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

2004－EVA－8　(1)
2004／3／18

MPIプログラムの改善可能性を評価するための効率的な実行履歴の生成

神戸友樹[†]　置田真生[†]
伊野文彦[†]　萩原兼一[†]

本稿では，MPI プログラムの性能改善可能性を評価することを目的として，そのための効率的な実行履歴の生成手法を提案する．この評価は性能予測に基づいていて，典型的な改善手法をプログラムに適用したときの実行時間を予測する．提案手法は，実行時間予測に影響しない通信に対する記録を集約することで，実行履歴のファイルサイズを削減する．このような通信はプログラム実行時に送信命令および受信命令の呼び出し時刻を比較することで自動的に特定する．提案手法により，実行履歴のファイルサイズだけでなく予測に要する時間をおよそ半減できた例を示す．

Efficient Instrumentation for Performance Improvement Assessment of MPI Programs

Yuki Kanbe,[†] Masao Okita,[†] Fumihiko Ino[†]
and Kenichi Hagihara[†]

This paper proposes an efficient instrumentation technique for assessing the performance improvement of MPI programs. This assessment is based on what-if predictions, which give a predicted execution time if the target program is modified according to typical tuning techniques. In order to reduce the size of trace file, our instrumentation technique aggregates records of communications that have no influence on what-if predictions. Such unnecessary records are automatically identified during program execution, according to the call times of send and receive routines. We also show a case study, where the time for what-if predictions as well as the size of trace files is reduced roughly into a half.

1. はじめに

メッセージ通信パラダイム[1]は，クラスタおよびグリッドなどの分散メモリ型並列計算環境に適したプログラミング手法である．たとえば，標準メッセージ通信仕様 MPI（Message Passing Interface）[1]を用いることで，これらの計算環境において性能のよい並列プログラムを開発できる．しかし，その開発にはプログラムの性能を繰り返し改善することが重要である．

そこで，並列プログラムの性能改善を支援するために，実行履歴に基づく性能解析に関する研究[2)~5)]が行われている．実行履歴とは，プログラム実行中に発生したイベントをその発生時刻などとともに記録したファイルを指す．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．

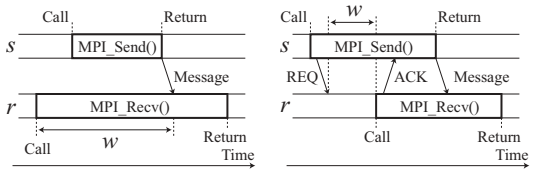
† 大阪大学大学院情報科学研究科コンピュータサイエンス専攻
Department of Computer Science, Graduate School of Information Science and Technology, Osaka University

多くの性能解析ツール[2)~4)]は，実行履歴を基に性能に関するデータを可視化できる．一方，PerWiz[5]は，MPI プログラムの改善可能性を評価するために，特定の通信における実行時間を短縮したと仮定し，プログラムの実行時間を予測する．たとえば，修正候補となる通信の実行時間を 0 秒に仮定することで，その修正により得られる改善効果の上限を予測できる．

このように実行履歴に基づくツールは性能改善において有用である．しかし，イベントごとの情報を記録するため，イベント数の増大とともに実行履歴のファイルサイズが増大する問題がある．さらに，生成した実行履歴をツールが解析するための時間も長くなる．

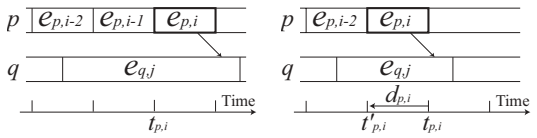
この問題を解決するための手法は，以下の 3 種類に分類できる．(1) 記録集約による記録イベント数の削減[6]，(2) 動的計測（Dynamic Instrumentation）による発生イベント数の削減[7]，および (3) 実行時解析による実行履歴の生成回避[8]．

そこで本研究では，MPI プログラムの改善可能性



(a) 非同期通信　(b) 同期通信

図 1　LogGPS モデルにおける通信



(a) 再構築前　(b) 再構築後

図 2　プログラムの改善可能性を評価するための性能予測

を評価することを目的として，効率的な実行履歴の生成手法を提案する．提案手法は，一部のイベントに対する記録をプログラム実行中に集約することで実行履歴の記録量を削減する．ここで，集約対象とするイベントは，PerWiz による予測結果を変えないものをプログラム実行中に判定する．

以降では，まず 2 章で PerWiz について述べる．次に，3 章で記録を集約できるイベントの条件を定義する．その後，4 章および 5 章でこのイベントをプログラム実行時に判定するアルゴリズムおよびその適用実験を示す．最後に，6 章でまとめを述べる．

2. 性能予測ツール PerWiz の概要

本章では，改善可能性を評価するための PerWiz における性能予測について述べる．

2.1 改善可能性を評価するための性能予測

PerWiz は，評価対象とする MPI プログラム $\mathcal{M}$ の実行履歴 $\mathcal{L}$ を用い，$\mathcal{L}$ における任意のイベントの実行時間を短縮したときの $\mathcal{M}$ の実行時間を予測する．以降では，短縮後における $\mathcal{M}$ の実行時間を T と表す．T を予測するために，まず並列計算モデル LogGPS[9] に基づいて $\mathcal{L}$ を再構築し，新たな実行履歴 $\mathcal{L}'$ を得る．次に，$\mathcal{L}'$ のクリティカルパスを基に T を算出することで，$\mathcal{M}$ の改善可能性を評価する．

図 1 に，LogGPS における非同期通信および同期通信の様子を示す．通信イベントの発生および完了はそれぞれ MPI 関数の呼び出しおよび完了に対応する．図中の w は，同期のための待ち時間を表し，送信プロセス s および受信プロセス r において発生しうる．r では，受信関数の呼び出しからメッセージの到達までの時間を指し，s では，送信要求 REQ の到達から受信関数の呼び出しまでの時間を指す．以降では，イベント a から b への通信を $a \to b$ と表し，プロセス p において i 番目に発生したイベントを $e_{p,i}$ と表す．

図 2 に，PerWiz における再構築の例を示す．この例は，計算イベント $e_{p,i-1}$ の実行時間を 0 秒に短縮した場合で，直後の送信イベント $e_{p,i}$ の発生時刻が $t_{p,i}$ から $t'_{p,i}$ に早まっている．そこで，短縮時における $e_{p,i}$ の完了時刻を定める．$e_{p,i}$ が通信イベントの場合は，LogGPS により定める（2.2 節で後述）．$e_{p,i}$ が計算イベントの場合は，$\mathcal{L}$ における実測の実行時間により定める．同様に，以降のイベントにおける発生時刻および完了時刻を定め，$\mathcal{L}'$ を得る．

以降では，$\mathcal{L}$ および $\mathcal{L}'$ におけるイベント $e_{p,i}$ の発生時刻の差分 $t_{p,i} - t'_{p,i}$ を時刻変化量 $d_{p,i}$ と呼ぶ．

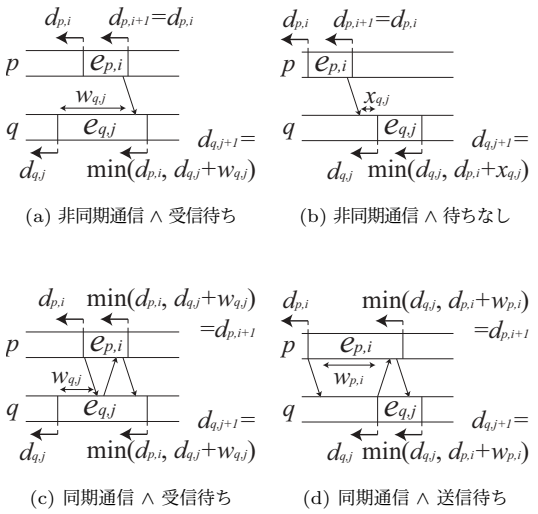
2.2 LogGPS モデルによる性能予測

LogGPS を用いて T を予測する手法について述べる．$\mathcal{L}'$ を得るために，PerWiz は任意のイベント $e_{p,i} \in \mathcal{L}$ に対し，発生時刻 $t_{p,i}$ の順に時刻変化量 $d_{p,i}$ を定める．$\mathcal{L}'$ において最後に発生するイベントの時刻変化量を基に T を予測できる．$d_{p,i}$ は，通信の種類（同期・非同期通信）および待ち時間の有無を組み合わせた 4 通りの場合に応じて一意に定まる．

図 3 は，これら 4 通りの通信 $e_{p,i} \to e_{q,j}$ に対し，$d_{p,i}$ および $d_{q,j}$ に関する漸化式を導出した結果である．図中の $w_{q,j}$ は $e_{q,j}$ における待ち時間を表し，$x_{q,j}$ は非同期通信におけるメッセージの到着から受信関数の呼び出しまでの時間を表す．

以下，受信待ちを伴う非同期通信の場合（図 3(a)）について述べる．まず，通信が非同期であるため，送信待ちは発生しない．ゆえに，$e_{p,i}$ の実行時間は再構築の前後において同一であり，$d_{p,i+1} = d_{p,i}$ となる．一方，受信イベントの実行時間は待ち時間を含む．ゆえに，$d_{q,j+1}$ は $\mathcal{L}'$ における $e_{q,j}$ の待ち時間 $w'_{q,j}$ の有無に依存する．$w'_{q,j} = 0$（すなわち $d_{p,i} \geq d_{q,j} + w_{q,j}$）の場合，$e_{q,j}$ の実行時間は $w_{q,j}$ だけ減少するため，$d_{q,j+1} = d_{q,j} + w_{q,j}$ となる．逆に，$w'_{q,j} > 0$（$d_{p,i} < d_{q,j} + w_{q,j}$）の場合，$e_{q,j}$ の実行時間は $d_{q,j} - d_{p,i}$ だけ増大するため，$d_{q,j+1} = d_{p,i}$ である．まとめると，$d_{q,j+1} = \min(d_{p,i}, d_{q,j} + w_{q,j})$ となる．

残り 3 通りの場合についても同様に定まる．このように，イベントごとの時刻変化量は通信状況に依存した漸化式として定まる．



(a) 非同期通信 ∧ 受信待ち　(b) 非同期通信 ∧ 待ちなし

(c) 同期通信 ∧ 受信待ち　(d) 同期通信 ∧ 送信待ち

図 3　LogGPS モデルによる時刻変化量の定義

3. 記録を集約できるイベントの定義

本章では，PerWiz の性能予測において記録を集約できるイベントの条件を定義する．

3.1 集約条件の定義

$\mathcal{L}$ における任意のイベントに対し，その実行時間を短縮したと仮定する．このとき，$\mathcal{L}$ において記録を集約できるイベント $e_1, e_2, \ldots, e_n$ とは，集約の前後において同一の T を得るものであり，以下に示す条件 R1 および R2 で定義できる．

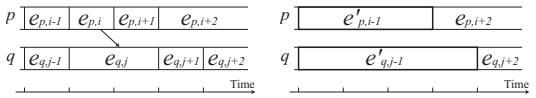
R1：$e_1, e_2, \ldots, e_n$ が連続して発生していること．

R2：$e_1, e_2, \ldots, e_n$ における実行時間の和が短縮の有無に関わらず同一であること．

例えば，n 個の計算イベントが R1 かつ R2 を満たすとき，これらを 1 個の計算イベントに集約したとしても，T の値は変わらない．一方，通信イベント $e_{p,i}$ を含めて集約する場合，条件 R2 を満たすために式 (1) を満たす必要がある．

$$\exists\, e_{q,j} \in \mathcal{L} \mid ((e_{p,i} \rightarrow e_{q,j}) \vee (e_{q,j} \rightarrow e_{p,i})) \wedge (d_{p,i} = d_{q,j}) \quad (1)$$

図 4 に示すように，式 (1) を満たす $e_{p,i}$ （$e_{q,j}$）を前後の計算イベント $e_{p,i-1}$ および $e_{p,i+1}$ （$e_{q,j-1}$ および $e_{q,j+1}$）と集約する．このとき，集約後のイベント $e'_{p,i-1}$ における実行時間は，$e_{p,i-1}$，$e_{p,i}$ および $e_{p,i+1}$ における実行時間の和と定めることにより，実行履歴の記録量を削減できる．また，通信に関する情報 $e_{p,i} \rightarrow e_{q,j}$ を除去し，$e'_{p,i-1}$ を計算イベントとして定義する．なお，$e'_{q,j-1}$ についても同様である．



(a) イベント集約前　(b) イベント集約後

図 4　イベント集約による実行履歴の記録量削減

図 4 を用い，集約の前後において T の値が変わらないことを示す．そのためには，集約により時刻変化量 $d_{p,i+2}$ および $d_{q,j+2}$ が変わらないことを示せばよい．以下，$d_{p,i+2}$ について述べる．まず，集約前において $e_{p,i}$ および $e_{q,j}$ が式 (1) を満たすとき，図 3 より，同期・非同期通信の違いおよび待ち時間の有無に関わらず $d_{p,i+1} = d_{p,i}$ である．次に，計算イベント $e_{p,i-1}$ の実行時間は $\mathcal{L}$ および $\mathcal{L}'$ において同一であるため，$d_{p,i} = d_{p,i-1}$ である．同様に，$d_{p,i+2} = d_{p,i+1}$ である．これら 3 つの式より，集約前において $d_{p,i+2} = d_{p,i-1}$ となる．一方，集約後においては，$e'_{p,i-1}$ が計算イベントであること，さらにその時刻変化量は集約前の $d_{p,i-1}$ であることから，$d_{p,i+2} = d_{p,i-1}$ となる．ゆえに，$d_{p,i+2}$ は集約の前後において同一である．同様に，$d_{q,j+2}$ についても示せる．

3.2 記録を集約できる通信パターン

2.2 節で示したように，時刻変化量 $d_{p,i}$ は直前の $d_{p,i-1}$ およびその通信状況に依存して定まる．以降では，式 (1) を満たす通信 $e_{p,i} \rightarrow e_{q,j}$ を発生させる通信パターンの例を 2 つ示す．

C1：直前の通信 $e_{p,i-2} \rightarrow e_{q,j-2}$ が同期通信である場合．図 3(c) および (d) より，$e_{p,i-2} \rightarrow e_{q,j-2}$ の直後のイベントにおいて $d_{p,i-1} = d_{q,j-1}$ である．さらに，$e_{p,i-1}$ および $e_{q,j-1}$ は計算イベントゆえ，$d_{p,i} = d_{q,j}$ となり，式 (1) が成り立つ．

C2：直前の通信 $e_{p,i-2} \rightarrow e_{q,j-2}$ が受信待ちを伴う非同期通信であり，かつ $d_{p,i-2} \leq d_{q,j-2} + w_{q,j-2}$ を満たす場合．図 3(a) より，$e_{q,j-2}$ の直後において $d_{q,j-1} = \min(d_{p,i-2}, d_{q,j-2} + w_{q,j-2}) = d_{p,i-2}$ となり，$e_{p,i-2}$ の直後において $d_{p,i-1} = d_{p,i-2}$ である．これらの式より $d_{p,i-1} = d_{q,j-1}$ となり，C1 同様 $d_{p,i} = d_{q,j}$ である．

図 5 に，C2 の例を示す．この例では，メッセージがプロセス p および q 間を往復していて，双方において受信待ちが発生している．この通信パターンが C2 を満たすことは，次のように説明できる．なお，表記を簡潔にするため，$a = d_{p,i-4}$，$b = d_{q,j-4}$ とする．

まず，$e_{q,j-4} \rightarrow e_{p,i-4}$ が同期通信の場合，C1 より $d_{p,i-2} = d_{q,j-2}$ となるため，C2 を満たす．次に，非同

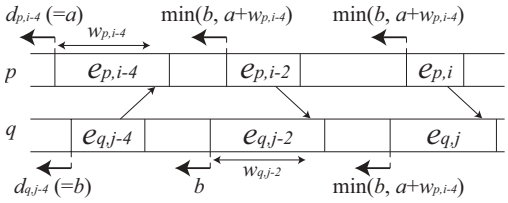


図 5 記録を集約できる通信パターンの例

```
Input: L, trace file containing communication events
Output: L', aggregated trace file
 1: Algorithm Aggregate(L) {
 2:   L' := φ;
 3:   while (L ≠ φ) {
 4:     Select e_{p,i}, e_{q,j} ∈ L such that e_{p,i} → e_{q,j} is
          the earliest paired communication;
 5:     if (IsAggregatable(e_{p,i}, e_{q,j}) = true) {
 6:       Delete e_{p,i} and e_{q,j} from L; // Unadded
 7:     } else {
 8:       Add e_{p,i}, e_{q,j} to L' and delete them from L;
 9:     }
10:   }
11: }
12: function IsAggregatable(e_{p,i}, e_{q,j}) {
13:   if (e_{p,i} or e_{q,j} are for what-if prediction) {
14:     Leave d_{p,i+1} and d_{q,j+1} as variables;
15:   } else { // For Fig. 3(a)
16:     d_{p,i+1} := d_{p,i};
17:     d_{q,j+1} := min(d_{p,i}, d_{q,j} + w_{q,j});
18:   }
19:   if (d_{p,i} = d_{q,j}) { return true; } // Eq. (1)
20:   else { return false; }
21: }
```

図 6 実行履歴を集約するアルゴリズム

期通信の場合，$e_{q,j-4} \rightarrow e_{p,i-4}$ は図 3(a) に分類できるため，$d_{q,j-3} = b$ かつ $d_{p,i-3} = \min(b, a + w_{p,i-4})$ となる．これらの式より $d_{p,i-3} \leq d_{q,j-3}$ である．さらに，$e_{p,i-3}$ および $e_{q,j-3}$ は計算イベントであることから $d_{p,i-2} \leq d_{q,j-2}$ となり，C2 を満たす．

4. 記録を集約できる通信の判定

式 (1) を満たす通信イベントをプログラム実行時に判定し，それらの記録を集約した実行履歴を生成する手法を示す．説明を容易にするために，まずプログラム実行後に実行履歴を集約する手法について述べる．

4.1 実行履歴の集約手法

図 6 に，実行履歴 $\mathcal{L}$ を入力として，集約後の実行履歴 $\mathcal{L}'$ を出力するアルゴリズムを示す．なお，簡単のため，$\mathcal{L}$ は通信イベントの記録からなり，計算イベントの記録を持たないものとする．さらに，受信待ちを伴う非同期通信（図 3(a)）の場合のみを扱う．残りの場合に対しては，図 3 の結果をもとに 16 および 17 行目に漸化式を場合分けとともに追加すればよい．

```
 1: int MPI_Send(message) {
 2:   t_{p,i} := PMPI_Wtime(); // occurence time of e_{p,i}
 3:   Copy message to static_buf with d_{p,i} and t_{p,i};
 4:   PMPI_Send(static_buf);
 5:   d_{p,i+1} := d_{p,i};
 6: }
 7: int MPI_Recv(message) {
 8:   t_{q,j} := PMPI_Wtime(); // occurence time of e_{q,j}
 9:   PMPI_Recv(static_buf);
10:   Extract message, d_{p,i} and t_{p,i} from static_buf;
11:   if (IsAggregatable(e_{p,i}, e_{q,j}) = false) {
12:     Record e_{p,i} and e_{q,j} with t_{p,i} and t_{q,j};
13:   }
14: }
```

図 7 記録を集約できる通信の実行時判定

このアルゴリズムでは，$\mathcal{L}$ において送信および受信の組が成立する通信から順に（4 行目），図 3 の漸化式をもとに時刻変化量を定めていく（16 および 17 行目）．この過程において，式 (1) を満たす通信に対し（19 行目），その記録を省く（6 行目）．一方，式 (1) を満たさない通信は集約できないものとして，$\mathcal{L}'$ に記録する（8 行目）．

なお，PerWiz では任意のイベントに対し，実行時間に関する仮定を置く可能性がある．このような仮定付きのイベントは，その実行時間が変わるため R2 を満たさない．ゆえに，仮定を置く可能性のあるイベントは集約できない．そこで，このアルゴリズムでは，そのようなイベントにおける時刻変化量を変数として扱い評価を保留する（13 および 14 行目）．このように，集約時には変数からなる時刻変化量を用いる．性能予測時には PerWiz が定める仮定に応じて変数に値を代入する．

また，あるイベントに対して仮定を置くか否かは，そのイベントを短縮することにより大きな改善効果を得る可能性があるか否かに基づいて判断する．例えば，メッセージ長があらかじめ定めた閾値を超える通信に対しては，仮定を置く可能性があると判定し，集約対象から除外する．

4.2 プログラム実行時における集約手法

4.1 節の集約手法は，$d_{p,i}$ および $d_{q,j}$ の比較に基づく．一方，プログラム実行時において，各々の値は送信プロセス p および受信プロセス q が保持する．ゆえに，プログラム実行時における集約手法では，$d_{p,i}$ および $d_{q,j}$ をプロセス間でやりとりする必要がある．

そこで，提案手法は記録集約および実行履歴生成のための処理を MPI 関数に追加することで，集約済の実行履歴をプログラム実行時に生成する（**図 7**）．

提案手法では，送信するメッセージに $d_{p,i}$ を付加して送信することで（3 および 4 行目），プログラム実行中に受信プロセス q が $d_{p,i}$ および $d_{q,j}$ を比較する（11 行目）．このとき，q が $e_{p,i}$ を集約するか否かを決定するため，$e_{p,i}$ に関する記録情報も付加して送信し，q が $e_{p,i}$ および $e_{q,j}$ を実行履歴に記録する（12 行目）．

なお，プログラム実行時における時刻変化量 $d_{p,i}$ は変数からなる（4.1 節）．ゆえに，漸化式をもとに時刻変化量を定める（図 6 の 17 行目）たびに，$d_{p,i}$ を構成する変数の数が増加する．この増加は，送信するメッセージに付加するデータの情報量を増加させる．そこで，$d_{p,i}$ を高々m 変数からなる式とすることで，付加情報を m 個に制限する．このように，仮定を置く通信イベントを最新の m 個として，集約の可否を判定する．

5. 適 用 実 験

以下に示す 3 つの観点から，提案手法を評価した結果を示す．

- 実行履歴の記録量
- PerWiz による性能予測のための解析時間
- 実行履歴生成に伴う記録オーバヘッド

実験では，MPI 実装として MPICH-SCore[11] を用い，実行履歴の生成にはその実行履歴生成ライブラリ MPE（Multiprocessing Environment）を使用した．また，MPI プログラムの実行には，64 台の PC をミリネット[12] で相互接続したクラスタを用いた．

評価に用いた MPI プログラムは，1 組の 3 次元画像に対して画像間の位置を対応づけるプログラム[10] である．このプログラムでは，画像をブロック状に分散保持し，画像内に配置した制御点ごとに類似度を表す関数を微分する．制御点 ϕ ごとの微分計算を並行処理するために，ϕ の近傍画素を持つプロセスの集合 $\mathcal{P}_\phi$ が ϕ のための計算および通信を行う．この微分計算を i 回繰り返すことで，位置を合わせる．

5.1 実行履歴の記録量

表 1 に，イベントを集約する場合および集約しない場合の記録イベント数 N_1 および N_2 を示す．ここで，プロセス数は 64 台とし，$m = 3$ とした．

N_1 および N_2 より，提案手法により記録イベント数をおよそ半減できている．また，削減率 σ_N より，$i \leq 3$ では $\sigma_N < 50$ であるのに対し，$i > 3$ では $\sigma_N > 50$ となっている．

この理由は，互いに通信しあうプロセスが増加すると，記録を集約できる可能性が低下するためである．

表 1 実行履歴の記録イベント数および PerWiz の解析時間（プロセス数 64 台，$\sigma_N = 100 \cdot (1 - N_2/N_1)$，$\sigma_T = 100 \cdot (1 - T_2/T_1)$）

繰り返し	記録イベント数（個）		削減率	解析時間（秒）		短縮率
	集約なし	集約あり	(%)	集約なし	集約あり	(%)
i	N_1	N_2	σ_N	T_1	T_2	σ_T
1	50,080	37,292	25.5	338.8	204.3	39.7
2	35,104	22,916	34.7	158.7	66.1	58.3
3	13,888	8,092	41.7	21.0	4.8	77.1
4	21,136	1892	91.0	52.7	0.3	99.4
5	672	0	100.0	0.0	0.0	100.0
6	2,908	56	98.1	0.8	0.0	100.0
総計	182,744	98,412	46.1	572.1	275.5	51.8

このプログラムの場合，通信は同一の $\mathcal{P}_\phi$ に属するプロセス間のみで発生する．ゆえに，すべてのプロセス $p \in \mathcal{P}_\phi$ において時刻変化量が等しくなった場合，式 (1) より，以降これらのプロセス上で発生するイベントの記録を集約できる．ただし，$\mathcal{P}_\phi$ に属さないプロセスから $p \in \mathcal{P}_\phi$ への通信が発生すると，式 (1) を満たせず，以降の記録を集約できなくなる．

実際に，$\mathcal{P}_\phi$ を構成するプロセス数 $|\mathcal{P}_\phi|$ は，σ_N が大きい $i = 4$ および 6 では各々$|\mathcal{P}_\phi| = 4$ および 2 であり，σ_N が小さい $i = 1$ では $|\mathcal{P}_\phi| = 12$ であった．

まとめると，提案手法が効果的に記録を集約できるプログラムとは，互いに通信しあうプロセスが少なく，それらのプロセス間における通信が繰り返し発生する通信パターンを持つものである．

5.2 PerWiz による性能予測のための解析時間

集約なしおよび集約ありの実行履歴に対し，PerWiz による性能予測のための解析時間 T_1 および T_2 を調べた（表 1）．提案手法により解析時間を 572.1 秒から 275.5 秒に短縮でき，およそ半減できている．

削減率 σ_N および短縮率 σ_T を比較すると，総じて $\sigma_N \leq \sigma_T$ であることから，記録イベント数の削減効果を超えて解析時間を短縮できている．この理由は，記録イベント数を N とすると，性能予測のための時間計算量が最悪で $O(N^2)$ であり，最良で $O(N)$ であるためである．ゆえに，総じて $\sigma_N \leq \sigma_T$ となった．

なお，集約の有無に関わらず，PerWiz が予測した改善可能性が大きい通信の順序は同一であった．このように，記録を集約することにより実行履歴のサイズを 9 MB から 5 MB に削減でき，解析時間を短縮できた．

5.3 実行履歴生成に伴う記録オーバヘッド

提案手法における記録オーバヘッドを示す．記録オーバヘッドは以下の 3 種類に分類できる．

O1：ファイルへの出力（図 7 の 12 行目）

O2：記録を集約できる通信の判定（図 7 の 11 行目）

O3：付加情報のやりとり（図 7 の 3，4，9，10 行目）

O1 は集約の有無に関わらず，実行履歴を生成する手法において共通に発生する．例えば，20 B からなるイベントごとの記録（イベントの種類，発生時刻，発生回数および通信相手）に対し，MPE 関数が要する記録時間は 5 マイクロ秒程度である．提案手法は記録イベント数を半減できたため，O1 を減少できる．

残る O2 および O3 は，提案手法における固有の記録オーバヘッドであり，いずれの時間も m に比例する．$m = 3$ の場合，O2 が要する時間は 1 マイクロ秒以下であった．ゆえに，O2 は O1 と比較して無視できる．

一方，O3 の影響を調べるために，付加なしおよび付加ありの往復遅延時間（RTT：Round Trip Time）T_3 および T_4 を，メッセージ長 l ごとに計測した．

$m = 3$ のとき，T_3 に対する O3（$T_4 - T_3$）の比率は常に 8% 以上であり，l とともに増大する．ここで，メッセージのコピー（図 7 の 3 および 10 行目）が O3 に対して占める比率が高く，l とともに増大し，$l \geq 20$ KB においては 99%であった．

よって，記録オーバヘッド O3 は無視できないため，付加情報のやりとりにおけるメッセージのコピーに要する時間を短縮する必要がある．

6. ま と め

本稿では，MPI プログラムの改善可能性を評価するための効率的な実行履歴の生成手法を提案した．提案手法は，実行履歴の記録量を削減するために，PerWiz よる実行時間予測に影響しない通信命令をプログラム実行中に特定し，その記録を集約する．

提案手法を用いて記録を集約した結果，実行履歴の記録量および PerWiz の解析時間をほぼ半減できた．ゆえに，提案手法は MPI プログラムの改善可能性を評価する際に有用であると考える．

今後の課題としては，記録を集約できるイベントの条件および通信パターンを増やすことが挙げられる．

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) Heath, M.T. and Etheridge, J.A.: Visualizing the Performance of Parallel Programs, *IEEE Software*, Vol.8, No.5, pp.29–39 (1991).
3) 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).
4) Zaki, O., Lusk, E., Gropp, W. and Swider, D.: Toward Scalable Performance Visualization with Jumpshot, *Int'l J. High Performance Computing Applications*, Vol.13, No.2, pp.277–288 (1999).
5) 神戸友樹, 置田真生, 伊野文彦, 萩原兼一：メッセージ通信プログラムの改善可能性を評価するための性能予測ツール, 情報処理学会論文誌：コンピューティングシステム, Vol.44, No.SIG 11(ACS 3), pp.101–110 (2003).
6) Yan, J.C. and Schmidt, M.A.: Constructing Space-Time Views from Fixed Size Trace Files — Getting the Best of Both Worlds, *Proc. Conf. Parallel Computing (ParCo'97)*, pp.633–640 (1997).
7) Yan, J., Sarukkai, S. and Mehra, P.: Performance Measurement, Visualization and Modeling of Parallel and Distributed Programs using the AIMS Toolkit, *Software: Practice and Experience*, Vol.25, No.4, pp.429–461 (1995).
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) 伊野文彦, 藤本典幸, 萩原兼一：LogGPS：メッセージ通信プロトコルの切り替えを考慮した高水準通信ライブラリ向けの並列計算モデル, 情報処理学会論文誌：ハイパフォーマンスコンピューティングシステム, Vol.42, No.SIG 9(HPS 3), pp.145–157 (2001).
10) Schnabel, J.A., Rueckert, D. and et al.: A Generic Framework for Non-rigid Registration Based on Non-uniform Multi-level Free-Form Deformations, *Proc. 4th Int'l Conf. Medical Image Computing and Computer-Assisted Intervention (MICCAI'01)*, pp.573–581 (2001).
11) 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).
12) 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).