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

伊野文彦[†] 杉野陽一[††] 山崎良太[†]
藤本典幸[†] 萩原兼一[†]

メッセージ交換ライブラリを用いた並列プログラムの性能改善を支援するため，プログラムの実行状況を記録したトレースデータを可視化する性能解析システムが利用されている．既存の性能解析システムでは，着目すべき箇所をユーザが可視化結果から探し出し，その原因を特定する必要があるため，ユーザの負担が大きい．ユーザによっては，その原因を特定できず，性能改善に結び付かないこともある．また，探し出した性能ボトルネックをどのようにすれば解消できるのか分からないこともある．そこで，我々は通信オーバヘッドが並列プログラムの性能ボトルネックになりやすいことに着目し，通信に関する性能改善手法が適用できる可能性のある箇所を指摘できる性能解析システム Gordini を開発した．Gordini は，トレースデータを入力として，改善可能性のある箇所をソースコード上で指摘できる．本論文では，Gordini の検索できる性能改善手法とその性能改善支援機能について説明し，並列ソフトウェアコンテストで優勝したプログラムに適用した例を示す．プログラムの改善結果から，Gordini は並列プログラムの性能改善に役立つことが示された．

A Performance Analysis System with Performance Improvement Aid Functions for Parallel Programs: Gordini

Fumihiko Ino,[†] Youichi Sugino,[††] Ryouta Yamasaki,[†]
Noriyuki Fujimoto[†] and Kenichi Hagihara[†]

Performance analysis systems have been used for improving performance of parallel message-passing programs. An existing system visualizes performance information from trace data, which is generated by running an instrumented program. In general, a parallel program operates for a great many data, and the visualized information about its execution becomes enormous. Therefore, it is not easy for most users to find its performance bottlenecks from the complicated visualizations. Furthermore, such a system gives no appropriate advice on the performance problems. In this paper, we propose a new kind of performance analysis system Gordini. Besides traditional functions, it has performance improvement aid functions, which can point out the parts of source codes where techniques for improving performace may be applied. According to Gordini's direction, we could improve the performance of programs that won the championships at parallel program contests. The result of our work shows that Gordini is useful to improve performance of parallel programs.

1. はじめに

ワークステーションクラスタや分散メモリ型並列計算機の普及とともに，MPI[1] や PVM[2] などのメッセージ交換ライブラリを利用した並列計算が普及してきている．これらの並列計算方法では，通信および同期に関するライブラリを C や Fortran などのプログラムで用い，この逐次プログラムを複数プロセッサで実行させることで並列計算を行う．以降では，このようなプログラムを並列プログラムと呼ぶ．プログラマは，プロセッサ数を考慮してデータや計算をどのように分散させ，どこで通信を行うかをプログラム中に明示的に記述する必要がある．プログラムの詳細な動作を記述できるため，プログラマの工夫次第で性能の良いプログラムを得ることができる．

一般に，逐次プログラムと比較して，性能の良い並列プログラムを作成することは難しい．なぜなら，データや計算を適切に分散しなければ，特定のプロセッサに負荷が集中したり，プログラム実行中の通信回数が多くなったりして，並列実行による高速化が得られないからである．したがって，並列プログラムの性能を悪くしている箇所（性能ボトルネック）を探し，その

† 大阪大学大学院基礎工学研究科情報数理系専攻
Department of Informatics and Mathematical Science, Graduate School of Engineering Science, Osaka University
†† 沖電気工業株式会社情報通信ネットワーク事業部
Corporate Information Network Systems Division, Oki Electric Industry Co., Ltd.

性能を改善することは重要である．

並列プログラムの性能改善を支援するために，性能解析システムに関する研究が行われている[3]．多くの性能解析システムは，並列プログラムの実行状態を記録したトレースデータをもとに性能解析を行っている．Jumpshot[4] や VAMPIR[5] は，プログラムを実行することであらかじめトレースデータを生成し解析を行う（post-mortem analysis）．いずれも各プロセッサの状態と通信の様子を時間軸に沿って表示（Timeline view）したり，通信に関する統計を可視化したりすることができる．Pablo[6] は，解析したいソースコードの行を GUI を用いて容易に選ぶことができ，行ごとの呼び出し回数や実行時間をソースコード上に表示（Source view）できる．かのこ[7] は，力学系モデルを利用したアニメーションや音声を用いて，性能ボトルネックの発見を支援する．一方，プログラムの実行中に，生成するトレースデータを選択しながら解析できる（dynamic analysis）システムがある．Paradyn[8] は，知識ベースをもとに，性能ボトルネックとなっている箇所とその原因を検索できる．

しかし，既存の性能解析システムはプログラムの性能改善を十分に支援しているとはいえない．Paradyn を除くシステムを用いてプログラムを改善する場合，改善すべき箇所をユーザが可視化結果から探し出し，性能ボトルネックの原因を特定する必要がある．たとえば，プログラムの実行時間の大半を特定の通信命令が占めていることに気づいたとしても，その原因が通信待ちにあるのか，あるいは通信の繰返しによる蓄積なのかは，複数の可視化結果をもとにユーザが考察しなければならない．したがって，ユーザによっては，その原因を特定できず，性能を改善できないことがある．また，大規模プログラムは可視化結果が複雑になる．複雑な可視化結果から人間の視覚で性能ボトルネックを探すことは容易ではない．さらに，性能ボトルネックを見落としてしまう可能性もある．一方，Paradyn は性能ボトルネックとなっている箇所とその原因を検索できるが，その改善方法はユーザが考察する必要がある．また，モジュール，関数，通信命令という具合に，可能な限り検索結果を徐々に狭めていく検索方法であるため，必ずしも性能ボトルネックとなっている箇所を直接指摘できるわけではない．したがって，さらに詳細な解析を行わなければプログラムを修正できないことがある．これらの問題に対処するためには，システムは以下の 2 点を満たす必要がある．

（R1）性能改善方法を示すこと

（R2）性能ボトルネックを検索できること

性能ボトルネックをトレースデータから探す作業と改善方法を決定する作業をシステムが担うことで，ユーザはプログラムの修正作業に専念できる．

そこで，我々は通信オーバヘッドが並列プログラムの性能ボトルネックになりやすいことに着目し，トレースデータを入力として，通信に関する性能改善手法が適用できる可能性のある箇所をソースコード上で指摘できる性能解析システム Gordini を開発した[9]．現時点で Gordini が考慮している性能改善手法は，通信と計算のオーバラップ[10),11)]，通信の集合化（aggregation）[12]，集合通信の利用[12] である．

本論文では，これらの性能改善手法と Gordini の性能改善支援機能について説明し，並列ソフトウェアコンテスト[13),14)] で優勝したプログラム[15),16)] に適用した例を示す．プログラムの性能を改善できたことから，提案する性能改善支援機能は有用であると考える．

2. 性能改善手法

本章では，Gordini が考慮している 3 つの性能改善手法について説明する．

2.1 通信と計算のオーバラップ

通信と計算のオーバラップとは，通信を行っている時間を利用して計算を行うことである．メッセージがネットワーク上を流れている時間を計算処理に割り当てることでプログラムの実行時間を短縮することができる．通信と計算のオーバラップを効率良く実現するための機構として，ノンブロッキング通信[1] がある．ノンブロッキング通信は，通信処理を起動するがその処理の完了を待たずに，次の命令をすぐに実行できる．通信処理を完了させるためには，`MPI_Wait()` などの通信完了命令を呼び出す必要がある．ノンブロッキング通信命令の呼び出しから通信完了命令の呼び出しまでの間に計算を並行して行うことで，通信と計算を効率良くオーバラップできる．ただし，通信と計算をオーバラップできるかどうかは，実行環境のハードウェアに依存する[1]．また，通信と計算をオーバラップさせるとき，以下の条件を満たす必要がある．

- 送信命令とオーバラップする計算命令は，送信バッファの内容の変更を行わない．
- 受信命令とオーバラップする計算命令は，受信バッファの内容の参照，変更を行わない．

ここで，送信バッファとは，送信命令がメッセージの送信時に参照するメモリ領域を，受信バッファとは，受信命令によって受け取ったメッセージが格納されるメモリ領域を意味する．上記の条件を満たさなければ，メッセージの送信を行う前に送信バッファの内容を書

き換えたり，メッセージが到着する前の受信バッファの内容を参照したりする可能性があるため，一般にはプログラムが正しく動かない．

2.2 通信の集合化

通信の集合化とは，送り先が等しい複数の通信を 1 つの通信にまとめることである．一般に，通信量の少ない通信を繰り返すよりも，通信量の多い通信を 1 回行う方が効率良く通信できるため，プログラムの性能向上が期待できる．

具体的には，連続したメモリ領域の送信を 1 つの送信にまとめる（**図 1**）．送信するデータがメモリ上で連続していない場合は，あらかじめ連続したメモリ領域に格納することで，1 つの送信にまとめることができる（**図 2**）．このとき，連続メモリ領域へのコピー処理が必要だが，そのための処理時間よりも通信のオーバヘッドの方が大きい場合，プログラムの性能向上は期待できる．

なお，集合通信どうしをまとめることも可能である．集合通信については 2.3 節で述べる．

2.3 集合通信の利用

集合通信[1] とは，指定されたプロセッサ集合に属する全プロセッサが同一の集合通信関数を呼び出し，一対多，多対一，あるいは多対多の通信を行うことである．並列プログラムの実行環境によっては，一対一通信を各プロセッサで繰り返し実行するよりも集合通信を一度実行した方がより高速に処理できる場合がある．ただし，全プロセッサが集合通信を実行する必要があるので，集合通信関数の呼び出しが遅れるプロセッサが存在すると，他のプロセッサはそのプロセッサが関数呼び出しを行うまで待つ必要がある．したがって，プログラムによっては，プロセッサの同期のための待ち時間が実行時間を逆に長くすることもある．

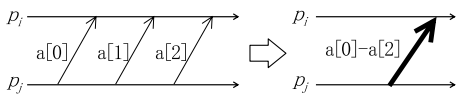

図 1 メッセージの集合化（メモリコピーなし）
Fig. 1 Aggregation of messages without copy.

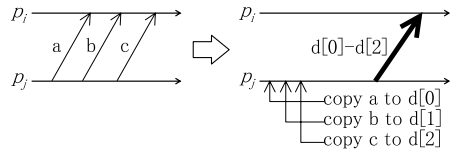

図 2 メッセージの集合化（メモリコピーあり）
Fig. 2 Aggregation of messages with copy.

3. Gordini の概要

Gordini は C++言語と Tcl/Tk 言語が利用できる環境で動作する．解析対象とする並列プログラムは MPI を利用した C 言語プログラムである．ただし，後述のトレースデータ生成関数を Fortran に移植すれば，Fortran 言語プログラムも解析対象にできる．**図 3** に Gordini を用いてプログラムの性能改善を行う際の作業の流れを示す．まず，ユーザはトレースデータを生成するための関数呼び出しを解析対象の並列プログラムに追加する．ユーザが直接ソースコードを編集することで関数呼び出しを追加することもできるが，Gordini ではそれらを自動的に追加する変換ツール mpi2g が利用できる．変換ツール mpi2g は，パターンマッチに基づき，ソースコードに含まれるすべての通信命令と代入文の直後にトレースデータ生成関数の呼び出しを追加する．ただし，パターンマッチに基づいているため，任意のソースコードに対応できているわけではない．次に，関数呼び出しを追加したプログラム（トレースデータ生成用プログラム）をトレースデータ生成ライブラリとリンクし，並列計算機上で実行することで，プロセッサごとのトレースデータを生成する．生成されたトレースデータをもとに，提案する性能改善支援機能（以下，提案機能）や既存の性能解析機能を用いて，性能改善の手がかりとなる情報を得る．その情報をもとに，ユーザはプログラムが修正可能かどうかを判断し，可能であれば修正を行う．

Gordini の入力となるトレースデータとは，各プロセッサで発生したイベントを時系列順に記録したものであり，プロセッサごとに 1 つずつ存在する．Gordini

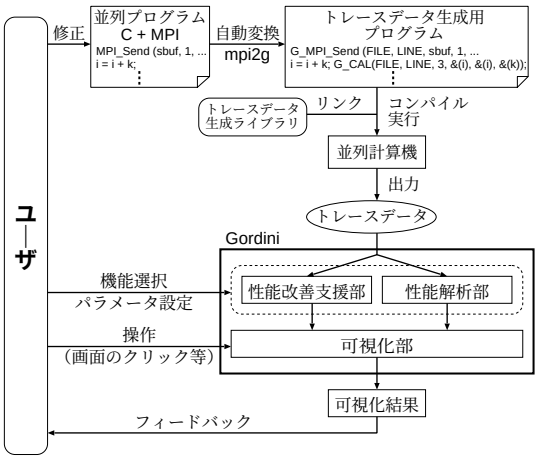

図 3 Gordini を用いた作業の流れ
Fig. 3 Process of performance debugging with Gordini.

表 1 イベント一覧
Table 1 List of events.

イベントの種類	記録する情報	
	共通情報	固有情報
送信	イベント番号 ソースコード行 発生時刻	送信相手，送信バッファのアドレスとサイズ
受信		受信相手，受信バッファのアドレスとサイズ
計算		参照・代入が行われた変数のアドレス
ユーザ定義		ユーザが自由に定義

表 2 Gordini の機能
Table 2 Functions of Gordini.

性能改善支援部	通信と計算のオーバラップ支援 通信集合化支援 集合通信の利用支援
性能解析部	ホットスポット解析 通信状況解析 プロセッサ使用状況解析
可視化部	棒グラフ表示 円グラフ表示 Timeline view 表示 ソースコード表示

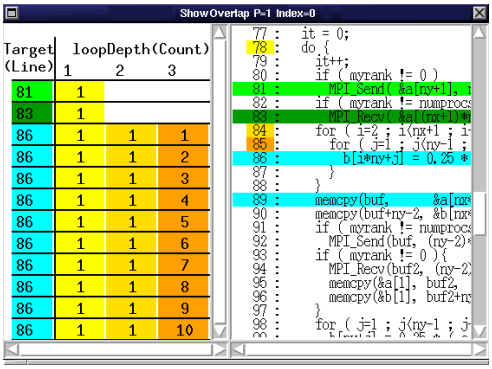

図 4 Gordini のスナップショット
Fig. 4 Snapshot of Gordini.

が着目するイベントとトレースデータに記録する情報を**表 1** に示す．送信イベントと受信イベントは，それぞれ送信命令と受信命令の実行に対応し，計算イベントはプログラム中の代入命令とメモリコピー命令（`memcpy()` など）の実行に対応する．ユーザ定義イベントという，イベントの発生条件とそのイベントが発生したときに記録する情報をユーザが定義できるものもある．トレースデータに記録する情報は，すべてのイベントに共通する情報とイベント固有の情報に分類できる．これらの情報をもとに，Gordini は性能改善手法が適用できる可能性のある箇所を探す．

Gordini は，性能改善支援部，性能解析部，可視化部を持つ（図 3）．各部の機能を**表 2** に示す．性能解析部を用いた結果は，棒グラフや円グラフ，Timeline view で示し，性能改善支援部の結果はソースコード上に示す．後者の表示方法は重要である．ソースコード上に結果が示されなければ，ユーザが該当箇所をソースコード上で探さなければならない．たとえば，時間軸上で通信が頻繁に生じている箇所を特定できたとしても，その箇所に対応する通信命令をソースコード上で特定できなければ，プログラムは修正できない．

図 4 に提案機能の可視化例を示す．画面右側は，81 行および 83 行の通信命令が 86 行と 89 行の計算命令とオーバラップできることを示している．画面左側は，より詳細な結果を示すものであり，イベント単位でオーバラップの可能性を示している．ループの中に命令があるときなど，1 命令が複数のイベントに対応する場合，その一部分のイベントとだけオーバラップできることがあるため，イベント単位の可視化も用意している．図 4 では，3 重ループ（78，84，85 行）のうち，外側のループ（78 行）の 1 回目の実行で記録されたイベントを可視化していて，通信イベント（81 行，83 行）とオーバラップできる計算イベントとして，中間のループ（84 行）の 1 回目の実行で記録され，かつ，内側のループ（85 行）の 1〜10 回目の実行で記録された計算イベントを指摘している．この場合，内側のループを 2 つのループ（10 回目までを実行するループと 11 回目以降を実行するループ）に分割することで，オーバラップを実現できる．

なお，トレースデータはプロセッサごとに存在するため，可視化結果もプロセッサの数だけ生成する．マスタ・スレーブ型のプログラムなどでは，あるプロセッサにおける指摘箇所が他プロセッサには見当らないことがある．この場合，指摘のあるプロセッサとそうでないプロセッサで条件分岐する命令をプログラムに追加し，指摘のある方だけ指摘箇所を修正すればよい．

4. 性能改善支援機能

本章では，提案機能のうち，通信と計算のオーバラップ支援機能（以下，オーバラップ支援機能）の検索アルゴリズムについて述べる．紙面の都合上，残りの提案機能については文献 9) を参照していただきたい．

2 章で述べた性能改善手法はプログラム中の命令の実行順序を変更するため，命令間の依存関係を考慮してプログラムを修正しなければならない．依存関係を無視して通信を集合化したり，通信と計算をオーバラップさせたりすると，プログラムが意図どおりに動作しない可能性がある．したがって，トレースデータから性能改善手法が適用できる箇所を求めるには，イベント間の依存関係を考慮する必要がある．次に示す 3 つの場合には，命令 a を実行しなければ命令 b が実行できない．この実行順序に関する制約を依存関係と呼び，以降では「$a \rightarrow b$」と書く．

- a で代入した変数のアドレスと b で代入した変数のアドレスが等しい（出力依存）．
- a で代入した変数のアドレスと b で参照した変数のアドレスが等しい（フロー依存）．
- a で参照した変数のアドレスと b で代入した変数のアドレスが等しい（逆依存）．

ここで，a もしくは b が通信命令のとき，代入と参照はそれぞれ受信と送信と読み換える．

トレースデータはプロセッサごとに存在するが，本論文の提案機能に関しては，複数のトレースデータを参照する必要はなく，各トレースデータを独立に検索するだけでよい．

4.1 通信と計算のオーバラップ支援機能

この機能は，トレースデータ中の各通信イベントに対して，各々の通信イベントとオーバラップ可能な計算イベントの集合を求め，結果を可視化する（図 4）．

以下，任意のプロセッサのトレースデータ T_j ($1 < j < N$, N：イベント数) が与えられて，通信イベント com_i ($i = 1, 2, \cdots$) とオーバラップできる計算イベント集合 Cal_i の組を得るアルゴリズムを示す（**図 5**）．おおまかには，実行時刻の遅い方の通信イベントから検索を行い，依存関係を持つ計算イベントが現れるまでの計算イベントをオーバラップ可能とする．

現在のオーバラップ支援機能は，通信イベントを起点としてオーバラップできる計算イベントを探すことにしている．一方，計算イベントを起点として通信イベントを探す方法も考えられる．しかし，計算イベントを起点にした場合，オーバラップ可能性があると指摘できるイベントの数は，通信イベントを起点にする方法よりも少なくなる可能性がある．たとえば，同一の作業用変数を繰り返し使うプログラムでは，作業用変数に代入するたびにそれらの計算イベント間に依存関係が生じるため，本来オーバラップできる計算イベントが指摘できなくなる可能性がある．

検索の様子を**図 6** に示す．図 6 では，プロセッサ $p2$ に着目し，各イベントは以下の条件を満たすものとする．

- $T_s \rightarrow com_i$ かつ $com_i \rightarrow T_e$ である．
- 任意の k ($s < k < j$) について $T_k \rightarrow com_i$ でない．
- 任意の k ($j < k < e$) について $com_i \rightarrow T_k$ でない．

このとき通信イベント com_i とオーバラップ可能な計算イベント集合として $\{T_{s+1}, T_{s+2}, \ldots, T_{e-1}\}$ が指摘できる．

オーバラップ支援機能は 1 章で述べた（R1）と（R2）を同時に満たす．つまり，具体的な性能改善手法に基づいた検索を行い，ユーザに有益な情報（オーバラップ可能性のある命令）だけを提示する．

```
入力：T[1..N]    // トレースデータ (N:イベント数)
出力：com[1..n]  // 通信イベント com[i](n:通信イベント数)
      Cal[1..n]  // com[i] とオーバラップ可能な
                 // 計算イベント集合 Cal[i]
int i = 1;
for (j=N; j>0; j--) {
  if (T[j] は通信イベントでない) continue;
  com[i] = T[j];
  for (e=j+1; e<=N; e++) {        // as late as
    if (com[i] が T[e] に依存あり) break;
    Cal[i] に T[e] を追加
  }
  for (s=j-1; s>0; s--) {         // as soon as
    if (T[s] が com[i] に依存あり) break;
    Cal[i] に T[s] を追加
  }
  i++;
}
```

図 5 検索アルゴリズム（オーバラップ支援機能）
Fig. 5 Algorithm to find overlappable calculation events.

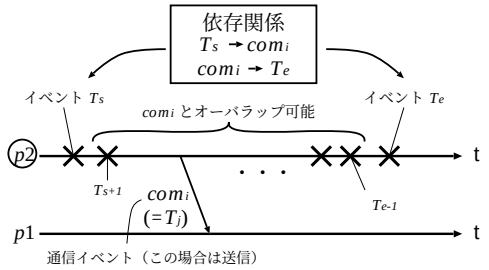

図 6 オーバラップ可能な計算を求める処理
Fig. 6 Process of finding overlappable calculation events.

5. 評　　価

本章では，提案する性能改善支援機能が有用であることを確認するため，Gordini を用いた性能解析を行い，その性能改善例を示す．評価に用いた並列プログラムは，並列ソフトウェアコンテスト

（PSC95[13]，PSC96[14]）の Cenju-3 部門で優勝したプログラム[15),16)]である．並列プログラムの実行には分散メモリ型並列計算機 NEC Cenju-3，Cenju-4 を用い，トレースデータの生成にはワークステーション・クラスタを用いた．Gordini の提案機能が必要とするのは命令の実行順序であり，命令の実行時間そのものは必要としないので性能改善に必要となる部分は実行環境に依存しない．したがって，我々にとって利用しやすいワークステーション・クラスタを用いた．ワークステーション・クラスタの構成は，ノードプロセッサが AT 互換機（PentiumII 450 MHz），ネットワークが Myrinet[17)] である．なお，使用したメッセージ交換ライブラリはすべて MPICH[18)] である．

5.1 通信と計算のオーバラップによる改善例（PSC95）

対象としたプログラム[15)] は，ガウスの消去法を用いて連立方程式 $Ax = b$（A：$n \times n$ の密行列）を解くプログラムであり，ソースコードの規模は約 5000 行である．なお，コンテストでは Cenju-3 で 64 台のプロセッサを用い，問題サイズ $n = 1000$〜2000 で解いた．

まず，変換ツール mpi2g を用いて，もとのソースコードにトレースデータ生成関数の呼び出しを追加した．トレースデータを生成した対象は，前進消去と後退代入を処理する部分のすべての通信命令，代入文，メモリコピー命令であり，行列の初期化などの本質的でない部分は除外した．次に，問題サイズ n を 256，プロセッサ数 p を 4 として，ワークステーション・クラスタ上でプロセッサごとのトレースデータを生成した．4 つのトレースデータ全体で，約 120 万個のイベントが記録され，そのサイズは約 70 MBytes であった．

次に，提案機能を用いて改善可能性のある箇所を検索したところ，通信と計算のオーバラップおよび通信の集合化に関する箇所が指摘できた．指摘箇所の数（トレースデータ上の通信イベント数 T とソースコード上の通信命令数 S）およびその検索時間を**表 3** に示す．

最後に，指摘箇所が修正可能であるかどうかを確認し，プログラムを修正した．修正箇所はオーバラップ支援機能で指摘された 2 カ所である（付録 A.1 参照）．修正しなかった箇所は，プログラムの構造上，修正が容易でなかった箇所である．たとえば，集合化可能であると指摘された各々の通信が異なるソースファイルに存在し，かつループの中に含まれていた場合などがあげられる．また，指摘箇所の実行時間が短いため，改善の効果が期待できない箇所も修正していない．

表 3 指摘結果（PSC95，PSC96）
Table 3 Search results (PSC95, PSC96).

検索項目	PSC95 ($n = 256$; $p = 4$) 指摘数 T	S	検索時間	PSC96 ($m = 25{,}600$; $p = 4$) 指摘数 T	S	検索時間
オーバラップ	922	14	22 分	109	22	5 分
通信集合化	244	6	20 分	12	2	2 分
集合通信利用	0	0	20 分	0	0	1 分

T：トレースデータ，S：ソースコード

表 4 に，修正前の実行時間 t_1，修正後の実行時間 t_2，速度向上率 $\sigma = (t_1 - t_2)/t_1$ を示す．ここで，実行時間とは前進消去と後退代入の部分の実行時間を指す．なお，修正前のプログラムはコンテストの実施以降改良が加えられていて，Cenju-3 で動作しなかった（実行時エラー）ため，Cenju-4 の実行結果を掲載した．

問題サイズ n が 128 のときは改善効果が確認できないが，n の増加にともない速度向上率 σ が増し，n が 4096 のときは 50%弱の改善が確認できる．これは，t_1 に対して，オーバラップさせた計算命令と通信命令の実行時間の占める割合 ρ が n とともに大きくなっているためである．時刻を返す関数（`MPI_Wtime()`）を用いてオーバラップさせた計算命令と通信命令の実行時間を計測したところ，$p = 16$，$n = 256$ のとき ρ は 7.6%であるが，$p = 16$，$n = 4096$ のときは 99.5%であった．したがって，このプログラムは n が大きいほど，通信と計算のオーバラップの改善効果が出やすい性質を持っていたと考えられる．

5.2 通信の集合化による改善例（PSC96）

対象としたプログラム[16)] は複素離散フーリエ順変換・逆変換を行うプログラムであり，約 1000 行の規模である．コンテストでは Cenju-3 で 128 台のプロセッサを用い，データ数 $m = 3{,}276{,}800$ で解いた．

5.1 節と同様に，変換ツール mpi2g を用いて，もとのソースコードにトレースデータ生成関数の呼び出しを追加した．ただし，データの初期化などの本質的でない部分は除外した．次に，問題サイズ m を 25,600，プロセッサ数 p を 4 として，ワークステーション・クラスタ上でトレースデータを生成した．このプログラムに対しては，通信と計算のオーバラップおよび通信の集合化が指摘できた（表 3）．修正箇所は通信と計算のオーバラップに関して 8 カ所，通信の集合化に関してすべてである（付録 A.2 参照）．5.1 節と同様，残りの指摘箇所は実行時間が短いため，修正していない．なお，通信の集合化で指摘できた箇所は集合通信どうしを対象としたものであった．

表 5 に修正前の実行時間 t_1，修正後の実行時間 t_{ovl}，

表 4 Cenju-4 での実行時間（PSC95）
Table 4 Execution times on Cenju-4 (PSC95).

問題サイズ n	プロセッサ数 $p=4$ 修正前 t_1（秒）	修正後 t_2（秒）	速度向上率 σ（%）	プロセッサ数 $p=16$ 修正前 t_1（秒）	修正後 t_2（秒）	速度向上率 σ（%）
128	0.014	0.014	0.0	0.023	0.023	0.0
256	0.054	0.047	12.9	0.052	0.052	0.0
512	0.351	0.240	31.6	0.152	0.135	11.2
1024	3.367	1.918	43.0	0.846	0.577	31.8
2048	29.454	15.783	46.4	7.149	4.084	42.9
4096	234.418	121.515	48.2	60.114	31.748	47.2

表 5 Cenju-3 での実行時間（PSC96）
Table 5 Execution times on Cenju-3 (PSC96).

プロセッサ数 p	問題サイズ $m=819{,}200$ 修正前 t_1（秒）	t_{ovl}（秒）	修正後 t_{agg}（秒）	t_2（秒）	速度向上率 σ（%）	問題サイズ $m=3{,}276{,}800$ 修正前 t_1（秒）	t_{ovl}（秒）	修正後 t_{agg}（秒）	t_2（秒）	速度向上率 σ（%）
16	1.565	1.543	1.560	1.538	1.73	—	—	—	—	—
32	0.877	0.865	0.865	0.855	2.51	3.990	3.878	3.972	3.874	2.91
64	0.450	0.447	0.440	0.438	2.67	2.021	1.972	1.974	1.927	4.65
128	0.213	0.213	0.193	0.193	9.39	1.030	1.011	0.972	0.965	6.31

t_{ovl}：通信と計算のオーバラップのみ修正，t_{agg}：通信の集合化のみ修正，t_2：両方とも修正

t_{agg}，t_2，速度向上率 σ を示す．ここで，t_{ovl}，t_{agg}，t_2 はそれぞれ順に，通信と計算のオーバラップのみ修正したプログラムの実行時間，通信の集合化のみ修正したプログラムの実行時間，両方とも修正したプログラムの実行時間を表す．また，$p=16$，$m=3{,}276{,}800$ の欄が空白になっているが，プログラムの必要とするメモリ量が実メモリ量を超えたため，実行できなかった．

表 5 から，$p=128$，$m=819{,}200$ の場合，プログラムの性能改善に効果があったのは，集合通信の集合化にあったことが分かる．一方，$p=32$，$m=3{,}276{,}800$ の場合は，通信の集合化よりも通信と計算のオーバラップの方が効果的であることが確認できる．以下，通信と計算のオーバラップおよび通信の集合化による改善効果について考察する．集合通信はプロセッサを同期させるため，プロセッサ数が増えるほど，同期のための待ち時間が大きくなる傾向がある．したがって，p が大きいほど，集合通信を集合化することによる改善効果は大きい．また，問題サイズを固定しプロセッサ数を増やすと，1 プロセッサあたりの計算量が減るため，計算時間が全実行時間に占める割合は小さくなり，通信時間の占める割合が大きくなる．したがって，通信の集合化のように，通信が占める実行時間を削減する特徴を持つ改善手法の改善効果が大きくなる．一方，通信と計算のオーバラップは，一方の実行時間を他方で隠蔽する特徴を持つ．問題サイズを固定しプロセッサ数を減らすと，1 プロセッサあたりの計算量が増えるため，通信と計算のオーバラップによる改善効果が大きくなることが期待できる．$m=819{,}200$ の欄（表 5）の t_1 と t_{ovl} を見ると，$p=16$ では 0.022（秒）改善できているが，$p=128$ では改善効果が確認できない．

6. 考　　察

6.1 性能改善支援機能の有用性

提案機能は，通信に関する処理が並列プログラムの性能ボトルネックになりやすいことに着目している．したがって，性能をあまり考慮していないプログラムは，3 つの提案機能を用いて改善できる可能性が高い．しかし，ユーザが着目しやすい性能ボトルネックが一般的な並列プログラムに残されているのか，という疑問がある．これは，5 章の改善結果から，ユーザがすべての性能ボトルネックを必ずしも解決していないことが分かった．この原因は 1 章で述べたユーザの見落としにあると考えられる．さらに，実行時間や回数といった数値に着目した既存の性能解析機能では，修正すべき箇所はすべて修正した，という確信が持てないことも見落としの要因の 1 つである．一方，提案機能は考慮した性能改善手法に関してはすべての箇所を指摘できるため，ユーザの見落としを防ぐ効果が期待できる．

既存の性能解析機能で改善しにくい性能ボトルネックとして，通信と計算をオーバラップすることで改善

できる性能ボトルネックがあげられる．この改善手法は，通信命令と計算命令に着目し，依存関係を満たす範囲で，一方の実行時間を他方で隠蔽する，という特徴を持つ．既存機能は依存関係を考慮しないため，ユーザがオーバラップできる通信命令と計算命令をソースコードから探す必要がある．一方，Gordini のオーバラップ支援機能は，オーバラップ可能性のある箇所をソースコード上で示すため，ユーザは修正すべき箇所を探す必要がない．ただし，実際にオーバラップが可能かどうかはユーザが確認する必要がある．なぜ確認する必要があるのかについては後述する（6.3 節）．

また，ソースコードの命令ごとに満偏なく遅いプログラムがあった場合，既存機能では，性能ボトルネックを特定することが難しい．均等な数値が並んだ状況では，着目すべき箇所を見つけることができない．提案機能は，性能改善手法に基づく依存解析を行うことで性能ボトルネックを検索しているため，上記のようなプログラムにも対応できる可能性がある．

しかしながら，提案機能だけで性能ボトルネックを修正しても，修正した箇所の占める実行時間の割合が小さければ，改善効果が小さい．5.1 節では，指摘箇所の実行時間の占める割合 ρ が小さいとき（$\rho = 7.6$），通信と計算のオーバラップによる改善効果が確認できなかった．また，Gordini が考慮していない性能ボトルネックを持つプログラムに対しては，その性能ボトルネックを特定できない．したがって，性能改善手法に着目した提案機能と実行時間に着目した既存機能を併用することが良い．具体的には，提案機能を用いて性能ボトルネックの候補となるソースコードの行を洗い出し，既存機能でそれらの行の修正優先度をつける．実行時間の長い候補からプログラムを修正すれば，ユーザは効率良くプログラムを改善できる．

6.2 検索機能の重要性

Gordini では計算機が性能ボトルネックの検索を行う．計算機がユーザの代わりに検索を行うことで，膨大なトレースデータに対しても有益なデータだけを取り出すことができる．たとえば，5.1 節では約 120 万個のイベントを含むトレースデータ（約 70 MBytes）の中から 20 行（オーバラップ 14 行，通信集合化 6 行）の修正候補を取り出すことができた．また，我々の環境（メモリ容量 2 GBytes）では約 2200 万個のイベントを含むトレースデータ（約 2 GBytes）に対して検索できる．現在の Gordini はトレースデータ全体をメモリ上に展開するため，扱えるトレースデータのサイズはメモリ容量の制約を受ける．しかし，4.1 節で述べた検索アルゴリズムはトレースデータ全体を一度に必要としないので，実装を工夫すればトレースデータのサイズの制限はなくなる．

一般に，並列プログラムのトレースデータは問題サイズに加えてプロセッサ数にも比例して大きくなるため，1 章で述べた（R2）を満たすことは重要である．

6.3 トレースデータの生成意義

Gordini の機能を用いるためには，プログラムを実行しトレースデータを生成する必要がある．一方，プログラムの実行を必要としない方法もある．文献 11) では，ソースコードを入力として，代入文の位置を動かすことで通信と計算をオーバラップする手法を提案している．しかし，オーバラップの対象とするコードを代入文が連続する基本ブロックに限定しているため，オーバラップできる通信と計算は同一基本ブロック内にしか存在できない．互いに異なる基本ブロックに存在する通信と計算は，本来オーバラップできたとしても，この方法ではオーバラップできない．Gordini はトレースデータに対して検索を行うため，ソースコードの構造に関係なく，トレースデータ上の依存関係を満たす限り多くの計算と通信のオーバラップを指摘できる．一般に，通信命令は代入文よりも長い処理時間を必要とするので，多くの代入文をオーバラップ可能であると指摘できることは，改善効果の面で有利である．

ただし，現在の Gordini は 4 章で述べたデータ依存関係（出力依存，フロー依存，逆依存）のみを考慮していて，制御依存関係を考慮していない．したがって，Gordini の指摘した箇所が実際にオーバラップ可能かどうかはユーザが確認する必要がある．たとえば，ある通信 com とオーバラップできる計算に S1 という代入文が含まれていて，ソースコードが if S1 else S2 と記述されていた場合，S1 同様 S2 も com とオーバラップ可能であるとは限らない．S2 と com 間に依存関係があり，かつ，実行する度に if 文の分岐結果が変わるプログラムに対してはオーバラップしてはいけない．一方，文献 11) の方法では，検索範囲を基本ブロック内に限定しているので，必ずオーバラップできるソースコードが得られる．

6.4 プロセッサ間の時刻のずれ

トレースデータをもとにした性能解析システムでは，各プロセッサの持つ時計間の時刻のずれを考慮しなければならない．時刻のずれが引き起こす問題として，送信イベントの発生時刻より対応する受信イベントの終了時刻の方が早い，という矛盾があげられる．MPI には，プロセッサ間の時刻のずれを補正した時刻を返す関数（`MPI_Wtime()`）が用意されているが，ソース

コード上の 1 命令単位でイベントを記録するシステムでは補正精度が不十分である．提案機能は，イベントの発生時刻ではなく，1 プロセッサで生じるイベントの実行順序をもとにした検索を行うので，プロセッサ間の時刻のずれは問題にならない．プロセッサ間の時刻の補正精度が十分なものになれば，プロセッサ間のイベントの発生時刻を利用した検索もプログラムの性能改善に役立つと考えられる．たとえば，送信イベントと受信イベントの発生時刻をもとに待ちイベントを定義し，待ちイベントに着目した検索が行える．

7. ま と め

本論文では，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 について述べた．Gordini は，通信と計算のオーバラップ，通信の集合化，集合通信の利用の 3 つの性能改善手法に関する性能ボトルネックを指摘できる．Gordini を用いて，並列ソフトウェアコンテストで優勝したプログラムに対して性能解析を行った結果，その性能を改善することができた．改善結果から，提案する性能改善支援機能は有用であると考える．今後の課題としては，新たな性能改善手法への対応や性能改善手法を適用した場合の効果の予測，より大規模なプログラムへの対応があげられる．

謝辞 本研究は一部平成 11〜12 年度文部省科学研究費補助金・基盤研究（C)(11680357）および PDC（並列・分散処理研究推進機構）の補助による．並列計算機 Cenju を利用させていただいた日本電気（株）に感謝する．有益なご意見をいただいた査読者の方々に感謝する．

参 考 文 献

1) Snir, M., Otto, S.W., Huss-Lederman, S., Walker, D.W. and Dongarra, J.: *MPI: The Complete Reference*, The MIT Press (1996).
2) PVM: Parallel Virtual Machine. http://www.epm.ornl.gov/pvm/ (1999).
3) Browne, S., Dongarra, J. and London, K.: Review of Performance Analysis Tools for MPI Parallel Programs. http://www.cs.utk.edu/%7Ebrowne/perftools-review/review.html (1999).
4) Zaki, O., Lusk, E., Gropp, W. and Swider, D.: Toward Scalable Performance Visualization with Jumpshot, Technical Report ANL/MCS-P763-0699, ANL (1999).
5) VAMPIR: Visualization and Analysis of MPI Programs. http://www.pallas.de/pages/vampir.htm (1999).
6) Reed, D.A., Aydt, R.A., Noe, R.J., Roth, P.C., Shields, K.A., Schwartz, B. and Tavera, L.F.: Scalable Performance Analysis: The Pablo Performance Analysis Environment, *Proc. Scalable Parallel Libraries Conference*, pp.104–113, IEEE Computer Society (1993).
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) Miller, B.P., Callaghan, M.D., Cargille, J.M., Hollingsworth, J.K., Irvin, R.B., Karavanic, K.L., Kunchithapadam, K. and Newhall, T.: The Paradyn Parallel Performance Measurement Tools, *IEEE Computer*, Vol.28, No.11, pp.37–46 (1995).
9) 杉野陽一，伊野文彦，山崎良太，藤本典幸，萩原兼一：並列プログラムの改善支援機能を持つ性能解析システムの開発，情報処理学会研究報告，99-AL-66, pp.81–88 (1999).
10) Quinn, M.J. and Hatcher, P.J.: On the Utility of Communication - Computation Overlap in Data - Parallel Programs, *Journal of Parallel and Distributed Computing*, Vol.33, pp.197–204 (1996).
11) 渡邊誠也，湯淺太一：データ並列言語における通信最適化のためのコード移動方法，情報処理学会論文誌，Vol.40, No.3, pp.1257–1266 (1999).
12) Gropp, W. and Lusk, E.: Tuning MPI Applications for Peak Performance. http://www-unix.mcs.anl.gov/mpi/tutorial/perf/ (1999).
13) PSC95: http://www.info.waseda.ac.jp/muraoka/project/PSC95/ (1995).
14) PSC96: http://proton.is.s.u-tokyo.ac.jp/psc96/ (1996).
15) 建部修見：SOFTWARE. http://phase.etl.go.jp/%7Etatebe/software/index-j.html (1999).
16) PSC '96: http://www.ppc.nec.co.jp/all/touroku/psc96/index.html (1999).
17) Myrinet: Myricom. http://www.myri.com/ (1999).
18) MPICH: A Portable MPI Implementation. http://www.mcs.anl.gov/mpi/mpich/ (1999).
19) 小国 力，村田健郎，三好俊朗，Dongarra, J.，長谷川秀彦：行列計算ソフトウェア，丸善 (1991).
20) 建部修見：分散メモリ型並列計算機による LU 分解，SWoPP 別府 '95，95-HPC-57, pp.55–60 (1995).
21) 荒木拓也：プログラムのアルゴリズム．http://www.mtl.t.u-tokyo.ac.jp/%7Earaki/psc96/psc96.html (1996).

付　　録

A.1 プログラムの修正箇所（PSC95）

5.1 節で解析対象としたプログラムは，部分ピボット選択付ガウスの消去法[19] を用いて連立方程式を解いている．データ分散はサイクリック・サイクリック分割[20] となっている．

プログラマは，ソフトウェアパイプライニング[20]，非同期的消去[20]，多段同時消去[19] という実装方法を用いて，ピボット行および列の通信と消去の計算をオーバラップさせている．しかし，ピボット列および選択したピボット行を送信するときにブロッキング通信を用いているため，問題サイズが大きくなる（通信量が多くなる）と，性能ボトルネックとなっていた．Gordini のオーバラップ支援機能で指摘できた箇所は，ピボット列を送信する 2 カ所の送信命令である．ピボット列を持つプロセッサは，次に自分の持つ列がピボット列となるまで送信バッファを参照・代入しないことが Gordini の指摘で分かった．したがって，ブロッキング通信をノンブロッキング通信に置き換え，送信バッファを書き換える代入文の直前に通信完了命令を入れることで，さらに効率の良いオーバラップを実現できた．

A.2 プログラムの修正箇所（PSC96）

5.2 節で解析対象としたプログラムは，複素離散フーリエ順変換と逆変換を続けて行う．ただし，順変換が正しいことを確認するために，順変換の後，絶対値最大およびその次のフーリエ係数の値と各々の番号が正しいことを確認することが義務付けられている．データ分散はブロック分割[21] となっている．

プログラマは，2 次元離散フーリエ変換（DFT）に写像することで並列化を実現している[21]．行方向および列方向の DFT を行うためにはそれぞれ行列全体の転置が必要となっていて，全プロセッサ対全プロセッサの通信が生じる．プログラムでは，まず，各プロセッサがローカルに持つ部分行列の転置を並列に処理し，その後，一対一通信を組み合わせることで行列全体の転置を実現している．このときの通信がオーバラップ支援機能で指摘できた．具体的には，受信命令を前倒しすることが可能であり，ローカルな転置処理とオーバラップできることが指摘できた．

次に，通信集合化支援機能で指摘できた箇所は，順変換が正しいことを確認している箇所である．絶対値最大およびその次のフーリエ係数の値と各々の番号を求めるために，元のプログラムではそれぞれ 1 回ずつ集合通信を呼び出していた．しかし，通信バッファは個々に用意していたので，4 回の通信呼び出しを 1 回の通信呼び出しに集合化できることが指摘できた．さらに，求めた 4 つの値を放送する箇所では，値ごと，番号ごとに集合通信を呼び出していたが，上と同じ理由で 1 回の通信呼び出しに集合化できることが指摘できた．

(平成 11 年 8 月 30 日受付)
(平成 12 年 2 月 4 日採録)

伊野　文彦（学生会員）

平成 10 年大阪大学基礎工学部情報工学科卒業．現在，同大学院基礎工学研究科博士前期課程在学中．並列計算機のソフトウェア開発環境に関する研究に従事．

杉野　陽一

平成 9 年大阪大学基礎工学部情報工学科卒業．平成 11 年同大学院基礎工学研究科博士前期課程修了．同年沖電気工業入社．

山崎　良太

平成 11 年大阪大学基礎工学部情報工学科卒業．現在，同大学院基礎工学研究科博士前期課程在学中．並列計算機のソフトウェア開発環境に関する研究に従事．

藤本　典幸（正会員）

平成 4 年大阪大学基礎工学部情報工学科卒業．平成 6 年同大学院基礎工学研究科博士前期課程修了．平成 9 年同大学院基礎工学研究科博士後期課程単位取得退学．現在，同大学院基礎工学研究科助手．並列アルゴリズム，並列言語の処理系・開発環境等に興味を持つ．

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．同大学基礎工学部助手，講師，助教授を経て，平成 5 年奈良先端大教授．平成 6 年より大阪大学基礎工学部教授．平成 4〜5 年文部省在外研究員（米国メリーランド大）．現在，並列処理の基礎および応用に興味を持っている．