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

神戸友樹[†] 置田真生[†]
伊野文彦[†] 萩原兼一[†]

本稿では，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 PerWiz（Performance Wizard）を提案する．PerWiz の特長は，プログラムにおいて大きな改善効果が期待できる箇所を指摘する点にある．この指摘を実現するために，PerWiz はプログラムの実行履歴を並列計算モデル LogGPS に基づいて解析し，特定の通信における待ち時間を 0 秒にしたとき，あるいは計算負荷を均等にしたときなどの仮定のもとでプログラムの実行時間を予測する．また本稿では，PerWiz を用いて性能を改善できた例を 2 つ示す．これらの例より，PerWiz はメッセージ通信プログラムを修正するための労力，およびその修正がもたらす改善効果を検討できる点で有用であると考える．

A Performance Prediction Tool for Evaluating Potential Improvement of Message Passing Programs

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

This paper proposes a performance prediction tool, named Performance Wizard (PerWiz), for evaluating the potential improvement of message passing programs. PerWiz has a novel facility for indicating where significant improvement can be established. To indicate this, PerWiz performs post-mortem analysis based on a parallel computational model, LogGPS, so that predicts what performance will be obtained if developers eliminate wait time for a specific message or balance workloads in the target program. We also present two case studies where PerWiz has been shown to play a key role in their improvements. From these studies, we believe that PerWiz allows developers to investigate the effort to modify message passing programs and the effect of performance improvement derived from the modification.

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．このパラダイムは，これらの計算環境において性能の良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，性能予測および性能解析に関する研究[2)~8)] が行われている．文献 2) では，並列プログラムの実行を DAG（DirectedAcyclicGraph）を用いて表し，DAG の CP（CriticalPath）に着目した支援を行っている．具体的には，特定の手続きの実行時間を 0 秒と仮定したときの，プログラム全体の実行時間（改善後の上限）を実行時に予測している．この手法は，計算量の削減による性能改善において有用であり，開発者は大きな改善効果が期待できる手続きを特定できる．

また文献 3) では，計算負荷を均等にするために，プロセッサに対するプロセス割当てを変更したときの実行時間を予測している．この手法は，プログラム修正をともなわない負荷分散を試みる際に有用であり，開発者は良い性能が得られるプロセス割当てを特定できる．

一方，性能に関する情報を可視化する研究[4)~8)] も多数ある．Jumpshot [4] はプログラム実行時におけるプロセス間通信の様子を時間軸に沿ったタイムライン図として示す．ParaGraph [5] はプログラムの実行履歴を基にその動作および性能に関する情報を 25 種類の観

† 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

点から可視化でき，多角的な解析が行える．Virtue[6]では，グリッド上で実行中のプログラムに対し，地理的に離れた開発者が協調してその性能に関する情報を可視化できる．かのこ[7]は，人間が力学的なバランスや動作の変化を容易に識別できることに着目していて，力学系モデルに基づくアニメーションが大域的な性能ボトルネックの発見に役立つ．文献 8) は，624 台からなる大規模クラスタに対する監視を実現していて，トポロジを考慮して 3 次元状に描かれたネットワークのうえでクラスタが実行する複数のプログラムの通信量を可視化できる．これら可視化による手法はプログラムの動作を直観的に理解できる．しかし，動作に関する情報の増加とともに可視化結果が複雑になり（視認性が低下し），開発者の見落としが問題となる[9]．

このように，並列プログラムの性能改善を支援するための研究は多い．しかし，プログラムを修正する前に修正後の性能を予測する研究は，我々の知る限り文献 2) のみである．ただし文献 2) では，通信にともなう同期待ちの削減や負荷分散などの並列処理固有の改善手法をプログラムに適用したときの予測を考慮していない．これらの予測は，開発者が修正を行うか否かを判断する際に有用であると考えられる．

そこで，本研究ではメッセージ通信プログラムの改善可能性を評価することを目的として，特定の通信の待ち時間を 0 秒と仮定したときのプログラム全体の実行時間を予測できるツール PerWiz（PerformanceWizard）を提案する．PerWiz の特長は，先の仮定を置いたときに大きな改善効果が期待できる通信を指摘する点にある．さらに，開発者が並列実行を意図する部分（**並列領域**）をプログラム内で指定することにより，その部分の計算負荷などが均等であると仮定したときの実行時間を予測できる．PerWiz の指摘は，プログラムを修正するための労力およびその改善効果を検討する際に有用であり，開発者は性能改善のための修正作業を効率良く進めることができる．

現在の PerWiz は，メッセージ通信仕様 MPI（Message Passing Interface）[1] を用いて記述された並列プログラムを対象としている．PerWiz の予測は，並列計算モデル LogGPS[10] に基づいてプログラムの実行履歴を再構築することで実現していて，その予測結果はテキスト形式の出力とともに，既存の性能可視化ツール logviewer[4),11)] において視認できる．

以降では，まずメッセージ通信プログラムの実行をモデル化する（2 章）．次に，大きな改善効果が期待できる通信をモデル上で指摘する手法について述べる（3 章）．その後，この手法に基づく PerWiz について述べ（4 章），PerWiz の適用例およびその考察を示す（5 章）．最後に，まとめを述べる（6 章）．

```
1: PRGN_BEGIN;
2: for (k=0; k<N; k++) {
3:     MPI_Sendrecv();
4:     Calculation;
5: }
6: PRGN_END;
```

(a) 繰り返し全体を並列実行

```
1: for (k=0; k<N; k++) {
2:     PRGN_BEGIN
3:     MPI_Sendrecv();
4:     Calculation;
5:     PRGN_END
6: }
```

(b) 繰り返しの各々を並列実行

図 1 メッセージ通信プログラムにおける並列領域の指定例
Fig. 1 Example of parallel region specified in message passing programs.

2. メッセージ通信プログラム

本章では，DAG および LogGPS モデルを用いてメッセージ通信プログラムの実行をモデル化する．なお，LogGPS モデルを並列計算モデルとして選択した理由は後述する（5.3 節）．

2.1 並列領域の定義

メッセージ通信プログラム $\mathcal{A}$ における計算負荷の偏りを解析するためには，開発者の意図，すなわち並列実行を意図する処理を $\mathcal{A}$ 内で特定する必要がある．そこで，本節では並列領域という概念を定義する．

並列領域 $\mathcal{R}$ とは，開発者が並列実行を意図する複文 a を指す（a は $\mathcal{A}$ の一部）．すなわち，同一の $\mathcal{R}$ に属する任意の複文 $a \in \mathcal{R}$ は並列に実行されうることを意味し，実際に a が並列実行されているか否かは問わない．なお，a は並列処理にともなう通信のための関数呼び出しを含み得る．

図 1 に，メッセージ通信プログラムに対して並列領域を指定した例を 2 つ示す．PRGN_BEGIN および PRGN_END の間の複文が並列領域を表している．図 1 (a) が N 回の繰返し全体を並列化対象とするのに対し，図 1 (b) は繰返しの各々を並列化対象とする．

以降では，並列領域の 1 回の実行に対応する論理的な処理ステップを**並列ステップ**と呼ぶ．

2.2 プログラム実行のモデル化

$\mathcal{A}$ の 1 回の実行は，頂点の集合 $\mathcal{V}$ および辺の集合 $\mathcal{E}$ からなる DAG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ で表せる．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依存関係 $\rightarrow$[12] に対応する．

イベントは，任意のプロセスが $\mathcal{A}$ 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と呼び，通信関数の種類に応じて送信イベントおよび受信イベントに区別する．また，通信関数の完了から同一プロセスにおける次の通信関数の呼び出し

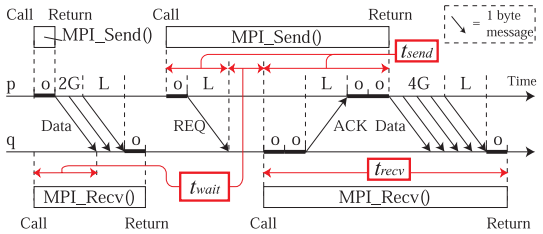


図 2 LogGPS モデルにおける非同期通信と同期通信
Fig. 2 Asynchronous and synchronous messages under LogGPS.

までの計算に対応するイベントを計算イベントと呼ぶ．

$\mathcal{G}$ の頂点および辺に重みを付加し，$\mathcal{G}$ の CP を定めることにより，$\mathcal{A}$ の 1 回の実行時間を表せる．そのための方法として，並列計算モデルを用いる方法がある．LogGPS モデル[10] は，LogP [13] および LogGP モデル[14] を基に，以下に示す 7 種類のパラメータを用いて非同期通信および同期通信をモデル化する（**図 2**）．

- L：通信遅延．送信プロセッサから受信プロセッサまでのネットワーク転送に要する時間．
- o：通信オーバヘッド．メッセージを送信もしくは受信するときにプロセッサが占有される時間．
- g：メッセージを連続して送信もしくは受信するときの最短の時間間隔．
- G：長いメッセージを送信するときの 1 バイトあたりの時間間隔．
- P：プロセッサ数．
- S：非同期で送信できるメッセージ長の上限値．
- s：1 パケットで送信できるメッセージ長の上限値．

なお，正確には o はメッセージ長に関する線形関数で表し送信側および受信側で区別すべきだが[10]，本稿では説明を容易にするために記述を簡略化した．

図 2 に示すように，送信イベント u および受信イベント v の処理時間は，メッセージ転送のための送信時間 t_{send} もしくは受信時間 t_{recv} に加えて，同期のための待ち時間 t_{wait} を含みうる．ここで，u における待ち時間とは，送信要求 REQ の到着から受信関数の呼び出しまでの時間を指す．一方，v における待ち時間とは，受信関数の呼び出しからメッセージの到着までの時間を指す．以降，u（v）の待ち時間を 0 秒と仮定するとは，u（v）の組となる通信イベント v（u）の開始時刻を早め，$t_{wait} = 0$ を得ることを指す．

3. メッセージ通信プログラムにおける改善可能性の評価

メッセージ通信プログラム $\mathcal{A}$ の処理は，通信処理および計算処理に分類できる．以降では，各々の処理に着目し，$\mathcal{A}$ の改善可能性を評価する手法を示す．

3.1 通信処理に対する改善可能性の指摘

通信のための処理時間を短縮する手法として，(I1) 同期のための待ち時間の短縮，および（I2）通信量の削減があげられる．ここでは，通信に固有の改善手法として（I1）に着目する．なお，(I2) に対しては，文献 2) の手法を通信関数に応用することで，メッセージ長を 0 バイトと仮定したときの改善の上限を予測できる．

一般に，待ち時間 t_{wait} が最長の通信イベント $u \in \mathcal{V}$ を修正することが大きな改善をもたらすとは限らない．たとえば，u に対して $t_{wait} = 0$ とできたとしても，$\mathcal{G}$ の CP が短縮しなければ，性能改善に到らない．また逆に，t_{wait} が短い通信イベントを修正することで，ドミノ倒しのように以降の通信イベントの t_{wait} を短縮できることもある（**ドミノ効果**）．このように，最長の t_{wait} に着目することが必ずしも大きな改善をもたらすとはいえないため，CP すなわちプログラム全体の実行時間 T を予測する必要がある．

そこで，少ない労力で大きな改善効果を得ることを目的として，ドミノ効果をもたらす通信イベントを指摘する手法を提案する．提案手法では，与えられた DAG $\mathcal{G}$ の依存関係 $\rightarrow$ を再帰的に遡りながら，特定の通信イベント $u \in \mathcal{V}$ に対して $t_{wait} = 0$ を仮定したときの T を繰り返し予測する．さらに，依存関係を遡る過程で特定できた u からなるパス（**ドミノパス**）を指摘する．なお，T の予測には t_{wait} を解析できる LogGPS モデルを用いる（4.2 節）．

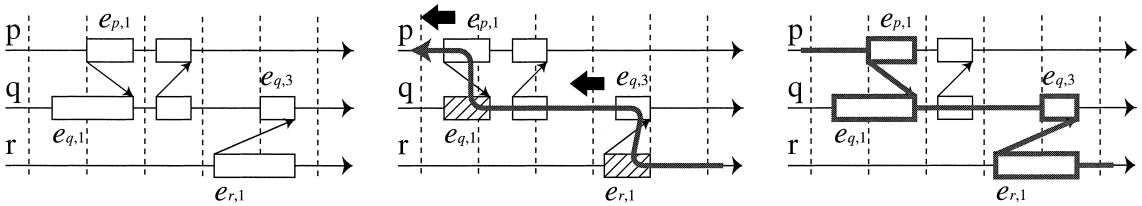
図 3 に，提案手法を示す．提案手法は，DAG $\mathcal{G}$ およびプロセスの集合 $\mathcal{P}$ を入力として，最短の実行時間 T を与えるドミノパスの集合 $\mathcal{D}$ を返す．なお，説明を容易にするために図 3 は処理の一部を簡略化している（後述）．以降では，プロセス p において i 番目に発生したイベントを $e_{p,i}$ と表す．

まず，p において最後に発生した $e_{p,i}$ を選択し（6 行目），p が $e_{p,i}$ 以前に処理した $e_{p,j}$ $(1 \leq j \leq i)$ のうち，$t_{wait} = 0$ を仮定したときに最短の実行時間 $T_{p,m}$ を与える $e_{p,m}$ のイベント番号 m $(1 \leq m \leq i)$ を探す（17 行目）．ここで，m は 1 個に特定できるとして簡略化しているが，実際には各々の m について総当たりの予測を以降で行う．

次に，$T_{p,m}$ がこれまでに予測した最短の実行時間 T_p よりも大きな値となる場合，もしくはドミノパス D_p がすでに $e_{p,m}$ を含む場合は，依存関係を遡ることを停止する（19 行目）．そうでない場合は，依存関係を基に $e_{p,m}$ と組を構成する通信イベント $e_{q,j}$ を調

```
Inputs: (1) G = (V, E), Direct acyclic graph
        (2) P, Set of processes
Outputs: (1) D, Set of domino paths
         (2) T, Predicted time if wait time zeroing
 1: Algorithm SamplingDominoPath(G, P, D, T);
 2: begin
 3:   T := ∞;
 4:   foreach process p ∈ P do begin
 5:     D_p := φ; // D_p: Domino path terminated at e_{p,i}
 6:     Select event e_{p,i} ∈ V such that
          ¬∃e_{p,j} ∈ V (e_{p,i} → e_{p,j});
 7:     w(D_p) := SamplingForEachProcs(G, e_{p,i}, ∞);
 8:     if (T > w(D_p)) then T := w(D_p); // Minimum
 9:   end
10:   D := {D_p | w(D_p) = T};
11: end
12: Function SamplingForEachProcs(G, e_{p,i}, T_p);
13: begin
14:   foreach event id j ∈ {1, 2, ···, i} do begin
15:     T_{p,j} := SimulateWaitTimeZeroing(e_{p,j}); // §4.2
16:   end
17:   Select event id m ∈ {1, 2, ···, i} such that
          ∀j ∈ {1, 2, ···, i} (T_{p,j} ≥ T_{p,m});
18:   if ((T_{p,m} > T_p) ∨ (D_p contains e_{p,m})) then
19:     return T_p;
20:   else begin
21:     Select event e_{q,j} ∈ V such that
          (p ≠ q) ∧ ((e_{q,j} → e_{p,m}) ∨ (e_{p,m} → e_{q,j}));
22:     Add events e_{p,m} and e_{q,j} to D_p;
23:     return SamplingForEachProcs(G, e_{q,j}, T_{p,m});
24:   end
25: end
```

図 3 ドミノパスを指摘するアルゴリズム
Fig. 3 Algorithm for computing domino path.



(a) 元の実行 (b) 依存関係を遡る様子 (c) ドミノパス

図 4 ドミノパスを指摘する際の流れ
Fig. 4 Computing process for domino path.

べ（21 行目），$e_{p,m}$ および $e_{q,j}$ を D_p に加える（22 行目）．その後，$e_{q,j}$ について同様の予測を再帰的に行う（23 行目）．以上の予測をすべてのプロセスについて行い（4 行目），T および $\mathcal{D}$ を返す（8，10 行目）．

図 4 に，ドミノパスを指摘する際の流れを示す．図 4 では，$e_{p,1}$，$e_{q,1}$，$e_{q,3}$ および $e_{r,1}$ からなるドミノパスを指摘していて，$e_{r,1} \rightarrow e_{q,3}$，$e_{q,1} \rightarrow e_{p,1}$ という順に依存関係を遡る（図 4 (b)）．$e_{p,1}$ の発生を早めることで，$e_{q,1}$ および $e_{r,1}$ の待ち時間を短縮できる．

3.2 計算処理に対する改善可能性の指摘

計算のための処理時間を短縮する手法として，(I3) プロセス間の負荷分散および（I4）計算量の削減があげられる．(I4）に対しては，(I2）と同様に文献 2) の手法を適用することで，その改善の上限を予測できる．

（I3）に対しては，プログラム中に並列領域を指定することで，その改善可能性を指摘できる．つまり，並列領域内の計算時間がプロセス間で均等であると仮定したときの T を予測する．さらに，k 番目の並列ステップ S_k におけるプロセスごとの計算時間 t_k の標準偏差 $\sigma(t_k)$ を基に，計算負荷の偏りが大きい並列ステップを指摘できる．ただし，(I1）と同様に，最大の $\sigma(t_k)$ に着目することが大きな改善をもたらすとは限らないため，計算時間が均等である（$\sigma(t_k) = 0$）と仮定したときに最短の T を与える S_k を指摘する．

なお，並列領域は通信を含みえる．通信に対しても計算と同様に，並列領域内の通信量がプロセス間で均等であるなどの仮定のもとで T を予測することにより，多様な改善手法を想定できる．

4. PerWiz：性能予測ツール

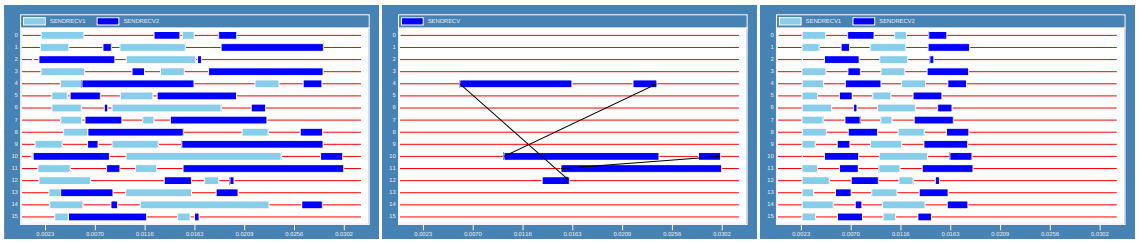
本章では，提案する性能予測ツール PerWiz の概要について述べ，LogGPS による予測手法を示す．

4.1 概　　要

現在の PerWiz は，ALOG 形式[11] の実行履歴に対して指摘を行う．ALOG 形式は，多くの MPI 実装の基となる MPICH[15] が対応していて，C/C++/Fortran 言語で記述したプログラムで利用できる．

PerWiz の機能は，以下の F1〜F3 に分類できる．

- **F1：改善候補を指摘する機能**
  F1 は，少ない労力で大きな改善効果を得ることを支援する（**図 5**）．通信処理に対しては，ドミノ効果をもたらす通信イベントをドミノパスとして



(a) 元の実行 (b) ドミノパスの指摘 (c) 負荷分散時の予測

図 5 logviewer のタイムライン図による指摘結果の可視化（プログラム修正前の実行履歴に基づく）
Fig. 5 Predicted results in timeline view visualized by logviewer.

指摘する（3.1 節）．一方，計算処理に対しては，計算負荷を均等にすることで大きな改善をもたらす並列ステップを指摘する（3.2 節）．

- **F2：プログラム修正後の実行時間を予測する機能**
  F2 は，F1 の基本となる機能である．LogGPS モデルに基づいて DAG を再構築することにより，特定の仮定のもとでプログラム全体の実行時間 T を予測する（図 5 (c)）．開発者が指定できる仮定は以下の 4 通りである．
  – 特定のイベントにおける待ち時間もしくは実行時間を 0 秒としたとき
  – 特定の並列ステップにおける計算時間もしくは通信量をプロセス間で均等にしたとき
  なお，各々の仮定を組み合わせることで，開発者は多様な改善手法を想定して性能を予測できる．
- **F3：性能改善の上限を示す機能**
  F3 は，並列領域における性能の上限を調べるための機能であり，現在の並列化手法にこだわらない改善を試みることを支援する．実行履歴におけるプロセスごとの計算時間，通信時間および待ち時間の和を基に T を予測する．つまり，F2 が DAG を再構築して適切な同期待ちを再現するのに対し，F3 は DAG を再構築せずに各々の時間を加算する．開発者は以下に示す 4 通りの仮定を指定でき，これらを組み合わせた予測が行える．
  – 特定の並列領域内における待ち時間もしくは通信時間を 0 秒としたとき
  – 特定の並列領域内における計算時間もしくは通信量をプロセス間で均等にしたとき

図 6 に，PerWiz を用いた性能改善の手順を示す．開発者はまず改善対象とする MPI プログラムに対して並列領域を指定する（図 1）．指定済 MPI プログラムをコンパイルし実行履歴生成ライブラリ libmpipw.a とリンクすることで，実行形式ファイル $\mathcal{B}$ を得る．$\mathcal{B}$ を並列実行することで ALOG 形式の実行履歴 $\mathcal{L}$ を生成し，$\mathcal{L}$ および上記の仮定を PerWiz に与えることで指摘結果をテキスト形式の出力および ALOG 形式の実行履歴 $\mathcal{L}'$ として得る．開発者は，これらの情報を基に改善手法を決定し，プログラムを修正する．なお，$\mathcal{L}'$ を PerWiz に与えることで，プログラムを再実行することなく予測を繰り返すことができる．

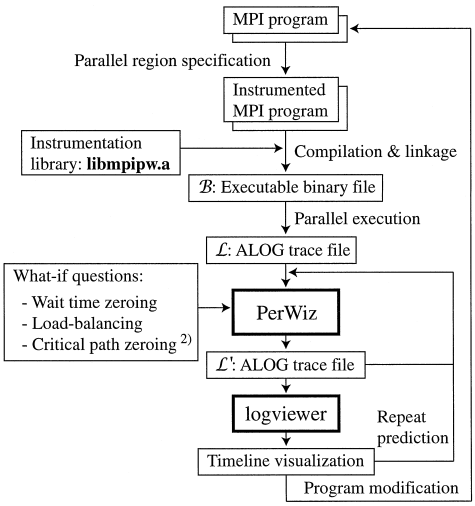


図 6 PerWiz を用いた性能改善の手順
Fig. 6 Performance improvement process using PerWiz.

4.2 実行時間の予測

LogGPS モデルに基づいて実行履歴 $\mathcal{L}$ を再構築し，$\mathcal{L}'$ を得る手法について述べる．

精度の良い予測を実現するために，PerWiz では $\mathcal{L}$ が持つイベントごとの処理時間を可能な限り再利用する．計算イベントに対しては $\mathcal{L}$ における処理時間を $\mathcal{L}'$ に適用する．一方，通信イベントに対しては，非同期通信および同期通信に応じて手法を使い分ける．

まず非同期通信に対しては，LogGPS モデルにおけるパラメータを用い，メッセージ長に応じた処理時間[10] を $\mathcal{L}'$ に適用する．一方，同期通信に対しては，$\mathcal{L}$ における送信時間 t_{send} および受信時間 t_{recv} を推定し，これらを $\mathcal{L}'$ に適用する．以下，送信イベント u の場合について述べる．まず $\mathcal{L}$ における

```
1: PRGN_BEGIN;
2: for (φ=1; φ ≤ N; φ++) {
3:    if (MPI_Comm_rank() ∈ process group g(φ)) {
4:       Local calculation for differential;
5:       Communication within g(φ) by MPI_Send(),
             MPI_Recv(), and MPI_Sendrecv();
6:    }
7: }
8: PRGN_END;
```

図 7 位置合わせプログラムにおける並列領域の指定
Fig. 7 Registration program and its parallel region.

u の処理時間 t_{all} を，$t_{send} = t_{all} - t_{wait}$ および $t_{wait} = max(t_v - t_u - (o+L), 0)$ に分解する．ここで，t_u および t_v は $\mathcal{L}$ における u の発生時刻および u の組となる受信イベント v の発生時刻を表し，$o+L$ はREQが受信側に到達するまでの時間を表す（図2）．そのうえで，$\mathcal{L}'$ を構築する過程で確定する，u および v の発生時刻 t'_u および t'_v を用いて，$\mathcal{L}'$ における u の処理時間を $t'_{all} = max(t'_v - t'_u - (o+L), 0) + t_{send}$ とする．受信イベントの場合も同様に定める．

5. 適用例およびその考察

本章では，PerWiz を用いて MPI プログラムの性能を改善できた例について述べ，その有用性を示す．さらに，PerWiz の予測結果に対する LogGPS モデルの効果および大規模プログラムに対する PerWiz の適用可能性について考察する．

MPI プログラムの実行には，双方向 2 Gb/s のミリネット（Myrinet）[16] および 100 Mb/s のイーサネットで相互接続された PC クラスタ（64 台）を用いた．また，MPI 実装には MPICH-SCore [17] を用いた．

なお，例として用いたプログラムでは非同期通信が発生しなかったため，LogGPS モデルのパラメータのうち L および o のみが予測結果に影響した（4.2 節）．これらの値は，文献 10) の計測手法を用いて $L = 9.11$ マイクロ秒および $o = 2.15$ マイクロ秒とした．

5.1 通信における待ち時間の短縮による性能改善例

待ち時間を短縮することで性能を改善できた例として，1 組の 3 次元画像に対して，画像内の位置を対応づける位置合わせプログラム[18] をあげる．

図 7 に，プログラムの概要を並列領域とともに示す．このプログラムでは，画像をブロック状に分散保持し，画像内に配置された制御点ごとに類似度を表す関数を微分する．開発者は，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 $g(\phi)$ が ϕ のための計算および通信（4，5 行目）を行うよう記述した．

表 1 PerWiz による性能改善の上限予測（機能 F3）
Table 1 Upper bound on improvement predicted by PerWiz (Function F3).

並列領域に対する仮定	予測した実行時間	
A1：待ち時間が 0 秒	位置合わせ	画像合成
A2：計算時間が均等	プログラム	プログラム
A3：通信量が均等	T_1（秒）	T_2（ミリ秒）
—	23.8	30.6
A1	8.2	22.3
A2	23.4	34.9
A3	21.4	25.7
A1 ∧ A2	5.3	16.5
A1 ∧ A3	8.0	16.8
A2 ∧ A3	23.6	38.2
A1 ∧ A2 ∧ A3	5.0	11.0

まず，並列領域を指定するための関数呼び出しを元のプログラムに追加し，プロセス数 $P = 16$ として実行履歴 $\mathcal{L}$ を生成した．このときのプログラム全体の実行時間 T_M は 23.8 秒であり，$\mathcal{L}$ のファイルサイズおよび通信イベント数は各々720 KB および 4,634 個であった．次に，PerWiz の機能 F3 を用いて性能改善の上限を調べた．**表 1** に，各々の仮定のもとで予測した実行時間 T_1 を示す．

表 1 より，仮定 A1 を指定した場合は $T_1 < 10$ であるのに対し，そうでない場合は $T_1 > 20$ であることが分かる．ゆえに，性能改善のためには，待ち時間を短縮することが不可欠であるといえる．

そこで，ドミノパスが指摘するイベントもしくは最長の待ち時間を持つイベントに着目し，制御点（繰り返しの各々）の処理順序を開発者が入れ換えることで，これらの待ち時間を短縮した．**表 2** に，プログラムの修正を l $(0 \le l \le 7)$ 回繰り返したときの結果を示す．なお，ドミノパスの指摘には Pentium III 1 GHz 上で 1 回あたり 10 秒を要した．

表 2 より，プログラムを 7 回修正することで，ドミノパスに着目した場合は実行時間 $T_M^{(l)}$ を 23.8 秒から 13.7 秒（修正前の 57.6%）まで短縮できている．一方，最長の待ち時間に着目した場合は 20.8 秒（修正前の 87.1%）であるため，ドミノパスに着目した修正は効率良くプログラムの性能を改善できている．

また，l 回目に着目した通信イベントの待ち時間 $t_{wait}^{(l)}$ を比較すると，最長の待ち時間に着目した場合は任意の l に対して $t_{wait}^{(l)} > 10$ であるのに対し，ドミノパスに着目した場合は修正とともに減少している．この理由はドミノ効果にあり，すなわち 1 回の修正で複数のイベントの待ち時間を短縮できるためである．実際に，ドミノパスに着目した場合は，待ち時間を修正 1 回当たり累計 25 秒短縮できた．一方，最長

表 2 ドミノパスに着目した修正および最長待ち時間に着目した修正の比較（位置合わせプログラム）
Table 2 Comparison of modification results achieved by focusing on domino path and longest wait time in registration program.

修正回数	ドミノパスに着目した修正				最長待ち時間に着目した修正		
	着目待ち時間	l 回目の実行時間（秒）		短縮率（%）	着目待ち時間	l 回目の実行時間	短縮率（%）
l	$t_{wait}^{(l)}$（秒）	予測 $T^{(l)}$	実測 $T_M^{(l)}$	$100 \times T_M^{(l)}/T_M^{(0)}$	$t_{wait}^{(l)}$（秒）	実測 $T_M^{(l)}$（秒）	$100 \times T_M^{(l)}/T_M^{(0)}$
0	—	—	23.8	—	—	23.8	—
1	11.8	21.6	21.8	91.4	12.5	23.9	100.2
2	10.7	19.5	19.5	82.0	14.7	22.6	94.7
3	7.5	18.2	18.2	76.4	11.9	22.6	94.7
4	5.8	15.9	15.8	66.4	13.5	21.5	90.1
5	3.4	14.8	14.8	62.0	11.4	21.5	90.1
6	0.4	14.4	14.4	60.4	12.3	20.7	87.0
7	0.7	13.7	13.7	57.6	10.7	20.8	87.1

の待ち時間に着目した場合は，累計 0〜10 秒の短縮であった．

さらに，短縮率 $100 \times T_M^{(l)}/T_M^{(0)}$ より，ドミノパスに着目した場合はプログラムを修正するたびに性能を改善できているが，最長の待ち時間に着目する場合は $l = 1$，3，5 および 7 において性能を改善できていない．この理由は，ドミノパスによる指摘では CP が短縮する箇所をモデル上で探索するのに対し，最長の待ち時間による指摘では必ずしも CP が短縮しないためである．たとえば，$l = 1$，3，5 および 7 では，$t_{wait}^{(l)}$ を短縮することで，以降の通信イベントにおいて待ち時間を増大させていた．

このように，PerWiz がドミノパスをその改善効果とともに指摘することにより，開発者はプログラムの改善可能性をあらかじめ知ることができ，性能改善のための修正作業を効率良く進めることができた．

5.2 負荷分散による性能改善例

負荷を分散することで性能を改善できた例として，3 次元データの可視化（ボリュームレンダリング）のための画像合成を行うプログラム[19] をあげる．

図 8 に，プログラムの概要を並列領域とともに示す．P 台のプロセスが利用できるとき，この画像合成プログラムは $\log P$ ステージで各プロセスの部分画像 P 枚を最終画像に合成する．各合成ステージにおいて，プロセスは組を構成し部分画像の半分（ブロック）を交換しあう．組を変更しながら $\log P$ 回の交換および合成を繰り返すことで，最終画像を得る．

理論上，k（$1 \le k \le \log P$）番目の合成ステージにおいて任意のプロセスは $1/2^k$ の画像領域を担当するため，計算負荷の偏りは生じない．しかし，透明な画像領域に対する合成は省略できるため，プログラムではこの工夫による偏りが生じていた．

まず，並列領域を指定するための関数呼び出しを元のプログラムに追加し，$P = 64$ として $\mathcal{L}$ を生成した．このときの実行時間 T_M は 30.6 ミリ秒であり，$\mathcal{L}$ のファイルサイズおよび通信イベント数は各々100 KB および 384 個であった．次に，5.1 節と同様に，機能 F3 を用いて性能改善の上限を調べた（表 1 の T_2）．

```
1: for (k=1; k ≤ log P; k++) {
2:     PRGN_BEGIN;
3:     Local calculation for image splitting;
4:     MPI_Sendrecv();
5:     Local calculation for image compositing;
6:     PRGN_END;
7: }
```

図 8 画像合成プログラムにおける並列領域の指定
Fig. 8 Image compositing program and its parallel region.

T_2 より，待ち時間を短縮するとともに，計算時間および通信量を均等にしたときに最短の実行時間 $T_2 = 11.0$ ミリ秒を実現できる可能性がある．また，仮定 A1〜A3 を順に追加することにより，T_2 をおよそ 5 ミリ秒ずつ短縮できていることから（22.3，16.5，11.0 ミリ秒），すべての改善手法が性能改善に貢献している．ここで，機能 F3 が DAG を再構築しない単純な予測であり，計算時間および通信量の偏りに起因する待ち時間を削減できないことに注意されたい．仮定 A2 および A2 ∧ A3 において $T_2 > T_M$ であることから，機能 F3 では仮定の組合せ全体から改善効果の大きい仮定を特定する必要がある．

このプログラムでは各合成ステージにおいて同期が発生することから，開発者は計算負荷および通信量を均等にすることで，待ち時間を短縮できると考えた．そこで機能 F2 を用いて，k ($1 \le k \le 6$) 番目の並列ステップ S_k における計算時間および通信量を均等にしたとき（仮定 A4(k)）の実行時間 T_3 を予測した（**表 3**）．表 3 より，(1) A4(1)〜A4(6) を実現することで，T_3 を 11.0 ミリ秒に短縮でき，(2) 特に S_2 において改善効果が大きいことが確認できる（21.6 ミリ秒）．

まず (1) について述べる．11.0 ミリ秒という値は，

表 3 画像合成プログラムに対する性能予測（機能 F2）
Table 3 Performance prediction for image compositing program (Function F2).

並列ステップ S_k に対する仮定 A4(k)：S_k における計算時間および通信量が均等	予測した実行時間 T_3（ミリ秒）
—	30.6
A4(1)	29.1
A4(2)	21.6
A4(3)	26.4
A4(4)	27.2
A4(5)	29.6
A4(6)	29.9
A4(1) ∧ A4(2) ∧ ··· ∧ A4(6)	11.0

表 1 の A1 ∧ A2 ∧ A3 における T_2 の値と一致する．つまり，開発者が考えたとおり，仮定 A4(1)〜A4(6) を実現することで，依存関係を考慮することなくすべての待ち時間を 0 秒と仮定したときと同等の改善が期待できる．そこで，仮定 A4(1)〜A4(6) を実現するために，各プロセスが担当する不透明領域を均等化できる手法を開発した（BSLBR 法[20]）．具体的には，部分画像をブロック状に分割することを避けインタリーブ状に分割することで，$T_M = 14.8$ ミリ秒を得た．

次に（2）に対しては，各合成ステージにおける画像の透明領域を調べた結果，透明領域を排除する工夫が不十分であり，S_2 においてのみ計算負荷および通信量の偏りを増大させることを確認できた．そこで，透明領域をさらに細かく排除できる手法を開発することで S_2 における偏りを抑制でき，$T_M = 11.8$ ミリ秒を得た（BSLMBR 法[20]）．

5.3 予測結果に対する LogGPS モデルの効果

本節では，PerWiz のための並列計算モデルとして LogGPS モデルを用いた理由および予測結果に対する各パラメータの効果について述べる．

PerWiz のための並列計算モデルが満たすべき条件は，以下の 2 つである．

R1：予測精度に関する条件 モデルがプログラムの性能ボトルネックを再現できること．

R2：実用性に関する条件 モデルが複雑すぎず，予測のための処理時間が十分短いこと．

PC クラスタにおける MPI プログラムの実行時間を予測するために，我々は以下の理由から LogGPS モデルを選択した．まず，LogGPS モデルは MPI などの高水準通信ライブラリを対象としているため，同期通信における送信側の待ち時間 t_{wait} を明確に定義していること（R1）．次に，LogGPS モデルは LogP および LogGP モデルと同様，イベント 1 個あたりの予測のための計算量が O(1) 時間であること（R2）．

表 4 LogGPS モデルによる性能予測（画像合成プログラム）
Table 4 Performance prediction by LogGPS for image compositing program.

並列領域に対する仮定 A1：待ち時間が 0 秒 A2：計算時間が均等 A3：通信量が均等	予測した実行時間 イーサネット T_3（ミリ秒）	 ミリネット T_4（ミリ秒）
—	178.8	29.8
A1	171.1	22.2
A2	181.1	34.1
A3	138.5	25.6
A1 ∧ A2	165.6	16.4
A1 ∧ A3	98.0	17.0
A2 ∧ A3	220.6	37.5
A1 ∧ A2 ∧ A3	93.4	11.3

表 5 LogGPS モデルにおける送信時間および受信時間
Table 5 Send and receive time under LogGPS.

条件	時間	イーサネット（マイクロ秒）	ミリネット（マイクロ秒）
非同期	t_{send}	$12.1 + 0.0901 \cdot k$	$2.15 + 0.00188 \cdot k$
$k \leq S$	t_{recv}	$12.1 + 0.0568 \cdot k$	$2.15 + 0.000937 \cdot k$
同期	t_{send}	$175 + 0.0859 \cdot k$	$28.9 + 0.00358 \cdot k$
$k > S$	t_{recv}	$264 + 0.0900 \cdot k$	$90.3 + 0.00485 \cdot k$

k: message length; S: 127,999 and 16,383 bytes for Ethernet and Myrinet, respectively.

なお，本稿では LogGPS モデルを用いたが，バス型のネットワークなどネットワーク競合が性能ボトルネックとなる計算環境に対しては，競合をモデル化する LoPC[21] や LoGPC モデル[22] が R1 のために必要である．計算機が広域に分散するグリッドに対しても同様に並列計算モデルを選択する必要がある．

次に，予測結果に対する各パラメータの効果について調べるために，5.2 節の $\mathcal{L}$ を基に，イーサネットおよび MPICH を用いたときの実行時間 T_3 を予測した（**表 4**）．ここで，予測対象および $\mathcal{L}$ を生成した実行環境が異なるため，t_{send} および t_{recv} は LogGPS モデルの定義に基づいた（**表 5**）．同様に，ミリネットに対しても再び予測を行った（表 4 の T_4）．

表 1 と同様，ミリネットでは仮定 A1〜A3 がおよそ 5 ミリ秒ずつ T_4 を短縮しているのに対し，イーサネットでは仮定 A1 ∧ A3 において最短の $T_3 = 93.4$ に近い 98.0 ミリ秒を得ている．すなわち，イーサネットでは o，G および L の値が大きく，メッセージ転送のための処理が性能ボトルネックとなり，計算時間を均等にすることの効果が小さい．一方，o，G および L の値が小さいミリネットでは，その効果が相対的に大きい．なお，イーサネットにおいて実測の実行時間 T_M は 175.3 から 96.6 ミリ秒に短縮していて，T_3 との誤差は 4%以下である．

適用例で用いたプログラムはともにメッセージ長 k の平均が 80 KB を超えるため，k に比例する o および G が予測結果に対して影響が大きい．一方，k の値が小さいプログラムに対しては，S が大きく影響しうる．特に，o および G の値が小さいミリネットでは，t_{send} および t_{recv} よりも t_{wait} が性能ボトルネックになりやすく，S の値を大きくすることにより実行時間が半減することもある[10]．

5.4 大規模プログラムに対する適用の可能性

PerWiz は実行履歴に対して解析を行っている．ゆえに，膨大な量のイベントを生成する大規模プログラムに対しては，以下の 2 つの制限が問題となり，PerWiz の適用が難しい場合がある．

- 実行履歴を生成できない（ファイルサイズの制限）
- 実行履歴を生成できるが解析のための時間が長い

これらの制限は，記録するイベントの数に起因するため，イベント数をうまく削減することが大規模プログラムに対して重要である．このための工夫としては，プログラムの実行時に解析を行う方法[2] に加え，以下の 2 つの方法があげられる．

性能ボトルネックに対する解析 あらかじめプロファイラを用いて性能ボトルネックとなる処理を特定し，記録対象を性能ボトルネックに絞り込む．

記録の省略 同一の処理を繰り返し行う部分に対しては，その 1 回の実行のみを記録対象とする．

なお，適用例のうち位置合わせプログラムに対しては，上記の工夫を用いた．あらかじめプロファイラ gprof を用い，性能ボトルネックが類似度を表す関数の微分処理にあることを特定し，さらに 4 回の微分処理のうち 1 回のみを解析対象とした．その結果，イベント数を 28,858 個から 4,634 個に削減できた．

6. ま と め

本稿では，メッセージ通信プログラムの改善可能性を評価する手法について述べ，その手法に基づく PerWiz を提案した．PerWiz は，LogGPS モデルに基づいてプログラムの実行履歴を再構築することで，改善効果の大きい通信および計算をドミノパスおよび並列ステップとして指摘できる．また，特定の仮定のもとで修正後プログラムの実行時間を予測でき，このときの実行の様子を logviewer で視認できる．

PerWiz を用いて MPI プログラムを修正した結果，ドミノパスに着目した手法は，待ち時間に着目した手法よりも効率良くプログラムの性能を改善できた．ゆえに，プログラムを修正するための労力およびその改善効果を検討する際に PerWiz は有用であると考える．

謝辞 本研究の一部は，日本学術振興会未来開拓学術研究推進事業（JSPS-RFTF99I00903），科学研究費補助金基盤研究（C）（2）（14580374）および NEC ネットワークス開発研究所の補助による．また，有益な御意見をいただいた査読者の方々に感謝いたします．

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) 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).
3) Eom, H. and Hollingsworth, J.K.: A Tool to Help Tune where Computation Is Performed, *IEEE Trans. Softw. Eng.*, Vol.27, No.7, pp.618–629 (2001).
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) Heath, M.T. and Etheridge, J.A.: Visualizing the Performance of Parallel Programs, *IEEE Software*, Vol.8, No.5, pp.29–39 (1991).
6) Shaffer, E., Reed, D.A., Whitmore, S. and Schaeffer, B.: Virtue: Performance Visualization of Parallel and Distributed Applications, *IEEE Computer*, Vol.32, No.12, pp.44–51 (1999).
7) 大澤範高，弓場敏嗣：並列プログラムの性能デバッギングを支援するアニメーション化ツール：かのこ，情報処理学会論文誌，Vol.39，No.11，pp.3111–3121 (1998).
8) Haynes, R., Crossno, P. and Russell, E.: A Visualization Tool for Analyzing Cluster Performance Data, *Proc. 3rd IEEE Int'l Conf. Cluster Computing* (*CLUSTER'01*), pp.295–302 (2001).
9) 伊野文彦，杉野陽一，山崎良太，藤本典幸，萩原兼一：並列プログラムの性能改善支援機能を持つ性能解析システム：Gordini，情報処理学会論文誌，Vol.41, No.5, pp.1577–1586 (2000).
10) 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).
11) Herrarte, V. and Lusk, E.: Studying Parallel

Program Behavior with Upshot, Technical ReportANL–91/15, Argonne National Laboratory (1991).
12) Lamport, L.: Time, Clocks, and the Ordering of Events in a Distributed System, *Comm. ACM*, Vol.21, No.7, pp.558–565 (1978).
13) Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K.E., Santos, E., Subramonian, R. and von Eicken, T.: LogP: Towards a Realistic Model of Parallel Computation, *Proc. 4th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming* (*PPoPP'93*), pp.1–12 (1993).
14) Alexandrov, A., Ionescu, M., Schauser, K. and Scheiman, C.: LogGP: Incorporating Long Messages into the LogP Model for Parallel Computation, *J. Parallel and Distributed Computing*, Vol.44, No.1, pp.71–79 (1997).
15) Gropp, W., Lusk, E., Doss, N. and Skjellum, A.: A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard, *Parallel Computing*, Vol.22, No.6, pp.789–828 (1996). http://www.mcs.anl.gov/mpi/mpich/
16) 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). http://www.myri.com/
17) 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). http://www.pccluster.org/
18) Schnabel, J.A., Rueckert, D., Quist, M., Blackall, J.M., Castellano-Smith, A.D., Hartkens, T., Penney, G.P., Hall, W.A., Liu, H., Truwit, C.L., Gerritsen, F.A., Hill, D.L.G. and Hawkes, D.J.: 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).
19) Ma, K.-L., Painter, J.S., Hansen, C.D. and Krogh, M.F.: Parallel Volume Rendering Using Binary-Swap Compositing, *IEEE Computer Graphics and Applications*, Vol.14, No.4, pp.59–68 (1994).
20) Takeuchi, A., Ino, F. and Hagihara, K.: An Improvement on Binary-Swap Compositing for Sort-Last Parallel Rendering, *Proc. 18th ACM Symp. Applied Computing* (*SAC'03*), pp.996–1002 (2003).
21) Frank, M.I., Agarwal, A. and Vernon, M.K.: LoPC: Modeling Contention in Parallel Algorithms, *Proc. 6th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming* (*PPoPP'97*), pp.276–287 (1997).
22) Moritz, C.A. and Frank, M.I.: LoGPC: Modeling Network Contention in Message-Passing Programs, *IEEE Trans. Parallel and Distributed Systems*, Vol.12, No.4, pp.404–415 (2001).

(平成 15 年 1 月 28 日受付)
(平成 15 年 5 月 4 日採録)

神戸 友樹
平成 14 年大阪大学基礎工学部情報科学科卒業．現在，同大学院情報科学研究科修士課程在学中．並列ソフトウェア全般に興味を持つ．

置田 真生
平成 15 年大阪大学大学院基礎工学研究科修士課程修了．現在，同大学院情報科学研究科博士課程在学中．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 12 年大阪大学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．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 4〜5 年文部省在外研究員（米国メリーランド大学）．現在，並列処理の基礎および応用に興味を持っている．