アルゴリズム 66−11
（1999. 1. 27）

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

杉野陽一† 伊野文彦† 山崎良太‡
藤本典幸† 萩原兼一†

†大阪大学 大学院基礎工学研究科 情報数理系専攻
‡大阪大学 基礎工学部 情報工学科

効率のよい並列プログラムを記述するため，プログラムの実行状況に関するデータをユーザに提示する性能解析システムが利用されている．一般的な性能解析システムを用いてプログラムの性能改善に有益な情報を見つけるには，複数のグラフを見比べたり，複雑な可視化結果を理解する必要があるなど，ユーザに負担がかかる場合があった．そこで本稿では，並列プログラムの実行時ログや命令間の依存情報を基に，並列プログラムの性能改善手法が適用できる可能性のある箇所を探し出しユーザに提示する機能を提案する．着目した性能改善手法とその手法の適用を支援する機能について説明し，この機能を利用した並列プログラムの性能改善例を示す．提案する機能を利用することで，並列プログラムの性能改善時のユーザの負担を軽減できると考える．

A Development of Performance Analysis System with Performance Improvement Aid Functions for Parallel Programs

Youichi Sugino†, Fumihiko Ino†, Ryouta Yamasaki‡,
Noriyuki Fujimoto† and Ken-ichi Hagihara†

†Department of Informatics and Mathematical Science,
Graduate School of Engineering Science, Osaka University
‡Department of Information and Computer Sciences,
Faculty of Engineering Science, Osaka University

A performance analysis system helps a user develop high performance parallel programs by visualizing various data on the performance. However, to improve performance of parallel programs with existing systems, sometimes user must examine many and/or complex views. In this paper, we propose performance improvement aid functions for parallel programs. These functions search points where performance improvement techniques can be applied. This paper explains these functions and shows several examples of applying them to parallel programs. These functions enable programmer to improve performance of parallel programs more easily.

1. はじめに

分散メモリ型並列計算機やワークステーションクラスタ向けの並列プログラム記述方法のひとつとして，逐次プログラムにMPI[1]やPVM[2]といったメッセージ通信ライブラリの呼び出し命令を埋め込む方法がある．この方法を用いた並列プログラム（以下，単に並列プログラムと呼ぶ）を記述する場合，プロセッサ数やデータ分散等を考慮し，通信命令を明示的にプログラム中に記述する必要がある．また，通信にかかる時間や通信の衝突等，プログラミング時に予測しにくい要素も存在する．そのため，性能のよい並列プログラムの開発には熟練を要する．慎重にプログラミングを行わなければ，通信のタイミングのずれや負荷の片寄り等が発生し，プロセッサが待ち状態に入る時間が長くなる．このような並列プログラムは性能がよいとは言えない．

実行時間がなるべく短い，性能のよい並列プログラムの開発を支援するシステムとして，プログ

ラムの改善に有益な情報をプログラマに提供する性能解析システムの研究がなされている．このような並列プログラム用性能解析システムで用いられる手法のひとつに，並列プログラムを実行させて得たトレースデータを基に性能解析を行う手法がある．Upshot[3]はトレースデータを基に各プロセスの状態と通信の様子を時間軸に沿って示す，ガントチャートを表示する．Pablo[4]は，パイチャートやバブルチャートをはじめとして，多数の可視化手段を用意しており，トレースデータとして記録する情報もユーザが自由に定義することができる．また，VAMPIR[5]は並列プログラムの実行の様子をアニメーション表示することができる．しかし，これらの性能解析システムはいずれも，並列プログラムの改善に有益な情報を見つけるには，複数のグラフを見比べたり，複雑な可視化結果を理解しなければならない場合がある．例えば，並列プログラムの実行中，各プロセッサの利用率にばらつきがあることが性能解析システムによって示されたとしても，原因が通信待ちにあるのかディスクの入出力など他の点にあるのかは別の可視化結果を見る必要がある．もし通信待ちが原因だと分かったとしても，通信待ちが発生した理由を知るには更に解析を行う必要がある．また，可視化結果がソースプログラムと対応しておらず，修正すべき箇所がどこなのかソースプログラム上で分かりにくい場合もある．

一方，並列プログラムの性能を改善する手法として，通信の一括化[6]や集合通信の利用[7]，通信と計算のオーバラップ[8]など，幾つかの手法が知られている．性能解析を行わずとも，これらの性能改善手法を適用することで並列プログラムの性能が向上する場合がある．そのため，性能改善手法の利用を念頭に置いて並列プログラムの開発を行うことが望ましい．しかし，並列プログラムの開発は逐次プログラムの開発よりも難しいため，性能改善手法の適用をプログラマが見逃してしまうことが考えられる．そのような場合，ある性能改善手法が適用できるかどうか，もしできるのであれば適用可能な箇所はどこなのかユーザに示すようなシステムがあれば，性能改善を行いたいユーザの手助けになると思われる．そこで本稿では，並列プログラムの性能改善手法が適用できる可能性のある場所を探し出しユーザに提示する機能を提案する．

2. 性能改善手法

並列プログラムの性能を改善するための手法は幾つか知られている．具体的には，

- 通信の一括化
- 集合通信の利用
- 通信と計算のオーバラップ

などがある．これらの手法を用いることで，並列プログラムの実行時間の短縮が期待できる．なお，以降で示す各種の性能データはプログラムの実行環境によって異なる．本稿では，特に分散メモリ型並列計算機 AP1000 の性能データを基に議論を進める．

2.1 通信の一括化

通信の一括化とは，送り先が等しい幾つかの通信をひとつの通信にまとめることで通信回数を減少させることである．具体的な手法としては，

- メッセージのベクトル化
- メッセージの集成

などが挙げられる．

2.1.1 通信回数削減の効果

図1は，AP1000 で，2台のプロセッサ間でメッセージの受け渡しを行った場合に要した時間を示したものである．図1より，一般に x バイトのメッセージの受け渡しにかかる時間を y とすると，2倍の情報量の $2x$ バイトのメッセージの受け渡しにかかる時間は，$2y$ よりも小さい値となる．よって，同じ情報量のデータを，メッセージを介して受け渡しするには，通信を複数回に分けるよりも1回の通信で済ませた方が短い時間で通信を終えることができる．従って，通信回数を減らすことで並列プログラムの実行時間を短縮できると期待できる．但し，通信回数を減らしたため通信のタイミングが変化し，受信待ちや同期命令の発行時間のずれ等が発生する場合があるため，常に実行時間が短縮できるとは言えない．

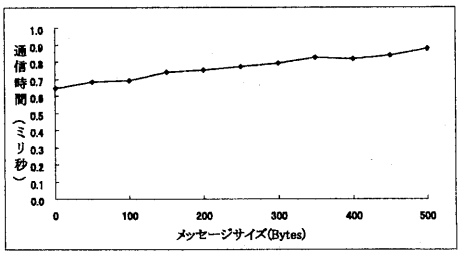

図1　メッセージの受け渡しに必要な時間

2.1.2 メッセージのベクトル化

通信を一括化するための方法のひとつとして，メッセージのベクトル化がある．メッセージのベクトル化とは，図2のように，送り先が等しい，連続したメモリ領域の送信を一度の送信で終えるようにすることを言う．

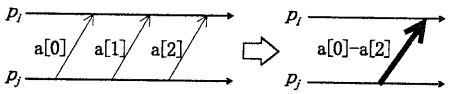

図2　メッセージのベクトル化

2.1.3　メッセージの集成

送信するデータがメモリ上で連続していない場合，一度複数のデータをバッファに格納し，その後格納したデータを一括して送信することで，通信回数を減らすことができる．この手法をメッセージの集成と言う．図3にメッセージの集成の例を示す．メッセージの集成ではメモリコピーの処理が必要だが，一般に，データのサイズが同じであれば，通信にかかる時間よりもメモリコピーにかかる時間の方が短いため，メッセージを個別に送信するより長く時間がかかることはない．但し，この場合も通信タイミングのずれによる待ちの発生がプログラムの実行時間を長くする可能性はある．

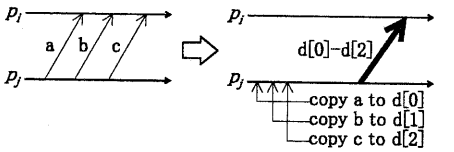

図3　メッセージの集成

2.2　集合通信の利用

p 台のプロセッサにおいて同一の送受信ルーチンを呼び出し，1対多，多対1，あるいは多対多の通信を行うことを集合通信[1]と言う．並列プログラムを実行する環境によっては，集合通信は各プロセッサと個別に通信を行うよりも短い時間で通信が終了する場合がある．例として，4キロバイトのメッセージを各プロセッサ毎に個別に送信した場合と，集合通信のひとつであるブロードキャスト通信 (以下BCと略す) を使用して送信した場合との通信時間を，プロセッサ数を変化させながら並列計算機 AP1000 上で計測した結果を図4に示す．図4より，プロセッサ数を増加させてもBCの実行時間はほとんど変化しないのに対して，個別に通信を行った場合は，プロセッサ数に比例して通信時間が増大していることが分かる．よって，すべてのプロセッサに送信するメッセージは，個別に送信を行うよりもBCを利用して送信した方が短い時間で通信を終えることができると言える．但し，BCなどの集合通信はプロセッサ間で同期が取られるため，同期処理による待ちが原因となり並列プログラムの実行時間が長くなってしまう場合がある．また，集合通信への変更と通信の一括化のどちらの手法も適用できる場合がある．例えば図5では，送り先が同じ送信を一括化し，a,b の送信と a,c の送信を行う方法と，a の送信を3つまとめてBCに変更する方法との2種類の性能改善手法が考えられる．このような場合，実際にプログラムを実行させなければ，どちらの手法がより効果を上げるか分からない可能性が高い．

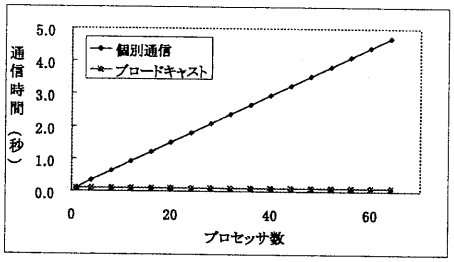

図4　個別通信とブロードキャストの通信時間の比較

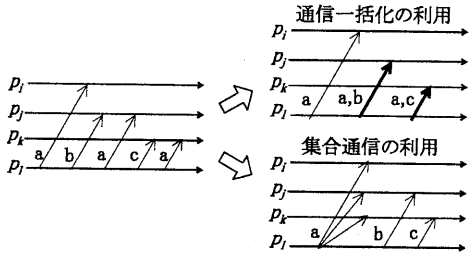

図5　通信一括化の利用と集合通信の利用

2.3　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，通信が行われている時間を利用して，計算を行うことである．通信命令の実行中，メッセージがネットワークに流れている間はプロセッサは待ち状態であるため，その時間を計算処理に当てることで並列プログラムの実行時間を短縮することができる．但し，通信と計算のオーバラップを行う際には，計算命令は以下の条件を満たしている必要がある．

- 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のことを言う．上記の条件を満たしていなければ，メッセージの送信が行われる前に送信バッファの内容を書き換えてしまったり，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てしまったりする可能性があるため，プログラムが正しく動作する保証が得られない．なお，通信と計算のオーバラップが可能かどうかはメッセージ通信ライブラリの実装に依存する[8]．

3. 性能解析システムの概要

本稿で提案する性能改善支援機能は，筆者らが開発している並列プログラム用性能解析システムに実装されている．そこでまず，開発中の性能解析システムの概要について述べる．

本システムが対象とする並列プログラムは，逐次プログラムにメッセージ通信ライブラリの呼び出し命令を埋め込んだ，分散メモリ型並列計算機やワークステーションクラスタ向けのものである．本システムの全体像を図6に示す．性能改善支援部，性能解析部の機能を用いた結果は可視化部で可視化され，ユーザに提示される．ユーザはシステムの可視化結果を基にして並列プログラムの性能改善に役立つ情報を得る．システム各部が備える機能を，表1に示す．

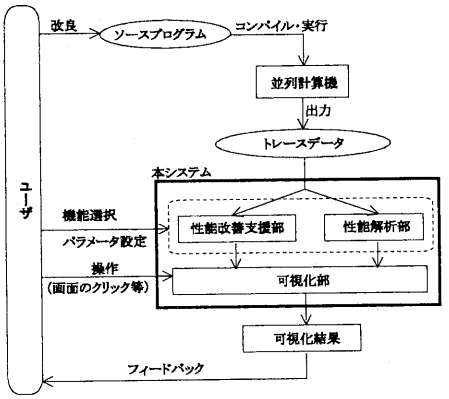

図6　システムの全体像

表1　システム各部が備える機能

性能改善支援部	通信一括化支援 集合通信への変換支援 通信–計算オーバラップ支援
性能解析部	通信状況の解析 プロセッサ情報の解析等
可視化部	棒グラフ パイチャート ガントチャート等の表示

性能改善支援機能を用いることで，ユーザは従来の性能解析システムを利用したときのように，複数の機能を何度も利用することなく，特定の性能改善手法が適用可能な箇所を知ることができる．一方，本システムが想定してない性能改善手法については，ユーザが望む情報を記録できるようにするユーザ定義イベントと，性能解析部で提供予定の検索機能とを組み合わせて用いることで，ある程度は対応できると考えている．

並列プログラム中の命令のことをイベントと呼び，命令が実行されたとき，イベントが発生したと呼ぶ．本システムで着目するイベントには，送信命令に対応した送信イベント，受信命令に対応した受信イベント，代入文に対応した計算イベントおよび発生条件と記録する情報をユーザが自由に定義できるユーザ定義イベントがある．イベントが発生すると，そのイベントに関連した情報がイベントデータとして記録される．共通して記録される情報は主に下の3つである．

- 各イベントに対応するイベント番号
- ソースプログラムの行番号
- イベント発生時刻

その他，イベントの種類に応じた情報も記録される．主なイベントとそれぞれのイベントが記録する情報を，表2に示す．プロセッサ p におけるトレースデータとは，時系列に沿った，プロセッサ p に関するイベントデータの系列のことである．

表2　各イベントが記録するイベントデータ

イベント	イベントデータの内容
送信イベント	送信相手, 送信バッファのアドレスとサイズ
受信イベント	受信相手, 受信バッファのアドレスとサイズ
計算イベント	参照, 代入が行われた変数のアドレス
ユーザ定義イベント	ユーザが自由に定義できる

4. 性能改善支援機能

トレースデータを利用して，2節で説明した性能改善手法が適用できる箇所を探し出す，性能改善支援機能について述べる．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，イベントデータ間の依存関係を考慮する必要がある．依存関係を無視して通信を一括化したり，通信と計算をオーバラップさせたりすると，プログラムが正常に動作しない可能性がある．ここで，イベントデータ a と b の間に依存関係があるとは，トレースデータ内で a と b の順序が入れ替わるような変更をプログラムに加えると，プログラムが正常に動作しなくなる可能性が出てくることを言う．具体的には，以下のような関係があれば a と b の間に依存関係がある．

- a で代入が行われた変数のアドレスと b で代入が行われた変数のアドレスが等しい．
- a で代入が行われた変数のアドレスと b で参照された変数のアドレスが等しい．
- a で参照された変数のアドレスと b で代入が行われた変数のアドレスが等しい．

4.1 通信一括化支援機能

通信一括化支援機能は，あるプロセッサのトレースデータから，ひとつの送信に一括化可能な送信イベントデータの集合を探し出し，ユーザに提示

する．それを求めるアルゴリズムのアイデアを述べる．プロセッサ p_j から p_i への送信において，一括化可能な送信の集合を求めるには，まずある送信 s の送信時刻 t_s を起点として，t_s より後に記録され，かつ s と依存関係がある計算のうち，最も早い時刻に発生した計算 c を求める．c の計算時刻を t_c とする．一括化のため，s の送信を c より後まで遅らせると，プログラムが正常に動作しなくなる可能性がある．従って，t_s と t_c の間にある送信が一括化の候補となる（図 7(a)）．一括化の候補が存在する範囲は，一括化可能な送信を求めるにつれて狭くなる．具体的には，s に近い送信 s' があれば s' の送信時刻 $t_{s'}$ と t_c の間に，s' と依存関係のある計算 c' の時刻 $t_{c'}$ が t_c よりも早ければ，$t_{s'}$ と $t_{c'}$ の間に狭められる．この様子を図 7の (b) に示す．s と s' は一括化可能である．求めた範囲の中から送信がなくなれば，一括化可能な送信の組がひとつ求められたことになる．図 7の (c) は，一括化可能な送信の候補がなくなった状態を表している．

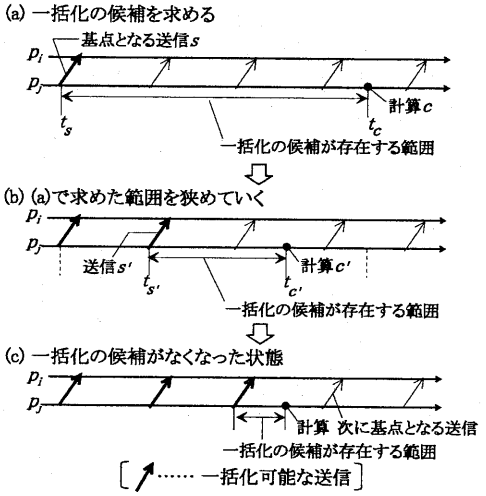

図7　一括化可能な送信を求める処理

プロセッサ q のトレースデータから，ひとつの送信に一括化可能な送信イベントデータの集合をすべて求める具体的な処理について説明する．なお，以下において，T_p はプロセッサ q のトレースデータのうち，q の計算イベントデータとプロセッサ p への送信イベントデータからなる集合を表す．また，集合変数 A は一括化チェック処理を終えた送信イベントデータの集合を，$time(x)$ はイベントデータ x が記録された時刻を表すものとする．そして，$dep(x, T_p)$ はイベントデータ x と依存関係があり，かつ記録された時刻が x より遅い T_p 内のイベントデータのうち，記録された時刻が最も早いイベントデータを表すものとする．

(1) 一括化可能な送信イベントデータの集合を格納する集合変数 R を ϕ にする．

(2) 一括化チェック処理が未適用のイベントデータ集合 $N = T_p - A$ 内で最も早い時刻に記録された送信イベントデータを求め，s とする．

(3) R に s を追加する．

(4) $c = dep(s, T_p)$ とし，$M_s = \{e \in T_p | (time(e) > time(s)) \wedge (time(e) < time(c))\}$ を満たす M_s 内にある送信イベントデータから，最も早い時刻に記録された送信イベントデータ s' を求める．

(5) **s' が存在：** $c' = dep(s', T_p)$ とする．$time(c') < time(c)$ なら $c = c'$ とする．$s = s'$ とし，(3) に戻る．
s' が存在しない： $|R| > 1$ なら R の要素は一括化可能．一括化可能な送信イベントデータの集合がひとつ求められた．

(6) R の要素を A に加える．

(7) N 内から送信イベントデータがなくなれば，一括化可能な送信イベントデータの集合はすべて求められたので終了．N 内に送信イベントデータが残っていれば，(1) へ戻る．

上記の処理を，T_0 から T_n（n はプロセッサに割り振っているプロセッサ番号の最大値）まで順番に行えば，プロセッサ q のトレースデータから，ひとつの送信に一括化可能な送信イベントデータの集合をすべて求めることができる．

通信一括化支援機能が行う処理の例を図 8に示す．ここで，Send とは送信イベントを，Cal は計算イベントを表している．図 8では，1. を遅らせることで 3. とまとめることができる．しかし，2. は送信先が異なるため一括化できない．また，5. の送り先は 1. および 3. と同じだが，一括化のため 1. を 5. の位置まで遅らせようとしても，4. との間に依存関係があるため遅らせることができない．よって 5. は 1. と 3. の組と一括化できない．通信一括化支援機能の指摘に基づいて修正したプログラムを実行した場合，図 8下部のようなトレースデータが得られる．

通信一括化支援機能の可視化例を図 9に示す．ひとつの送信に一括化可能な複数の送信がソースプログラム上でハイライトされている．また，ループ内の送信については，何度目の実行が一括化の対象として提示されているのか，左側の領域で表示している．

4.2 集合通信への変更支援機能

この機能では，集合通信へ変更可能な通信イベントデータの集合を求める．通信イベントデータ

元のトレースデータ (p_0)

	イベント	参照アドレス	代入アドレス	送信先
1.	Send	a		p1
2.	Send	b		p2
3.	Send	c		p1
4.	Cal		a	
5.	Send	d		p1

⇓

一括化後のトレースデータ (p_0)

イベント	参照アドレス	代入アドレス	送信先
Send	b		p2
Send	a,c		p1
Cal		a	
Send	d		p1

図8　通信一括化の例

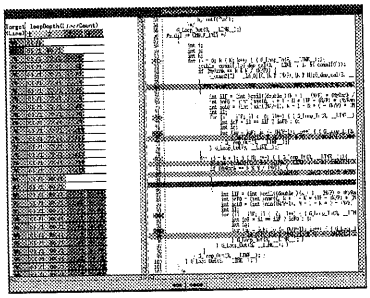

図9　通信一括化支援機能の可視化例

とは，送信イベントデータと受信イベントデータの両方を指す．現在は，全プロセッサを対象とするBCに変更可能な送信イベントデータの集合を求めているが，他の集合通信への対応も予定している．また，集合通信は全プロセッサ間で送受信を行うだけでなく，一部のプロセッサで行うこともできる．一部のプロセッサを対象とする集合通信にも，今後対応していく予定である．

BCに変更可能な送信イベントデータを求める処理は，以下の2つの段階に分けられる．

(1) BCに変更可能な送信イベントデータの集合を求める．

(2) ひとつのBCに変更可能なBCイベントデータの集合（(1)によって求められる送信イベントデータの集合もひとつのBCイベントデータとみなす）を求める．

(1)では，4.1節のように送り先が同じ送信イベントデータを調べていくのではなく，同じメッセージを送信している，各プロセッサへの送信イベントデータをひとつずつ求めていく．ある特定のプロセッサへ送信を行っているイベントデータが複数あれば，なるべく通信を遅らせないように，記録された時刻が早い送信イベントデータを選択する．もし自プロセッサ以外のすべてのプロセッサに同じメッセージを送信していることが分かれば，それらの送信イベントデータはBCへと変換が可能である．但し(1)では，ひとつの送信イベントデータが重複してBCへと変更可能だと言える場合がある．重複して送信イベントデータが選択されるトレースデータの例と，BCへと変更した場合に得られるトレースデータを図10に示す．なお，プロセッサ台数は4とする．元のトレースデータでは，1,2,3の組，1,5,6の組はともにBCへの変更が可能であり，1は重複して選択されている．従って，(1)において，BCへの変更が可能な送信イベントデータの集合を求める際には，4.1節のように送信イベントデータ毎に調べるのではなく，送信バッファの一要素毎にBCに変更可能かどうか調べる必要がある．

また，(2)についても処理内容は4.1節で説明したものとほぼ同様である．基準とするBCイベントデータ b，および b と依存関係のある計算イベントデータのうち，b より後方で，かつ最も時刻の早い計算イベントデータ c との間にあるBCイベントデータを，b とまとめられる候補とし，4.1節のように，b に近いBCイベントデータから順番に，一括化可能と判断していく．なお，可視化の様子は図9と同じである．

元のトレースデータ (p_0)

	イベント	参照アドレス	代入アドレス	送信先
1.	Send	a,b		p1
2.	Send	a		p2
3.	Send	a		p3
4.	Cal		a	
5.	Send	b		p2
6.	Send	b		p3

⇓

一括化後のトレースデータ (p_0)

イベント	参照アドレス	代入アドレス
BC	a	
Cal		a
BC	b	

図10　複数のブロードキャストへの変更例

4.3 通信-計算オーバラップ支援機能

通信-計算オーバラップ支援機能は，2.3節で説明した通信と計算のオーバラップの適用を支援する機能である．ある計算 c に着目し，c が行われた時刻から遡って，c とオーバラップ可能な通信の集合を探す．c と依存関係がある命令 d が出現しない間は，通信と c とはオーバラップが可能である．オーバラップが可能な通信を求めている様子を図11に示す．

プロセッサ p の各計算イベントデータに対して，オーバラップ可能な通信イベントデータを求める

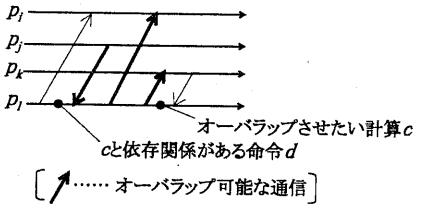

図11　オーバラップが可能な通信を求める処理

具体的な処理について説明する．プロセッサpのトレースデータをTとし，Tの中にある計算イベントデータを時刻順に並べた系列をIとする．Iの後ろから順番に計算イベントデータcを取り出し，以下のようにしてオーバラップ可能な通信を求める．

(1) $M_c = \{e \in T | time(e) < time(c)\}$ 内にあるイベントデータの内，cと依存関係があり，かつ最も記録された時刻の遅いイベントデータdを求める．

(2) $M'_c = \{e \in T | (time(e) < time(c)) \wedge (time(e) > time(d))\}$ の中にある通信イベントデータの集合Cは，cとオーバラップ可能である．

可視化の様子は図9とほぼ同様である．通信イベントデータ，そしてその通信とオーバラップ可能な計算イベントデータがハイライトされる．

5. システムの適用例

本システムの性能改善支援機能を並列プログラムに適用した例を示す．例として，集合通信への変換支援機能と通信-計算オーバラップ支援機能を使用する．通信一括化支援機能については，システムによって表示される情報が集合通信への変換支援機能と似ているため，ここでは省略する．

5.1 集合通信への変換支援機能の適用

ガウスの消去法[9]を行う並列プログラムに対して，集合通信への変換支援機能を使用したときの画面を図12に示す．

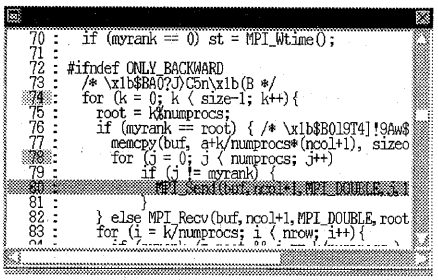

図12　集合通信への変換支援機能の適用結果

図12では，2重ループ中の送信命令がハイライトされており，BCへの変更が可能だと分かる．また，ループ中の送信の何回目の実行がBCへ変更できるかを示した図13では，外側のループでの1回目の実行，そして内側のループでの2,3,4回目の実行がBCに変更可能であることが分かる．このような結果は，外側のループの繰り返し回数が2回目を越えた後も確認された．システムがハイライトさせた行では，ループを用いて各プロセッサへ個別に同じデータを送信しているため，BCへの変更が可能である．システムの表示に従った修正をプログラムに施した結果，表3に示すように，プログラムの実行速度が改善された．プロセッサ数が増加するにつれて，BCへの変更の効果が大きく現れている．

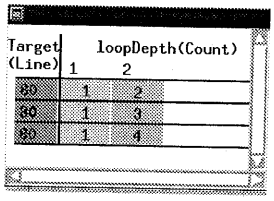

図13　ループ情報の表示

表3　ガウスの消去法の実行時間（問題サイズ512）

プロセッサ数	4	8	16	32	64
改善前 (sec)	78.80	41.28	24.13	19.09	22.35
改善後 (sec)	78.62	40.07	20.67	11.54	6.87
減少率 (%)	0.23	2.93	14.33	39.55	69.26

5.2 通信-計算オーバラップ支援機能の適用

ヤコビの反復法[9]を用いてポアソン問題を解く並列プログラムに対して，通信-計算オーバラップ支援機能を適用した．通信-計算オーバラップ支援機能の可視化結果から，2次元配列の各要素に対する代入文の一部がオーバラップ可能であることが分かった．このプログラムでは，$m \times n$の行列A，Bを各プロセッサが持っており，Aの$(i-1,j),(i+1,j),(i,j-1),(i,j+1)$要素を参照して$B$の$(i,j)(1 < i < m, 1 \leq j \leq n)$要素を求める操作と，$B$の$(i-1,j),(i+1,j),(i,j-1),(i,j+1)$要素を参照して$A$の$(i,j)(1 < i < m, 1 \leq j \leq n)$要素を求める操作を交互に繰り返す．但し，$A$から$B$を求める操作では，$A$の1行目と$m$行目に受信データを格納する必要がある．$B$から$A$を求める操作でも同様である．従って，$A$を求める操作，$B$を求める操作ともに，2行目と$m-1$行目の各要素の計算には受信データが必要なため通信とオーバラップできないが，3行目から$m-2$行目の各要素の計算は，受信データを必要としないため，通信とオーバラップが可能である．

システムから提示された情報に従ったプログラムの修正例を図14に示す．図では行列AからBへの計算を示しているが，行列BからAへの計算

についても同様の修正を行った．この修正によって得られた効果を表4に示す．改善前，改善後いずれも使用したプロセッサ数は8台である．表4より，問題サイズが大きくなるにつれて実行時間の減少の度合いが小さくなっている．この原因として，オーバラップに使用できる時間が短かった，ということが考えられる．問題サイズが大きくなるにつれて，オーバラップさせたい計算の個数も多くなるため，オーバラップの時間が短ければ一部の計算しかオーバラップできず，オーバラップの効果が小さくなる．

```
行列Bの2行目の送信
行列Bのm-1行目の送信
行列Bの1行目の受信
行列Bのm行目の受信
行列Aの各要素への代入
        ⇓
行列Bの2行目の送信（オーバラップ用）
行列Bのm-1行目の送信（オーバラップ用）
行列Bの1行目の受信（オーバラップ用）
行列Bのm行目の受信（オーバラップ用）
行列Aの3行目からm-2行目の各要素への代入
待ち命令（通信の終了を保証する）
行列Aの2行目とm-1行目の各要素への代入
```

図14　通信と計算のオーバラップの利用例

表4　ポアソン問題の実行時間（プロセッサ数8）

問題サイズ	32	64	128	256
改善前 (sec)	1.59	13.77	128.61	1162.10
改善後 (sec)	1.40	12.82	123.98	1148.51
減少率 (%)	11.95	6.90	3.60	1.17

6. 考　　察

性能改善支援機能について考察を行う．性能改善支援機能は，結果がソースプログラム上で表示されるため，プログラムのどの部分を修正すればよいかが容易に分かる．ループ中の文に対しては，ループ内での位置を示すことで，何回目の実行が性能改善手法の適用対象となっているのかが分かるようになっている．よって，これらの情報を基に，ユーザは直ちにソースプログラムの修正に取り組むことができると考えられる．但し，通信一括化支援機能によって得られた情報に従ってプログラムを変更しても，性能が向上しない場合がある．これは，通信一括化により通信のタイミングが変化し，受信待ちや，同期命令の発行時間のずれ等が発生する可能性があるためである．よって，ユーザに無駄な手間をかけさせないために，プログラムの修正によって性能向上がどれだけ望めるかをあらかじめある程度予測する機能が必要だと考えられる．

7. ま と め

本稿では，並列プログラムの性能改善支援機能について述べた．並列プログラムの性能改善支援機能として，通信一括化支援機能，集合通信への変更支援機能，通信-計算オーバラップ支援機能の3つを紹介した．並列プログラムに性能改善支援機能を適用し，有効性について考察した．性能改善支援機能を用いることで，特定の性能改善手法が適用できるかどうか，もし性能改善手法が適用できるならば，適用できる箇所はどこなのか，ソースプログラム上で知ることができる．性能改善支援機能の利用により，並列プログラムの性能を向上させたいユーザの負担を軽減できると考えられる．今後の課題としては，他の性能改善手法への対応や改善手法を適用した場合の効果の予想，性能改善支援機能では対応し切れない並列プログラムに対する解析機能やユーザ定義イベントを利用した検索機能の充実等が挙げられる．

謝辞　本研究は一部文部省科研費（平成9～10年度基盤研究 (C)(2)09680336），並列・分散処理研究推進機構の補助による．並列計算機 AP1000 を利用させて頂いた富士通 (株) 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/pvm_home.html (1998).
3) Herrarte, V. and Lusk, E.: Studying parallel program behavior with upshot, Technical Report ANL-91/15, ANL (1991).
4) 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, *Proceedings of the Scalable Parallel Libraries Conference*, IEEE Computer Society, pp. 104–113 (1993).
5) VAMPIR: http://www.pallas.de/pages/vampir.htm (1998).
6) Barasundaram, V., Fox, G., Kennedy, K. and Kremer, U.: An interactive environment for data partitioning and distribution, *Proceedings of the 5th Distributed Memory Computer Conference*, IEEE Computer Society (1990).
7) Gropp, W. and Lusk, E.: Tuning MPI Applications for Peak Performance, http://www.mcs.anl.gov/Projects/mpi/tutorials/perf/index.html (1997).
8) Gropp, W.: An Introduction to Performance Debugging for Parallel Computers, Technical Report MCS-P500-0295, ANL (1995).
9) Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P.: *NUMERICAL RECIPES in C*, Cambridge University Press (1988).