大した際の急激な実行時間の増大を回避できる．スワップの発生を回避できるため，$M = 1{,}800$ の場合でも，本来の 3 倍以下の 149 秒で実行できた．ただし，$M \leq 1{,}100$ のときは，提案手法を用いない場合よりも実行時間が長い．これは，実行履歴のサイズが実メモリの容量を超えない場合も，実行履歴の退避のためのオーバヘッド DP' が発生するためである．以上から，提案手法が有用な状況は，生成する実行履歴のサイズおよびプログラムが使用するメモリ量の合計が実メモリの容量より大きい場合である．

なお，提案手法を用いない場合，実行履歴のサイズは実メモリの容量とスワップファイルのサイズの合計でおさえられる．今回の実験では最大で 1,800 MB であり，それ以上の実行履歴は生成できなかった．一方，提案手法では，実行履歴のサイズはディスクの容量でおさえられる．したがって，ディスクの空き容量がある限り実行履歴を生成できる．

4.2 実行時間の復元精度

まず，提案手法におけるプログラムの実行時間の復元精度を調べるため，退避のためのオーバヘッド DP' を除去して実行時間を復元した（**表 1** の $R(T_1)$）．評価に用いたプログラムは，4.1 節と同じものであり，プロセス数を 32 個とし，$S = 512$ MB とした．$M \leq 896$ のとき，復元した実行時間 $R(T_1)$ は本来の実行時間 T と比較して最大 1.1 秒大きく，実行時間の復元誤差は約 1.8%である（復元誤差 $= |R(T_1) - T|/T$）．

次に，イベント単位での間接摂動の発生を調べるため，プログラムの最後に実行する MPI_Barrier 命令の処理時間をプロセスごとに計測した（**図 8**）．図 8

表 1 実行履歴のサイズを変化させたときの生成に要する時間および復元した実行時間
Table 1 Measured and estimated execution time with different trace sizes.

実行履歴サイズ (MB) M	使用メモリ量 (MB)	総実行時間（秒） 退避なし	提案手法（動的） $S=512$		$S=256$		$S=128$		静的 $S=512$		再計測なし $*$ $S=512$	
		実測 T	実測 T_1	復元 $R(T_1)$	実測 T_2	復元 $R(T_2)$	実測 T_3	復元 $R(T_3)$	実測 T_s	復元 $R(T_s)$	実測 T_a	復元 $R(T_a)$
0	680	58.4	—	—	—	—	—	—	—	—	—	—
256	1,067	59.3	60.7	59.7	77.6	60.0	72.0	59.7	69.3	58.8	64.3	59.6
512	1,312	60.0	91.9	60.8	90.8	61.1	91.9	63.2	110.5	60.0	95.4	60.9
896	1,705	61.1	93.3	62.2	105.1	62.2	104.4	64.1	111.9	61.5	98.0	61.8
1,152	2,089	69.6	120.4	63.3	118.9	63.6	119.5	65.2	152.9	62.5	126.2	63.2
1,536	2,345	239.8	147.8	66.2	143.8	64.6	135.1	66.2	196.8	66.4	153.1	65.6
1,792	2,730	459.5	149.6	67.0	147.5	66.0	146.1	67.0	170.5	70.2	155.7	67.5

$*$ 中断時間を動的に決定．ただし，プログラム実行前に時刻のずれを計測し，実行中は再計測をしない．

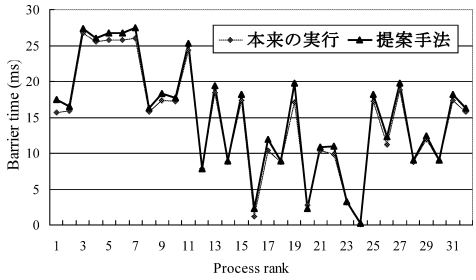


図 8 各プロセスにおける MPI_Barrier の処理時間
Fig. 8 Execution time for MPI_Barrier on each process.

は，横軸がプロセスを表し，縦軸にプロセスごとの MPI_Barrier の処理時間を表す（$M = 896$）．この MPI_Barrier では実行履歴をディスクへ記録せず，中断後すぐに再開した．図 8 における本来の実行は，プログラムの中断を行わない場合の MPI_Barrier の処理時間を示す．本来の実行および提案手法を用いた実行におけるプロセスごとの処理時間はほぼ等しく，22 台のプロセスで誤差 1 ミリ秒以下で一致している．一致しないプロセスにおいても，その誤差は 30 ミリ秒の通信に対して最長 2 ミリ秒と小さい．このことは，提案手法において，プログラムの中断を挿入しても間接摂動がほとんど発生しないことを示している．この理由は，図 8 が示すように，プロセス間の相対的な MPI_Barrier の開始および終了タイミングが本来の実行とほぼ同じためである．

なお，実行履歴をディスクへ記録した直後の MPI_Barrier について同様に調べたところ，処理時間の誤差は最長 20 ミリ秒であり，無視できない大きさの間接摂動が発生していた．この原因は，記録直後の中断時間の均等化に失敗するためである．提案手法では，中断時間の通知に要する時間を平均通信時間から予測して中断時間を決定している．しかし，実際に計測したところ，実行履歴をディスクへ記録した直後の中断時間の通知において，通信時間が平均通信時間より長くなる場合があった．この際，p_0 以外のプロセスは p_0 が決定した中断時間の経過後に中断時間を受信し，中断時間の均等化に失敗する．その結果プロセス間の通信タイミングがずれ，間接摂動が発生した．

このため，実行履歴をディスクへ記録する頻度が高いほど間接摂動が増大し，実行時間の復元精度が悪化する．表 1 の $R(T_2)$ および $R(T_3)$ が示すように，S が小さいほど，復元後の実行時間は長い．記録直後の通信時間が増大する原因については判明しておらず，正確な通信時間の予測は今後の課題である．ただし，$S = 512$ かつ $M = 896$ のとき，間接摂動が増大した MPI_Barrier は 4 回（全体の 0.7%）であり，均等化に失敗する頻度は非常に少なかった．

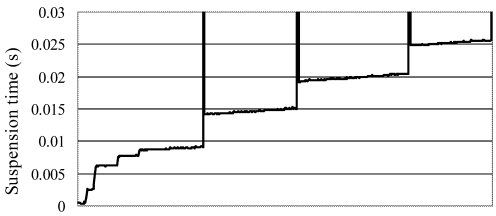
4.3 中断時間を決定する手法の比較

4.1 節で用いたプログラムを対象に，中断時間を決定する手法について比較する．

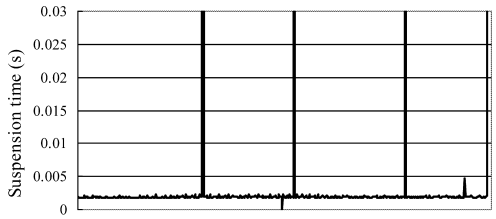
まず，中断時間を静的に決定する場合と提案手法を比較し，その復元精度およびオーバヘッドについて調べた．表 1 に結果を示す．静的決定では，長い中断時間と短い中断時間をあらかじめ決定し，通常は短い中断時間を利用し，実行履歴のサイズが閾値を超えた場合には長い中断時間を利用した．512 MB のデータをファイルへ記録する時間が 35 秒前後であることから，長い中断時間を 40 秒とした．また，短い中断時間は 20 ミリ秒とした．

$M \leq 896$ における復元精度を比較する．$R(T_1)$ の復元誤差が平均 1.2% であることに対して，$R(T_s)$ の復元誤差は平均 0.7%であり，復元精度において静的決定が優れる．この理由は，提案手法では，実行履歴をディスクへ記録した直後の中断時間の均等化に失敗して間接摂動が増大する場合があるためである．

一方，実行時間を比較すると，すべての M におい



(a) 再計測なし



(b) 再計測あり（提案手法）

図 9 時刻ずれの再計測による中断時間の減少
Fig. 9 Decrease of suspention time by run-time clock adjustment.

て $T_s > T_1$ であり，静的決定のオーバヘッドが大きい．この理由は，静的決定では中断時間が固定であることに対して，提案手法ではファイルへの記録が早く終わる場合に中断時間が短くなるためである．

なお，今回の実験では稀にファイルへの記録に通常の倍の時間を要する場合（512 MB に対して 60 秒程度）があった．この際，動的決定ならば記録時間の増大に合わせて中断時間を変化させて対応できたが，静的決定では対応できず，間接摂動が増大した．以上から，提案手法は記録時間が増大しても通信タイミングを保存できる点およびオーバヘッドを小さく抑えられる点で優れている．

次に，各プロセス間の時刻のずれが復元精度およびオーバヘッドに及ぼす影響を調べるために，時刻の再計測をしない場合と提案手法を比較した．表 1 より，再計測なしの場合（$R(T_a)$）の復元誤差は平均 1.1%であり，復元精度は提案手法とほぼ同じである．一方，再計測しない場合の実行時間 T_a は実行履歴のサイズにかかわらず T_1 より 5 秒程度長く，オーバヘッドは増大した．

この理由は，3.4 節で示したように，プロセス間の時刻のずれの増大にある．**図 9** は横軸に MPI_Barrier を実行順に並べ，縦軸に MPI_Barrier ごとの中断時間を表したものである．図 9 (a) が示すように，再計測をしない場合，プログラムの進捗に従って中断時間が増大する．提案手法では中断処理ごとに時刻のずれを再計測することで，中断時間の増大を回避できた（図 9 (b)）．したがって，提案手法では復元精度を保ったまま全体のオーバヘッドを小さくできる．

4.4 提案手法の実用性

提案手法の実用性な用途として，以下の 3 通りのケースが想定できる．まず，アプリケーション自体が大量のメモリを使用するために，実行履歴を蓄積するための空きメモリ容量が少ない場合があげられる．次に，キャッシュヒット率の遷移など，計算に対する詳細な実行履歴を生成する必要があり，単位時間あたりの実行履歴生成量が多い場合があげられる．最後に，アプリケーションの実行時間が長いために実行履歴を蓄積するための空きメモリ容量が不足する場合があげられる．このうち，最初のケースに相当するプログラムを用いて提案手法の実用性を評価する．

この実験に用いたプログラムは，姫野ベンチマーク 98[12] である．このプログラムはポアッソン方程式解法をヤコビの反復法で解く場合に主要なループの処理速度を計るものである．このプログラムを問題サイズ Large（$512 \times 256 \times 256$）で実行する場合のキャッシュの効果を調べるために，CPU のパフォーマンスカウンタ[3] を用いてキャッシュミスの回数を計測し実行履歴に記録する．今回の実験で生成した実行履歴のサイズは 277 MB であった．プログラムが本来の処理に使用するメモリ量は 1,815 MB であり，実験環境の実メモリ容量は 2 GB であることから，プログラムの実行中にスワップが発生した．提案手法を用いる場合，退避の閾値は $S = 32$ MB とすることでスワップを回避する．なお，実行履歴を生成しない場合の本来の実行時間は約 218 秒であった．

このプログラムを 8 個のプロセスで実行したときの，最終ループにおけるプロセスごとの通信命令の実行時間を**図 10** に示す．この図はプログラム実行の一部を抜粋したもので，縦軸はプロセスを表し，横軸は経過時間を表す．図の Send および Wait はそれぞれ MPI_Send および MPI_Wait を表し，各通信命令の開始時刻と終了時刻をグラフに示す．なお，各プロセスの時刻はプロセス 0 の時刻に変換したものを表示している．図 10 (a)〜(c) において経過時間が異なるのは，履歴生成文および履歴退避文の直接摂動によって，通信命令の開始時刻が遅れるためである．

図 10 (a) および (b) を比較すると，プロセス 6 を除くすべてのプロセスに，スワップ発生時に大きな間接摂動が発生している．開始時刻の変化はプロセス 0 の Send2 が最も大きく，約 180 ミリ秒早まっている．また，実行時間の変化はプロセス 7 の Send1 が最も大きく，約 230 ミリ秒増加している．これらは本来の実

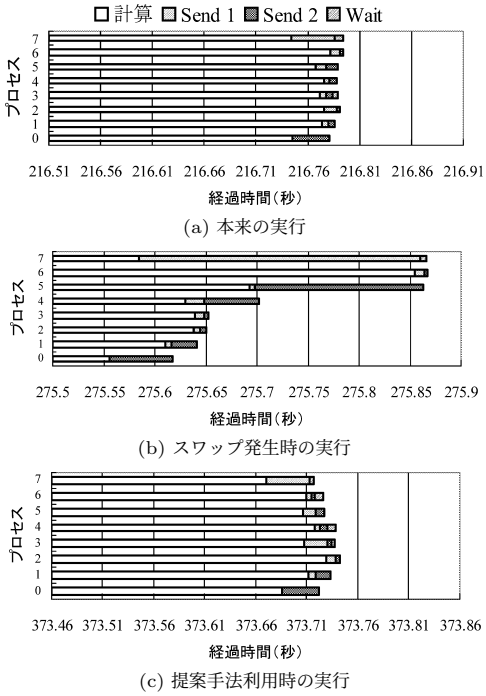


図 10 プロセスごとの通信命令の実行時間
Fig. 10 Execution times of communication routines on each process.

行における通信命令の実行時間（最大でも 40 ミリ秒）と比較して大きく，正確な性能解析を困難にする．

一方，図 10 (a) および (c) を比較すると，提案手法を用いることで間接摂動を小さく抑えていることが分かる．開始時刻の変化は最大でもプロセス 7 の Send1 における 25 ミリ秒であり，実行時間の増加は最大でもプロセス 3 の Send1 における 17 ミリ秒であった．このため，提案手法を用いると，プログラムの性能に関わる特徴を保存できている．たとえば，最も実行時間の長い通信（プロセス 0 の Send2 とプロセス 7 の Send1）が変化していないため，性能ボトルネックの正確な特定が期待できる．このように，提案手法は本来の実行に近い性能を記録できる点で実用性が高い．

4.5 提案手法の適用可能性

提案手法は，全プロセスが同期する集合通信に着目し，集合通信の直後に実行履歴を退避することで間接摂動の増大を回避している．そのため，集合通信を含まないプログラムに対しては適用できない．集合通信を含まないプログラムを対象とした実行履歴の退避は今後の課題である．たとえば，1 対 1 通信で構成される通信パターンから論理的な同期箇所を特定して実行履歴を退避する手法が考えられる．

また，中断時間の決定において，提案手法は平均通信時間を基に通信時間を予測している．そのため，平均通信時間にばらつきがあるネットワーク上で通信する場合や実行履歴をディスクへ記録した直後など通信時間が極端に増大する場合には，予測に失敗し中断時間が不均等になる可能性がある．平均通信時間を大きく見積もることで中断時間を均等にできるが，オーバヘッドが増大し実行履歴の生成に要する時間も増大する．以上から，提案手法は通信時間が正確に予測できる環境において有用性が高い．

5. 関 連 研 究

本章では，関連研究として，大規模なプログラムに対する性能解析を実現する手法および性能摂動を除去する手法を紹介する．

まず，大規模なプログラムに対する性能解析を実現する手法は，以下の 2 種類に分類できる．

- 実行履歴のサイズを削減する手法：この手法は，プログラムの実行中に実行履歴のサイズを削減することで，大規模なプログラムの実行履歴を生成する．この手法は，可逆圧縮およびフィルタリング，情報集約，動的計測などに大別できる[13]．生成する実行履歴のサイズを削減できるため，多くの情報をメモリ上に蓄積できる．ただし，メモリ上に実行履歴を蓄積することに違いはなく，安全に生成できる実行履歴のサイズの上限は実メモリの容量でおさえられる．
- 実行履歴のオンライン解析[14]：この手法は，異なる計算機に実行履歴を転送することで，大規模なプログラムの性能解析を実現する．プログラムの実行中に，通常の通信で利用するネットワークとは別のネットワークを用いて解析用の計算機に実行履歴を送信する．プログラムを実行する計算機のメモリ上に実行履歴を残さないため，スワップは発生しない．プログラムの実行中に性能解析ができるため，数日にわたって実行するプログラムに有用であるが，2 種類のネットワークおよび解析用の計算機などの他のハードウェアが必要となる．

提案手法をこれらの手法と比較すると，提案手法の特長は，ハードウェアを追加することなく，実メモリ容量の制限を超えて実行履歴を生成できる点にある．

次に，実行履歴生成時に発生する性能摂動を回避および除去する手法を示す．

- Malony ら[15] のタイミング解析：プログラムの実行後，実行履歴から本来のイベントの実行タイミングを復元する．あるイベントの実行までに発

生した性能摂動をモデルを基に概算し，実行時刻から性能摂動を減算することで本来のタイミングを求める．ただし，通信イベントに発生する間接摂動が無視できるほど小さい状況を想定しているため，間接摂動が大きくなると正確な実行タイミングを復元できない．

- Wu ら[16),17)] の性能摂動のオンライン除去：分散システム上で，プログラム実行中に性能摂動を除去する．この際，プログラムに対してクリティカルポイントを定め，その実行までに性能摂動を除去することを保証し，クリティカルポイント以降への性能摂動の波及を回避する．また，メッセージの送信時に送信時刻を付与し，それを基に受信側が受信順序を再現することで，受信順序を保存する．主に通信処理に発生する性能摂動の除去および回避を重視しており，通信タイミングの解析に有用である．

我々の知る限り，スワップが原因で発生する間接摂動を考慮した手法は存在しない．本研究の特長は，実行履歴の退避のためのオーバヘッドを均等化することで，間接摂動の発生を回避し実行の復元を容易にする点にある．

6. ま と め

本稿では，スワップが原因で発生する間接摂動を回避しつつ実メモリの容量を超える実行履歴を生成する手法を提案した．提案手法は，実行履歴を退避するタイミングを全プロセスで合わせ，かつそのためのオーバヘッドを均等にすることで，通信タイミングの変化を回避する．評価実験において，実行履歴の生成に要する時間を半分以下に削減でき，得られた実行履歴から容易に実行を復元できた．ゆえに，提案手法は性能解析のための大規模な実行履歴の生成に有用であると考える．

今後の課題としては，ファイルへの記録直後の通信時間が増大することを考慮した中断時間の決定手法や，集合通信を含まないプログラムを対象とした実行履歴の退避があげられる．

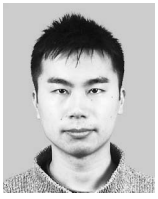
謝辞 本研究の一部は，文部科学省科学研究費補助金若手研究（B）(15700030）および日本電気（株）の補助による．また，有益なご意見をいただいた査読者の方々に深く感謝いたします．

参 考 文 献

1) Message Passing Interface Forum: MPI: A Message-Passing Interface Standard, *Int. 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 ISCA Int. Conf. Parallel and Distributed Computing Systems* (*PDCS'01*) (2001).
4) Nagel, W.E., Arnold, A., Weber, M., Hoppe, H.-C. and Solchenbach, K.: VAMPIR: Visualization and Analysis of MPI Resources, *J. Supercomputing*, Vol.12, No.1, pp.69–80 (1996).
5) 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).
6) Malony, A.D., Reed, D.A. and Wijshoff, H.A.G.: Performance Measurement Intrusion and Perturbation Analysis, *IEEE Trans. Parallel and Distributed Systems*, Vol.3, No.4, pp.433–450 (1992).
7) Lamport, L.: Time, Clocks, and the Ordering of Events in a Distributed System, *Comm. ACM*, Vol.21, No.7, pp.558–565 (1978).
8) Hollingsworth, J.K.: Critical Path Profiling of Message Passing and Shared-Memory Programs, *IEEE Trans. Parallel and Distributed Systems*, Vol.9, No.10, pp.1029–1040 (1998).
9) Haridasan, M. and Pfitscher, G.H.: Use of the Parallel Port to Measure MPI Intertask Communication Costs in COTS PC Clusters, *Proc. 2nd Workshop Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems* (*PMEO'03*) (2003). 6 pages (CD-ROM).
10) 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. Conf. Supercomputing* (*ICS'98*), pp.243–250 (1998).
11) 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).
12) Himeno Benchmark (2005). http://accc.riken.jp/HPC/HimenoBMT/

13) Uhlig, R.A. and Mudge, T.N.: Trace-Driven Memory Simulation: A Survey, *ACM Computing Surveys*, Vol.29, No.2, pp.128–170 (1997).
14) Brunst, H., Malony, A.D., Shende, S.S. and Bell, R.: Online Remote Trace Analysis of Parallel Applications on High-Performance Clusters, *Proc. 5th Int. Symp. High Performance Computing* (*ISHPC'03*), pp.440–449 (2003).
15) 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).
16) Wu, W., Gupta, R. and Spezialetti, M.: Experimental Evaluation of On-line Techniques for Removing Monitoring Intrusion, *Proc. 2nd Symp. Parallel and Distributed Tools* (*SPDT'98*), pp.30–39 (1998).
17) Wu, W., Spezialetti, M. and Gupta, R.: A Protocol for Removing Communication Intrusion in Monitored Distributed Systems, *Proc. 18th Int. Conf. Distributed Computing Systems* (*ICDCS'98*), pp.120–129 (1998).

(平成 17 年 10 月 4 日受付)
(平成 18 年 2 月 1 日採録)



置田 真生

平成 13 年大阪大学基礎工学部情報科学科卒業．平成 15 年同大学院基礎工学研究科修士課程修了．平成 18 年同大学院情報科学研究科博士課程修了．博士（情報科学）．平成 15 年電気通信普及財団第 18 回テレコムシステム技術学生賞受賞．現在，株式会社アクセスに勤務．並列ソフトウェア開発環境に興味を持っている．

伊野 文彦（正会員）

平成 10 年大阪大学基礎工学部情報工学科卒業．平成 12 年同大学院基礎工学研究科修士課程修了．平成 14 年同大学院同研究科博士課程中退．同年同大学助手．博士（情報科学）．平成 15 年国際会議 HiPC'03 最優秀論文賞，平成 16 年先進的計算基盤システムシンポジウム SACSIS'04 最優秀論文賞受賞．並列計算機の応用およびソフトウェア開発環境に関する研究に従事．

萩原 兼一（正会員）

昭和 49 年大阪大学基礎工学部情報工学科卒業．昭和 54 年同大学院基礎工学研究科博士課程修了．工学博士．同大学助手，講師，助教授を経て，平成 5 年奈良先端科学技術大学院大学教授．平成 6 年より大阪大学教授．平成 4 年〜5 年文部省在外研究員（米国メリーランド大学）．平成 15 年国際会議 HiPC'03 最優秀論文賞，平成 16 年先進的計算基盤システムシンポジウム SACSIS'04 最優秀論文賞受賞．現在，並列処理の基礎および応用に興味を持っている．