情報処理学会論文誌　コンピューティングシステム　Vol. 1　No. 3　67–82 (Dec. 2008)

負荷バランスの動的最適化による MPI ブロードキャスト性能改善

曽我武史[†1]　栗原康志[†2]　南里豪志[†3]
黒川原佳[†4]　村上和彰[†2]

2 点間通信の組合せによって構成されるブロードキャスト通信には多くのアルゴリズムがあり，それらのアルゴリズムは MPI ライブラリとして実装されている．これらのライブラリでは，集団通信に参加するすべてのプロセスが同時に通信を開始するとうい仮定のもとに最適なアルゴリズムを使用しているが，実際の通信においてそれぞれのプロセスが通信を開始する時間は負荷バランスの不均衡等の要因により異なっている．通信開始時間の相違は通信アルゴリズムの実行時に予定しない待ち時間を生むが，二項木や二分木等のアルゴリズムでは，適切な順序で 2 点間通信を行うことでこの待ち時間を削減することが可能である．本研究では．プログラム実行中の各通信プロセスで計測された待ち時間をもとに適切な 2 点間通信の順序を決定し，決定した順序に従って動的に 2 点間通信の経路を切り替えることで，プログラム実行中にブロードキャスト通信性能の改善を実現している．また，実験によって本手法がブロードキャスト通信性能の改善に効果があることを確認した．

Dynamic Optimization of Load Balance in MPI Broadcast

Takeshi Soga,[†1] Kouji Kurihara,[†2] Takeshi Nanri,[†3]
Motoyoshi Kurokawa[†4] and Kazuaki Murakami[†2]

There are many algorithms that compose broadcast from point-to-point communications, such as Binary Tree and Binomial Tree. Though many implementations of these algorithms are proposed in MPI libraries, most of them are based on an assumption that all processes begin the broadcast at the same time. That means the orders of the point-to-point communications in the broadcast are arranged numerically, according to the rank of each process. However, naturally each process starts broadcast at different times, mainly because of the imbalance of workload of each process. That causes unnecessary waiting time on processes. Our method purposes to solve this problem dynamically. This method solves this problem by profiling the waiting time of each process at runtime and adjusting the orders of point-to-point communications according to the information. The experimental result shows that this method is effective for the improvement of the broadcast communications.

1. はじめに

ブロードキャスト通信を 2 点間通信によって実現するアルゴリズムは二項木，二分木等多種提案されており，これらのアルゴリズムは MPICH [1] や OpenMPI [2] 等で MPI ライブラリとして実装されている．これらの実装ではすべてのプロセスがブロードキャスト通信を同時に開始するという仮定のもとで通信経路を決定しているが，実際の通信では通信を開始する時間が負荷バランスの不均衡等の要因によりプロセスごとに異なっているため，この実装では最適な通信経路とならない可能性がある．この通信開始時間の相違は通信アルゴリズムの実行時に予定しない待ち時間を生むが，二項木や二分木等のアルゴリズムでは，負荷の大きいプロセスに対応するランクでの受信を遅らせることにより待ち時間の削減が可能である．また，二項木や二分木のアルゴリズムでは送受信に要する時間がプロセスごとに異なるため，通信以外の負荷が大きいプロセス上での送受信の負荷を小さくすることにより，負荷バランスの不均衡を軽減させることが可能である．

本研究では特に二項木アルゴリズムに着目し，負荷バランスの不均衡によるブロードキャスト通信性能の悪化を抑止するために，通常二項木の各ノードに固定して配置されているプロセスを，どのノードにも自由に配置できるようにしている．また，ノードへの配置をプログラム実行中に各プロセスで計測された通信待ち時間を用いて適切に割り当てるためのアルゴリズムを考案した．さらに，このアルゴリズムが動的最適化可能になるための MPI ライブラリ実装を MPI_Bcast 関数に対して施すことにより，通信性能の改善を図っている．

動的最適化はシステムの動作中に適応的に最適化を行っていく手法で，本研究において

†1 九州先端科学技術研究所
Institute of Systems, Information Technologies and Nanotechnologies
†2 九州大学大学院システム情報科学府
Graduate School of Information Science and Electrical Engineering, Kyushu University
†3 九州大学情報基盤センター
Computing and Communications Center, Kyushu University
†4 理化学研究所情報基盤センター
Advanced Center for Computing and Communication, RIKEN

は，アプリケーションプログラムの実行中に与えられる様々な負荷バランス状態に合わせて，そのプログラムの実行中に適宜理想的なノード割り当てを行っていくことを指している．本研究における動的最適化手法は，疎行列演算において子プロセスごとに処理列を割り当てて演算する場合等，集団通信が同一パターンの負荷バランス下で繰り返し行われた場合において有効となるが，どのようなパターンであっても適応的に最適化を行えるためライブラリとしての汎用的な使用が可能になっている．また，本動的最適化手法はノード間の通信レイテンシが等しくなるような，ネットワークが均一なシステムを対象としている．

以下 2 章では関連研究について記し，3 章では二項木アルゴリズムによるブロードキャスト通信の手法と特徴を説明する．4 章では動的最適化によるブロードキャスト通信の最適化の手法を述べ，5 章において動的最適化の効果を実験結果によって検証し，6 章においてまとめを行う．

2. 関 連 研 究

従来より静的または動的な手段をもって集団通信の性能を改善する手法は多数発表されている．

静的な手法ではプログラムの実行前に最適化を行う[3)–6)]．これらの静的な手法では最も効果的なアルゴリズムを選択するために，プロセス数や転送データ量，ネットワーク構成，ハードウェア仕様等の静的な要素を使用する．たとえば Vadhiyar ら[3)] は MPI ライブラリのインストール時に，メッセージサイズごとに数種類のアルゴリズムを用いて通信を行い通信性能を計測し，その結果からインストールしたハードウェア環境におけるメッセージサイズごとの最適なアルゴリズムを選び出している．静的な最適化手法ではプログラムを実行する前にアルゴリズムの選択は完了しているので，選択されたアルゴリズムが現在走行中のアプリケーションに対して最適であるかどうかを知ることはできない．

動的な手法としては Faraj ら[7)] が動的に集団通信を最適化できる MPI ライブラリ “STAR-MPI” を発表している．STAR-MPI では集団通信ごとに複数のアルゴリズムを実装している．STAR-MPI はプログラムが開始されてからそれぞれの集団通信が数十回から数百回実行されるまでの期間を，実装している各アルゴリズムを用いたときの実行時間を計測するためのテストにあて，テストフェイズが終わったときに最も速かったアルゴリズムを以後の集団通信からは使用する．STAR-MPI は定期的にテストフェイズを実行することによって現在選択しているアルゴリズムが現時点でも最速であるかの確認を行い，必要であればアルゴリズムの入れ替えを行う．動的最適化手法を用いれば走行中のプログラムに合わせて最適化が可能になるが，それにはいくらかのコストを必要とする．STAR-MPI の場合は，テストフェイズにおいて最適なアルゴリズムを選択するためには実行時間の遅いアルゴリズムであっても定期的に実行する必要があり，そのコストによってプログラムの実行時間は遅延する．

ブロードキャスト通信を高速化するための手法としては Application-Bypass Broadcast[8)] が提案されている．この手法ではブロードキャスト通信専用のスレッドを生成してそのスレッドを介して通信データを送受信する．通信データの順序保証は mutex を用いた排他制御によって実現している．この手法は単体のブロードキャストを高速に行うことは可能だが，実行中のアプリケーションに対して割込みをかけることによる全体性能の低下が問題となる．

本研究では，動的最適化を行うことによって現在走行中のアプリケーションに対して適応的に最適化を施すことを可能としている．また，アルゴリズムを変更しないことによって，つねに改善に向かう低コストな最適化を行うことを企図している．

3. ブロードキャスト通信

3.1 ブロードキャスト通信アルゴリズム

MPI_Bcast は MPI 標準[9)] により定義されている関数で，ルートからのメッセージをルートを含むグループ内の全プロセスに送る（**図 1**）．ここで，プロセスとは MIMD 形式で独自のコードを実行する最小単位，グループとは MPI プログラムを構成するプロセスの集合で，ここでは集団通信に参加するプロセスの集合のこと, ルートとはブロードキャスト通信においてはメッセージの発信元となるプロセスを指し，それぞれ MPI 標準によって定義されている．また，グループ内におけるプロセスに与えられる識別番号のことをランクと呼び，各プロセスに 0 からプロセス数 −1 まで昇順に与えられる．

MPI_Bcast を実現するアルゴリズムは複数種あるが，ここでは本研究の対象としている二項木（binomial tree）アルゴリズムについて説明する．二項木アルゴリズムは MPI_Bcast を 2 点間通信によって実現する一般的なアルゴリズムの 1 つで，受信メッセージを持つすべてのプロセスが送信を行うことで，最小の送信ステップ数でブロードキャスト通信を可能にしている．ルートからのメッセージは二項木を生成するように伝達される．

二項木の各ノードに割り当てるランクは，プロセス数とルートのランクより定式的に求めることができる．ブロードキャスト通信に参加するプロセス数を M, ルートランクを 0 としたとき，ランク 0 はメッセージをランク $2^i\{\ (i = n, n-1, n-2, ..., 0)$ n は $2^n \leq M < 2^{n+1}$

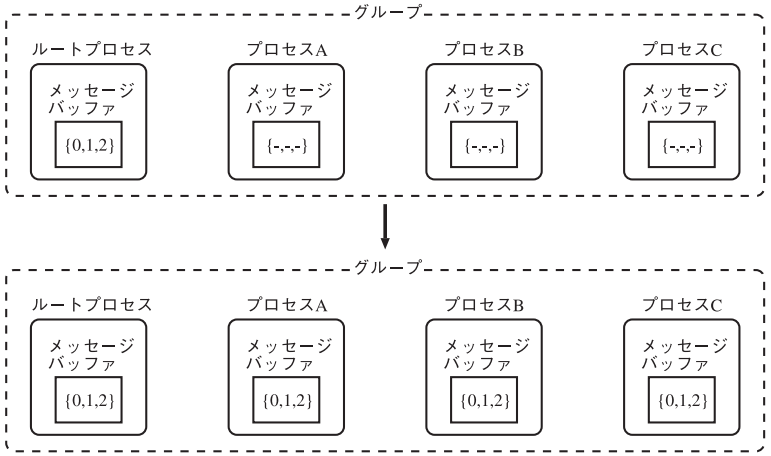

図 1　ブロードキャスト
Fig. 1　Broadcast.

を満たす値 } に対して順に送信する．メッセージを受信したランク 2^i は次に $2^i \leq I < 2^{i+1}$ を満たす I からなる部分木に対しランク 0 と同様に順にメッセージを送信する．**図 2** に 16 ノードからなる二項木におけるメッセージの伝達を図示した．円内の数字はランク，矩形内の数字は各プロセスにメッセージが伝わるまでのステップ数を示している．

二項木アルゴリズムでは受信済みの全ノードが送信を行うことから，受信ステップが早いほど送信数は多くなり，送受信処理に要する時間は大きくなる（**図 3**）．

3.2　二項木アルゴリズムと負荷バランスの関係

ブロードキャスト通信を含んだプログラム実行にかかるコストは，ブロードキャスト通信ライブラリでの性能改善可能性の観点から見ると以下の 3 つに分類することが可能である．

(1)　**ワークロード**

ブロードキャスト通信の外部で各プロセスに与えられる負荷で，その大部分は各プロセスの持つランク番号をもとにユーザによって与えられる．

(2)　**トランスロード**

ブロードキャスト通信内部でメッセージの送受信処理に要する負荷で，集団通信内部の実装アルゴリズムによって決まる．

(3)　**アイドル**

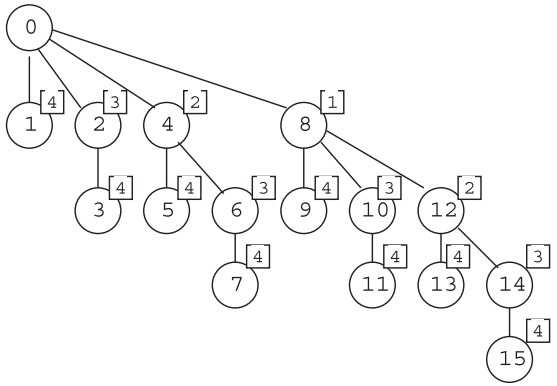

図 2　二項木アルゴリズム
Fig. 2　Binomial tree algorithm.

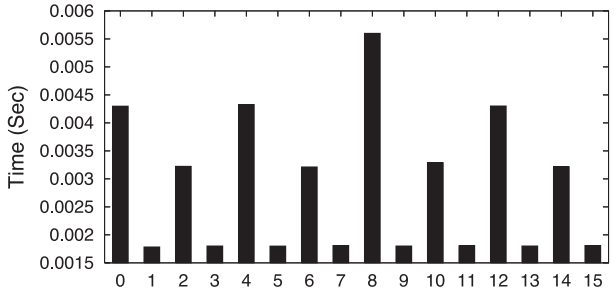

図 3　ランクごとのブロードキャスト通信時間（ルートランク=0，メッセージサイズ 16 MB，16 プロセス，3 GHz Xeon processor）
Fig. 3　Broadcast communication time for each rank.

ブロードキャスト通信内部で各プロセスがメッセージの送受信処理を開始するまでの待ち時間で，ワークロードと集団通信ライブラリ実装の組合せによって変動する．

ワークロードの量はライブラリによって制御できず，トランスロードの量はライブラリ内での処理にのみ依存する．アイドルはワークロードとトランスロードの組合せによって決まる．

ワークロード不均衡時にブロードキャスト通信が受ける影響は場合によって異なる．

図 4 にブロードキャスト通信における通信の様子を簡易図で示す．横軸は時間経過を表

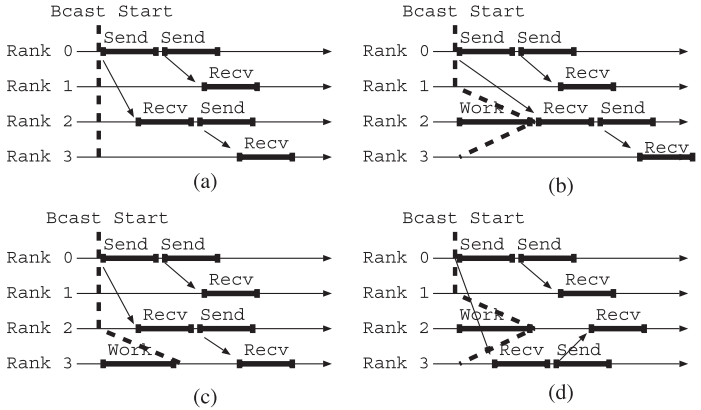

図 4　二項木と負荷の関係．(a) 負荷なし，(b) 負荷が通信に干渉，(c) 負荷は通信に無干渉，(d) 通信経路を変更
Fig. 4　Load balance on binomial tree.

し，縦軸に示された番号のランクがある時間においてどのような通信をしているかを示している．いずれもルートランクを 0 とし，図中の破線は各ランクにおけるブロードキャスト通信の開始時間をつなぐ線である．

図 4 (a) は負荷に不均衡がなくすべてのランクで同時にブロードキャスト通信が開始された場合を示す．ルートであるランク 0 はランク 1 および 2 に送信を行って通信を終了し，ランク 1 から 3 はそれぞれ受信を行う．ランク 2 は受信後さらにランク 3 に送信を行ってから通信を終了する．ランク 1 から 3 は通信においてアイドルが発生するが，これは本質的にアルゴリズム実行上必要な時間である．

これに対して図 4 (b) は，ランク 2 に与えられたワークロードが他のワークロードに比べて大きく，ランク 2 の通信開始時間が遅くなった場合である．ランク 3 の送信元にあたるランク 2 において受信開始が遅れることにより，ランク 3 の受信開始および受信完了が図 4 (a) と比較して遅くなる．すなわち，負荷バランスの不均衡による遅延が不均衡の起きたランクにとどまらず，他のランクにも影響を及ぼしている．結果としてランク 2 およびランク 3 の通信完了時間は図 4 (a) より遅くなる．

図 4 (c) は図 4 (b) と同程度の通信遅れがランク 3 で起きた場合である．ランク 3 の通信開始時間は遅れるが，ランク 2 からランク 3 への送信開始よりは早いためにワークロードによる影響は現れず，すべてのランクでの通信完了時間は図 4 (a) と同じになる．

ブロードキャスト通信がランク 0 による送信から始まってランク 3 の受信によって終わるまでの時間について図 4 の (a)，(b)，(c) を比較した場合，(a) = (c) < (b) となる．また，図 4 (c) は図 4 (a) では与えられていないワークロードがあるにもかかわらず通信完了時間が同じになることからも分かるように，各ランクでの受信開始から送信メッセージが送られてくるまでの待ち時間の合計は (c) < (a) ≤ (b) となる．このことは，ワークロードの不均衡と二項木アルゴリズムによるブロードキャスト通信との組合せが適切であった場合には計算機資源の効率的な利用が可能となるが，不適切であった場合には効率が悪くなることを示している．

また，ワークロードの大きなランクにトランスロードの大きいランクが割り当てられた場合には，負荷バランスの不均衡が助長される，特に二項木アルゴリズムでは，トランスロードが大きいランクは大きな部分木の親となるため，遅延が波及するランクも多くなる．

図 4 (b) において，二項木アルゴリズムにおけるランク 2 とランク 3 の役割を入れ換えて，ランク 3 がランク 0 からの受信メッセージをランク 2 に送り，ランク 2 はランク 3 からの受信のみを行うようにした場合，図 4 (c) と同様の効果が得られ，通信完了時間は図 4 (a) と同じでかつアイドルの総計は小さくなる（図 4 (d)）．

図 4 (c) および (d) のように，負荷バランスの状態に応じて柔軟に二項木アルゴリズムを構成する木構造における各ノードへのランク割当てを実施することによってブロードキャスト通信は最適化され，負荷不均衡の影響を打ち消すことが可能になる．また，最適化には負荷バランスの均等化の効果もあることから，負荷均等化によりプロセッサの待ち時間が減少することでプログラム全体の実行速度の向上も期待できる．

4. ブロードキャスト通信の動的最適化

3.2 節で示したように，二項木アルゴリズムによるブロードキャスト通信では，プロセス間の負荷不均衡を吸収し計算機資源を効率的に使用することが可能である．この効率化は，ワークロードによって生成される負荷バランスの分布パターンを知ることができれば，集団通信ライブラリ内における通信経路の切替えによるトランスロード調整により実施可能である．そこで，プログラム実行中に負荷バランスの分布パターンの測定と通信経路切替えによる最適化を行うことにより，実行時のワークロードに適応したブロードキャスト通信を実現する．

4.1 仮想ランクによるブロードキャスト通信の改善

3.1 節で述べた MPI_Bcast の実現方法のようにランクによって固定的にプロセスを二項

木のノードに割り当ててしまうと，図 4 (b) にあるような負荷不均衡による遅延の波及を改善することができない．これを改善するためにユーザに示されるランク（以下実ランク）と独立した集団通信内部で使用するランク（以下仮想ランク）を用意し，それぞれを別個に扱えるようにした．

各プロセスにおけるプログラムの処理内容は実ランクによって決まるため，実ランクはワークロードを決める要因となる．仮想ランクは集団通信時に各プロセスの二項木における位置を示し，通信経路およびトランスロードを決める要因となる．実ランクに対して仮想ランクを 1 対 1 に対応づけることにより，ブロードキャスト通信の経路が変更できる．

図 4 (a)，(b)，(c) の各通信を実ランクと仮想ランクの対応によって実現するためには，実ランクと仮想ランクが等しくなるような対応関係にすればよく，図 4 (d) の通信は，実ランクと仮想ランクが等しくなる対応関係から，実ランク 2 と実ランク 3 に対応する仮想ランクを入れ替えることで実現できる．

4.2　待ち時間計測によるブロードキャスト通信の最適化

図 5 に仮想ランクを用いたブロードキャスト最適化手法の概要を示す．

図 5 (a) はブロードキャスト通信を始める前の実ランク（Rank）と仮想ランク（VR）との関係を示す対応表（仮想ランクテーブル）であり，初期状態では実ランクと同じ番号の仮想ランクが割り当てられている．

この状態で実ランク 2 のプロセスに他のプロセスよりも大きいワークロードが与えられたときのブロードキャストの結果を図示したものが図 5 (b) である．図 5 (b) では横軸に時間経過を示し，縦軸に記された実ランクにあたるプロセスが時間ごとにどのような処理を行っているかを表示している．図 5 (c) は受信処理を開始してから実際にメッセージ処理が開始できるようになるまでの待ち時間を実ランク番号と対応づけた表になっている．この表から待ち時間最大になるのが実ランク 3 で最小になるのが実ランク 2 であることが分かる．

待ち時間があるプロセスでは，そのプロセスにおいてより早く受信メッセージが届いても処理が可能であり，また受信後の送信回数が増えたとしても受信メッセージが早く届いていれば待ち時間を送信のために使うことで，他のプロセスにおける送信による負荷の増加を軽減することが可能である．

待ち時間のないプロセスでは，そのプロセスに受信メッセージが届くのがより遅くてよい可能性や，受信後に送信がある場合には送信メッセージを待つプロセスに追加の待ち時間を発生させる可能性がある．また，そのプロセスでのワークロードが全体のボトルネックになっている場合，送信処理によってそのボトルネックをさらに増大させてしまう．

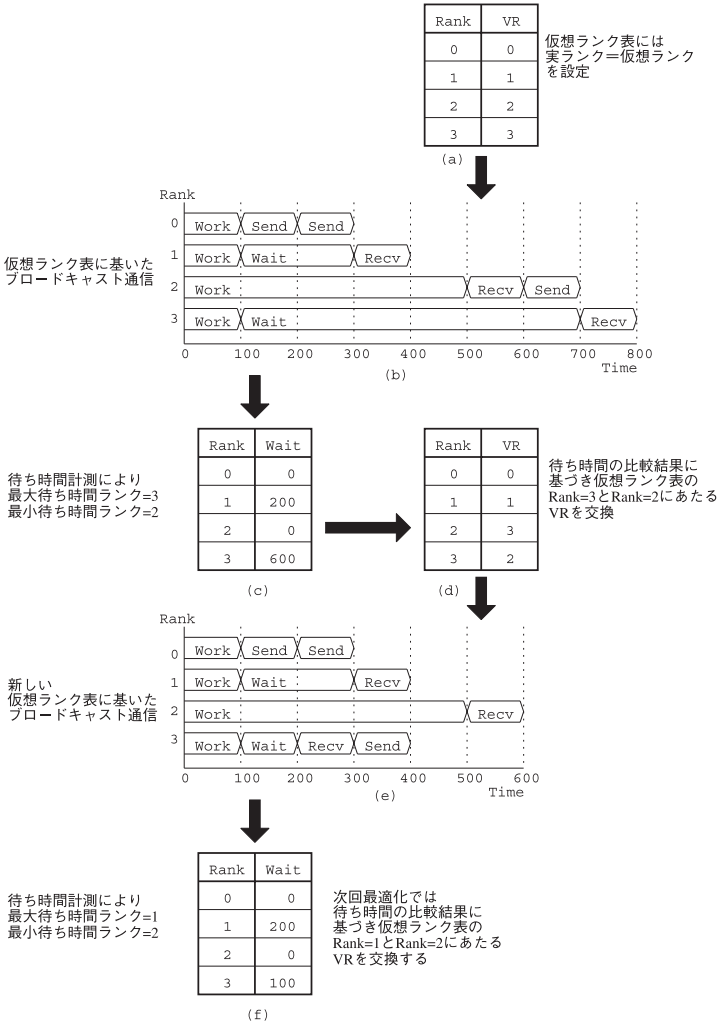

図 5　仮想ランクを用いた改善
Fig. 5　Improvement with virtual-rank.

そこで，待ち時間の小さいプロセスと待ち時間が大きいプロセスの間で仮想ランクを入れ替えることによって，負荷不均衡の改善とボトルネックの解消を図る．図 5 (c) の場合では実ランク 2 と実ランク 3 の仮想ランク入れ替えを行う．

仮想ランクの入れ替えを行ってできた新しい仮想ランクテーブル（図 5(d)）の状態で次に前回と同様のワークロードが与えられた場合には，ブロードキャスト通信の結果は図 5 (e) のようになり，そのときの実ランクごとの待ち時間は図 5 (f) で示される．

このように，ブロードキャスト通信の受信待ち時間を計測した結果に基づき仮想ランクの番号を割り当てることにより，負荷バランスの改善が可能になる．

待ち時間を用いることにより，次回以降待ち時間に大きな変化がない限りそのプロセスから新たな待ち時間情報を発信する必要がなくなるため，低コストな最適化が可能になる．

4.3　発見的手法による入れ替えを用いた仮想ランク割当て最適化

前節で説明した方法によって仮想ランクの入れ替えを発見的に繰り返すことによっても，多くの場合負荷バランスは改善される．しかしながら，待ち時間がないプロセスが複数存在した場合は，それらプロセス間におけるワークロードの比較が困難になる．

発見的手法に基づく改善を繰り返して最適解に至るためには，履歴を利用して待ち時間がないと見なせる全プロセスに対して，待ち時間最大のプロセスとの仮想ランク入れ替えを試行する必要があるが，二項木アルゴリズムのツリー構造を利用することにより，試行を効率的に行うことができる．

二項木アルゴリズムによる通信では送信回数の多いプロセスほど受信順序が早くなる．また，受信を行うプロセスは親ノードのプロセスよりも早く受信処理を開始しても効果がない．これらのアルゴリズムの特徴から，待ち時間がないと見なせるプロセスが複数ある場合には，送信回数が最も多いすなわちステップ数最小のプロセスから順に仮想ランク入れ替えの対象とする．

送信回数が同じであった場合には，対象プロセス間にはノードの親子関係はなくなるので，二項木アルゴリズム上は順序を意識する必要はない．しかしながらステップ数が同じ場合には，通信経路上の受信数が少ないと通信時間がより小さくなり受信開始時間が早まるため，受信数が少ないすなわちルートからの深さが浅いプロセスを優先して入れ替えの対象とする．

送信回数および受信数が同じ場合にはいずれのプロセスも受信順序に関して同一と見なせるので，実ランク番号等どのような優先順位をつけてもかまわない．

もし入れ替えを行って全体の待ち時間が増加した場合には，入れ替えた仮想ランクをもとに戻して次の優先順位にあるプロセスとの入れ替えを行う．全体の待ち時間が減少あるいは変わらなかった場合には，新しい待ち時間情報に基づいて最大・最小待ち時間を有するプロセスの探索を行う．

図 6 に，ワークロードをランク 0 からランク 7 まで後の番号になるにつれ増加するように与えた初期状態から最適解を得るまでの仮想ランク入れ替えの様子を図示する．図中の表は仮想ランクテーブルで，実ランク（Rank）に対応する仮想ランク（VR）を示し，その仮想ランクテーブルに従ってブロードキャスト通信を行った結果が表の右に図示されている．図 6 (a) を初期状態として，図 6 (b) から図 6 (e) には上述された手法に基づいて仮想ランク入れ替えを行い待ち時間が 0 になるまでの過程を順に示している．

図 6 (a) から図 6 (b) への入れ替えは全体の待ち時間が減少しているが，図 6 (b) から図 6 (c) への入れ替えは全体の待ち時間が増加しているため，図 6 (c) の状態をもとの図 6 (b) に戻したうえで，次の優先順位のプロセスを入れ替えの対象とする（図 6 (d)）．しかし，このときも基準となる図 6 (b) から見ると全体の待ち時間が増加しているため，再度図 6 (b) に戻したうえでその次の優先順位のプロセスとの入れ替えを行う（図 6 (e)）．

図 6 の説明においては 4 回の入れ替えにより最適解に至っているが，1 回の操作において複数組の入れ替えを許さない場合には，待ち時間増加時にもとの状態に戻すときに操作が 1 回加算されるため，6 回の入れ替えが必要になる．また，複数組の入れ替えを許す場合には，1 度の操作で待ち時間のないプロセスからなるグループと待ち時間のあるプロセスからなるグループの間で複数組の入れ替えが可能になり，入れ替えの操作回数を減少させる可能性があるが，待ち時間の大きさの順序とワークロードの大きさの順序はかならずしも一致しないため，選択されたプロセスに関するすべての組合せを試行するまで，その組合せにおける最適解が検出できない可能性もある．

4.4　ブロードキャスト関数への動的最適化適用

図 7 に動的最適化機構を実装した MPI_Bcast 関数のフローチャートを示す．MPI_Bcast 関数内では大別すると 3 つの処理を行っている．最適化フェイズでは最適化アルゴリズムを実行して新しい最適化情報を生成し，再構成フェイズでは最適化情報を伝達して仮想ランクテーブルを再構成し，プロファイルフェイズではブロードキャスト通信の待ち時間計測，加工および収集を行っている．

以下に最適化，再構成，プロファイルの各フェイズにおける処理の概要を述べる

(1)　**最適化フェイズ**

最適化フェイズはブロードキャスト通信を行うグループに含まれるプロセスの 1 つによっ

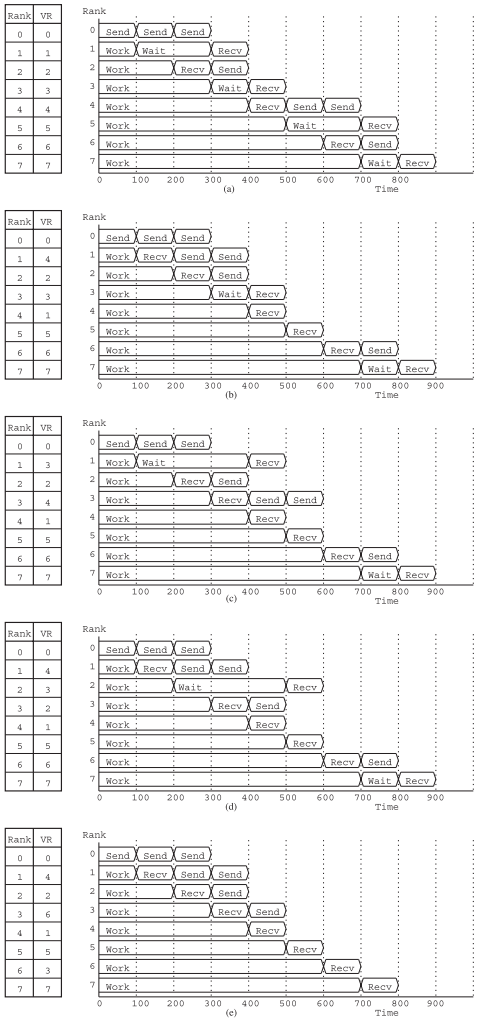

図 6　最大/最小ランク入れ替えによる最適化
Fig. 6　Optimization by swapping the maximum and minimum ranks.

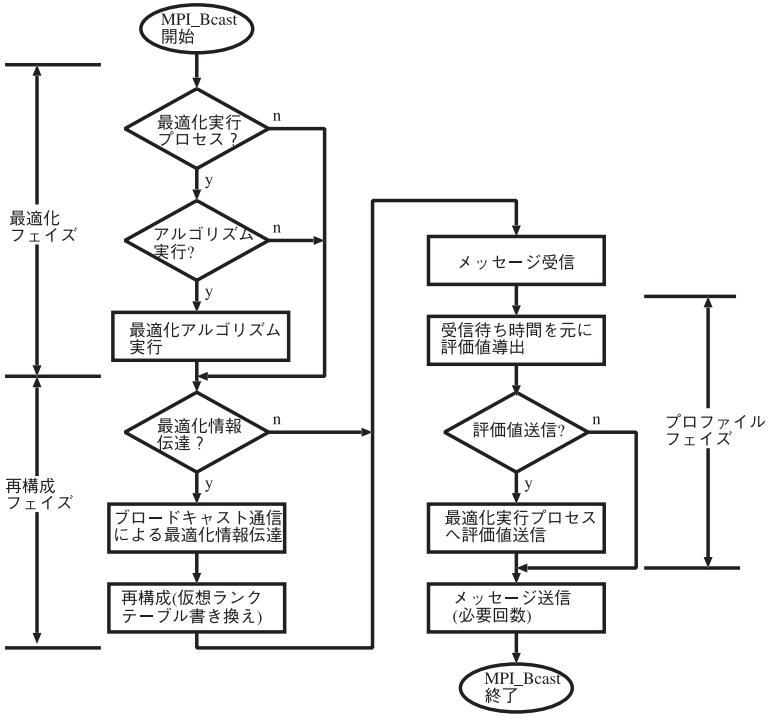

図 7　動的最適化を組み込んだ MPI_Bcast 関数の流れ図
Fig. 7　Flow chart of dynamic optimization embedded MPI_Bcast.

て実行される．

最適化フェイズでは 4.2 節に記した手法を用いて最適化機構が持つ仮想ランクテーブルを作り変えるとともに，再構成フェイズで使用される最適化情報を生成する．最適化フェイズはブロードキャスト通信を規定回数行うたびに実行される．この規定回数は，プロファイル情報の変化が少ない場合は減少させ，変化が大きい場合には増加させる．この規定回数は再構成フェイズでの最適化情報伝達間隔としても使用される．

最適化情報には，次回最適化情報伝達を行うまでの間隔および今回の交換対象の実ランクのペアを示す情報が含まれている．

(2)　再構成フェイズ

再構成フェイズはブロードキャスト通信に参加しているすべてのプロセスで実行される．

再構成フェイズでは，最適化フェイズを実行するプロセスからグループ内の全プロセスに対して最適化情報を伝達する．

最適化情報の伝達は，最適化フェイズを実行するプロセスをルートとしたブロードキャスト通信によって実施される．最適化情報を受信した各プロセスは，それぞれが持っている仮想ランクテーブルを実ランク交換情報に従って書き換えて，以後の通信は新しい表に基づいてブロードキャスト通信を実行する．

(3)　**プロファイルフェイズ**

プロファイルフェイズは全プロセスで実行され，最適化のもととなる情報として待ち時間を計測する．この待ち時間としてはブロードキャスト時に各プロセスが受信を行う際の MPI_Wait に要する時間を用いる．計測した待ち時間から相加平均，加重平均等によって評価値が算出される．

算出された評価値は MPI_Put を使って最適化フェイズを担当するプロセスに送付するが，前回送付した評価値との差があらかじめ定めた閾値を超えない場合には評価値の送付は行わない．

5. 実 験 結 果

4.4 節で述べた動的最適化手法について，2 つのベンチマークプログラムを作成して効果の確認を行った．

最初に 3.2 節であげた負荷バランスの不均衡を是正することの効果および動的最適化によるペナルティを調査するため，最大効果を測定するベンチマークプログラムを作成した．次に疎行列積計算という実際に科学技術計算分野で多用される演算における効果を測定する実験を行った．

実験は理化学研究所 RSCC（Riken Super Combined Cluster）システム上で行った．本実験では最大 128 ノードまで使用した．各ノードは Infini Band で相互接続されている．各ノードには 2CPU（Xeon 3.06 GHz）が実装されているが，本実験では各ノード 1CPU のみを使用している．各ノードのメインメモリは 4 GB，OS は Red Hat Linux（Kernel 2.4.21），コンパイラおよび MPI ライブラリは富士通製を使用している．

実験にあたっては 4.4 節で述べた各パラメータを決定する必要がある．具体的には，最適化フェイズにおける最適化アルゴリズムの実行間隔，プロファイルフェイズにおける評価値および評価値送信の閾値である．これらの値は，それぞれの相互関係，動的最適化オーバヘッドとの最適化効果のトレードオフおよび待ち時間の変化検出と外乱の除去のトレードオフに鑑みて決定している．

最適化の実行間隔を小さくした場合，評価値の変化に対する感度が向上し，繰返しのより早い段階で最適化を実施することが可能になるため最適化効果は向上するが，動的最適化オーバヘッドは上昇するので，評価値の変化があまりない場合には実行間隔を大きくしたほうがよい．そのため本実験では最適化フェイズにおいて総待ち時間を計測し，変化が小さい場合には実行間隔を大きくしている．

評価値の算出手法には外乱が大きな影響を与える．外乱がない場合には最新の待ち時間をもって評価値とすることが可能であるが，実際には大小の外乱により待ち時間が安定した値を示すことはないため，待ち時間を平均化する必要がある．本実験では待ち時間の評価値として加重平均を求めている（式 (1)）．

$$NewVal = \frac{((2^i) - 1) \times PreVal + WaitTime}{2^i} \tag{1}$$

式 (1) における PreVal は前回までの加重平均によって求められた値で，WaitTime は最新の待ち時間測定により得られた待ち時間，i は加重平均の重みを示すパラメータで整数値をとる，NewVal は加重平均により得られた新しい評価値を示す．i を小さくとると，評価値は最新の待ち時間の影響を大きく受けるためワークロードの変化を早期に検出することができるが，外乱の影響を受けやすくなる．また疎行列積算では，外乱の少ない場合に加重平均よりも高速に安定した待ち時間の値を得ることを狙って，測定された待ち時間が連続して変化しなかった場合には，それらの相加平均をもって評価値としている．

評価値送信の閾値が高い場合には，送信回数が減ることと，最適化機構における総待ち時間の変化が少なくなることによる動的最適化アルゴリズム実行間隔の広がりにより，動的最適化オーバヘッドは減少し，また外乱の影響も少なくなるが，それにともない最適化効率は低下し待ち時間の変化の検出は遅れる．本実験ではいずれの実験においてもワークロードは固定されていることから変化の検出よりも外乱の排除を優先して高めの閾値をとっている．

5.1　最大効果測定

本節では本手法の最大効果の測定を目的とした実験の結果を示す．本実験では，実ランクと仮想ランクが一致する場合にトランスロード最大となる実ランクのみにワークロードを与えて，動的最適化による性能改善率を測定する．二項木アルゴリズムにおいては，ルートランクから最初にメッセージを受信する中央のランクがトランスロード最大となる．バリアによる同期後にワークロードを与え，その後にブロードキャスト通信を実行するという手順

を 100 回繰り返し，毎回ブロードキャスト通信の開始時刻および終了時刻をプロセスごとに測定した．集団通信は 128 プロセスに対して実行し，ワークロードとして $N \times N$（N は整数）からなる正方行列間の乗算を行っている．

本実験では，最適化フェイズにおける最適化アルゴリズム実行間隔は，初期値を 1 とし，最適化アルゴリズム実行時に全体の待ち時間を集計した結果が，以前の集計に比べて 25%以下の差であった場合には次回実行間隔を 2 倍に増やし，25%を超える差であった場合には次回実行間隔を 2 分の 1 に減少させている．

また，プロファイルフェイズでは前回評価値と測定値の平均値を新しい評価値とし，前回送信評価値と新しい評価値に 50%以上の差があった場合に新しい評価値を送信する．

実験結果を評価する指標として，集団通信時間と平均通信時間を測定した．集団通信時間は全プロセスのうち最初にブロードキャスト通信を開始したプロセスの開始時間から，最後にブロードキャスト通信を終了したプロセスの終了時間までの時間である．平均通信時間はブロードキャスト通信実行時にそれぞれのプロセスが要した時間である．

集団通信時間は通信終了後にプロセス間の同期をとるようなプログラムにおいて有効な指標であり，平均通信時間はワークロードとブロードキャスト通信からなる処理終了後にプロセス間同期をとらずに以後の処理を続ける場合や，処理完了通知の順に次の処理を指示されるようなプログラムにおいて有効な指標である．

また，ランク入れ替えによる最適化の効果をみるために，理想的な仮想ランク表に基づくブロードキャスト通信に関しても測定を行った．具体的には，動的最適化に必要なプロファイルや待ち時間の大小比較，再構成のためのブロードキャスト通信等は行わず，最初から実ランク 64 と実ランク 127 の仮想ランクを入れ替えた状態で実験を行う．

集団通信時間の測定結果を**表 1** に，最適化時と非最適化時の比を**図 8** に，理想時と非最適化時の比を**図 9** に示す．平均通信時間の測定結果を**表 2** に，最適化時と非最適化時の比を**図 10** に，理想時と非最適化時の比を**図 11** に示す．

ここで，Load はワークロードの大きさを示し，数値は正方行列の大きさ N である．Type の Ori は動的最適化を適用しなかった場合の結果，Opt は適用した場合の結果，Ide は理想の結果をそれぞれ示している．また，Message Size はメッセージ長を示す．表に示された数値は時間でどちらの表も単位はミリ秒，図は性能改善比として Opt に対する Ori の時間比および Ide に対する Ori の時間比を示している．結果は各条件につき 100 回の測定を行い，表にはその平均値を示している．

図 8 では，動的最適化によって集団通信時間は $N = 0$ を除くいずれのワークロードにおいても短縮されている．N が大きくなるにつれて効果も高くなっていき最大では 40%強短縮されることが示されている．ワークロード実行後に行われていた仮想ランク 64 による送受信処理を他のワークロードのない実ランクのいずれかに移すことで，仮想ランク 64 はワークロード実行中に 1 回ある受信処理と 6 回ある送信処理の一部を実行することが可能になる．このワークロード実行中に行える送受信時間が集団通信時間の短縮として見えるため，最適化が最も効果を現すのは送受信処理時間がワークロード処理時間と同程度になるときであり，その短縮率は 50%ほどになる（**図 12**）．

表 2 および図 10 では，動的最適化による平均通信時間が N=0 を除くいずれのワークロードにおいても短縮されている．集団通信時間と同様に N が大きくなるにつれて効果も

表 1　負荷行列の大きさが 0/60/80/160/240 時の集団通信時間
Table 1　The overall time for load matrix size = 0, 60, 80, 160, 240.

Load		0			30			40			80			160		
Type		Ori	Opt	Ide	Ori	Opt	Ide	Ori	Opt	Ide	Ori	Opt	Ide	Ori	Opt	Ide
Message	1B	1.110	1.087	1.061	2.122	2.103	1.910	3.612	3.427	3.442	29.09	28.86	28.83	180.6	180.0	180.2
Size	1KB	1.105	1.119	1.100	2.158	1.888	1.906	3.709	3.463	3.436	23.91	23.72	23.70	183.7	183.4	183.7
	16KB	1.494	1.446	1.475	2.512	1.961	1.877	4.042	3.522	3.533	30.53	29.95	30.01	189.7	188.6	188.4
	32KB	1.949	1.958	1.892	2.865	2.018	1.962	4.403	3.458	3.455	24.27	23.36	23.34	176.6	174.9	174.7
	128KB	3.740	3.791	3.768	4.585	3.897	3.785	6.241	3.960	3.849	26.40	23.86	23.70	184.2	180.7	180.5
	1MB	24.58	24.94	24.60	25.07	25.10	24.76	25.81	25.46	25.37	48.18	28.27	27.13	231.5	210.2	209.1
	8MB	202.5	203.4	203.6	197.3	198.3	198.6	201.0	204.5	203.6	223.0	206.7	204.6	390.8	227.9	219.7
	16MB	404.9	407.1	407.4	402.0	407.0	408.0	410.2	411.0	410.7	414.5	410.1	409.1	580.1	414.0	405.4

Ori: Original　Opt: Optimized Ide:Ideal　Time = (ms)

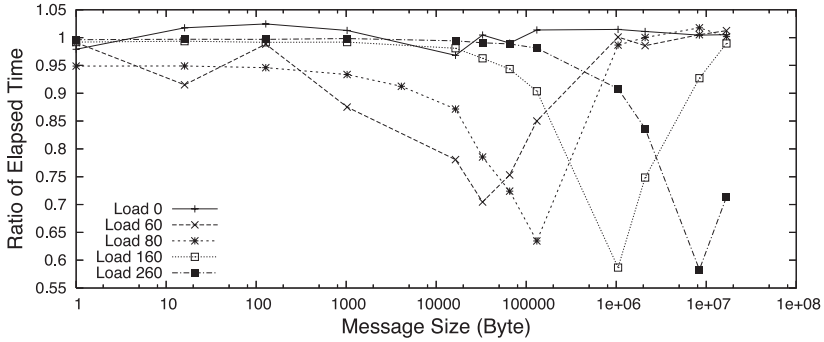

図 8　集団通信時間比（最適化時/非最適化時）
Fig. 8　The overall time ratio of original broadcast to optimized broadcast.

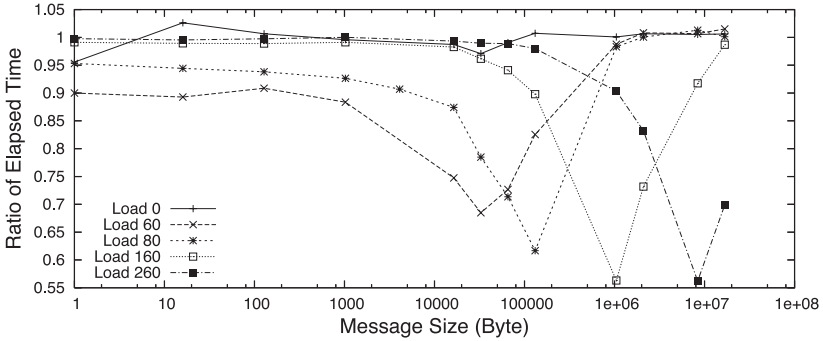

図 9　集団通信時間比（理想時/非最適化時）
Fig. 9　The overall time ratio of original broadcast to ideal broadcast.

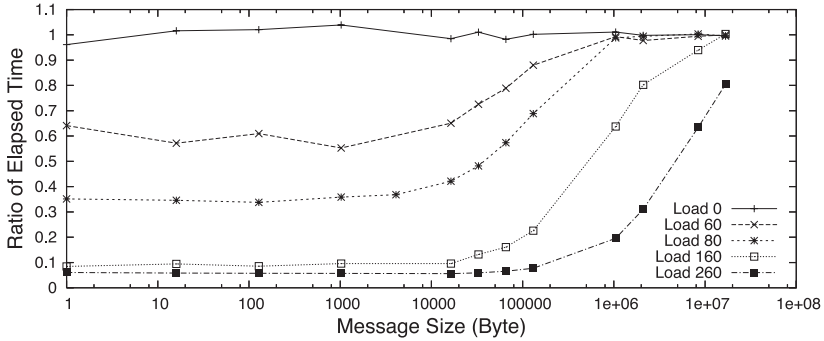

図 10　平均通信時間比（最適化時/非最適化時）
Fig. 10　The average time ratio of original broadcast to optimized broadcast.

表 2　負荷行列の大きさが 0/60/80/160/240 時の平均通信時間
Table 2　The average time of broadcast for load matrix size = 0, 60, 80, 160, 240.

Load		0			30			40			80			160		
Type		Ori	Opt	Ide	Ori	Opt	Ide	Ori	Opt	Ide	Ori	Opt	Ide	Ori	Opt	Ide
Message	1B	0.5916	0.5687	0.5562	1.086	0.6962	0.5635	1.826	0.6412	0.5686	14.32	1.206	0.4650	88.71	5.350	0.2239
Size	1KB	0.5751	0.5976	0.5826	1.108	0.6122	0.5875	1.867	0.6692	0.5728	11.79	1.130	0.5291	90.22	5.144	0.2511
	16KB	0.8321	0.8196	0.8425	1.341	0.8726	0.8127	2.096	0.8827	0.8464	15.09	1.444	0.7275	93.24	5.214	0.5626
	32KB	1.124	1.136	1.097	1.573	1.142	1.090	2.329	1.122	1.085	12.12	1.600	1.018	86.94	5.154	0.8991
	128KB	2.468	2.473	2.455	2.865	2.522	2.451	3.670	2.526	2.457	13.66	3.085	2.543	91.01	7.028	2.619
	1MB	17.85	18.05	17.98	18.41	18.28	18.05	18.51	18.28	18.29	29.43	18.77	18.17	119.6	23.36	18.34
	8MB	150.6	150.7	150.6	151.7	150.9	150.5	150.0	150.5	150.7	161.7	152.0	151.0	245.1	155.4	149.4
	16MB	342.0	340.6	339.7	342.7	342.5	338.3	341.7	340.1	339.5	340.1	341.2	339.9	427.3	343.8	337.8

Ori: Original　Opt: Optimized Ide:Ideal　Time = (ms)

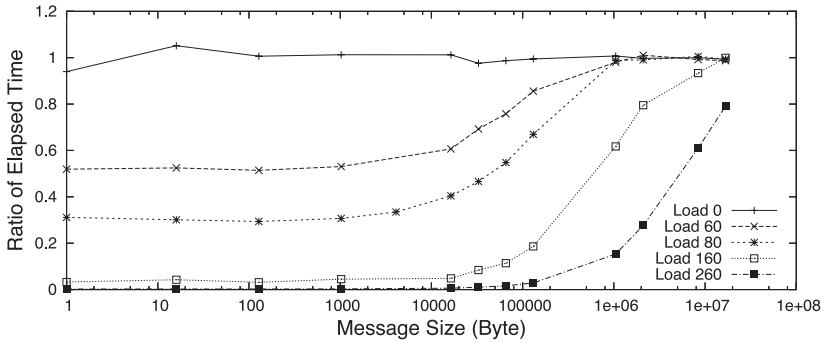

図 11　平均通信時間比（理想時/非最適化時）
Fig. 11　The average time ratio of original broadcast to ideal broadcast.

高くなっていき最大では 90%以上の短縮がなされている．平均通信時間には全プロセスのアイドルが加算されるため，最適化はワークロードが大きいほど効果を示し，トランスロードがアイドルに比べて小さいほど最適化による改善効果は高くなる．

N=0 のときは集団通信時間で最大約 2%, 平均通信時間で最大約 4%の悪化が見られるが，同じ N=0 のときに最大 4%程度の改善も見られるため，最適化フェイズでの計算および更新フェイズでのブロードキャスト通信に費やす時間はほとんど通信性能に悪影響を及ぼさないものと考えられる．

理想時と比較すると集団通信時間では 3%程度，平均通信時間では多くは 5%程度，最も差がある N=60, メッセージ長=1 バイトのときには 12%程度比率が悪化している．本実験のように待ち時間最小ランクの検出が比較的容易な場合には，精度の良い仮想ランクの入れ替えができており，ボトルネックの解消にともなう集団通信時間の改善効果は理想時により近くなるが，その他のランクにはすべて等しくワークロードを与えていないために，ワークロードを与えていないランクどうしでの入れ替えが発生するため，平均通信時間は集団通信時間ほど理想時に近づけないものと考えられる．

次に最大効果実験の派生実験として，プロセスごとにワークロードとして 0 から百万回までのランダムな回数の倍精度浮動小数点乗算を与える実験を行った．実験は乱数によって作成された 200 の負荷パターンについて行った．各プロセスで実行される乗算回数はランク

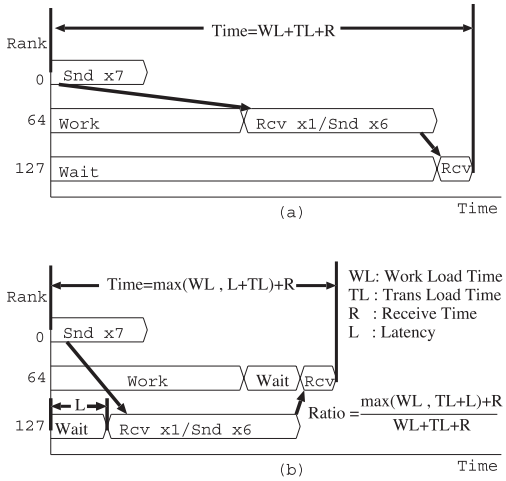

図 12　集団通信時間の短縮率
Fig. 12　Reduction ratio of the overall time.

番号および負荷パターン番号をシードとして生成された 31 bit の乱数を整数と見なし，その乱数値を百万で割ったときの剰余を使用している．

乱数によって作成された 200 のパターンのそれぞれについて，最大効果実験と同様にブロードキャスト通信を 100 回行い，その集団通信時間および平均通信時間を最適化を行った場合と最適化を行わなかった場合について測定し，その時間比を求めた．**図 13** および **図 14** はメッセージ長 1 バイトと 1 メガバイトのときに測定された時間比を改善率の高い順に並べた図で，縦軸に非最適化時に対する最適化時の計測時間比，横軸に改善率の高さの順序をとっている．図中の Overall Time は集団通信時間，Average Time は平均通信時間を表している．

メッセージ長 1 バイト時の集団通信時間では，分布図からは最大約 17%の改善効果があるように見えるが，実際には図 8 にも示されているとおりどのようなワークロードに対しても改善効果はない．図 13 においても時間比 0.95 から 1.05 の範囲に 66.5%の計測結果があてはまることからも通常の状態では時間比は 1.0 に近いであろうことが分かる．このことから，改善効果があるように見えるのは，非最適化時の測定では大きな外乱により通信に遅延が発生したが，最適化時の測定では発生しなかったためと考えられる．同様なことが時間比の上昇に対しても起きていることは，この集団通信時間分布が Order=100，Ratio=1.0

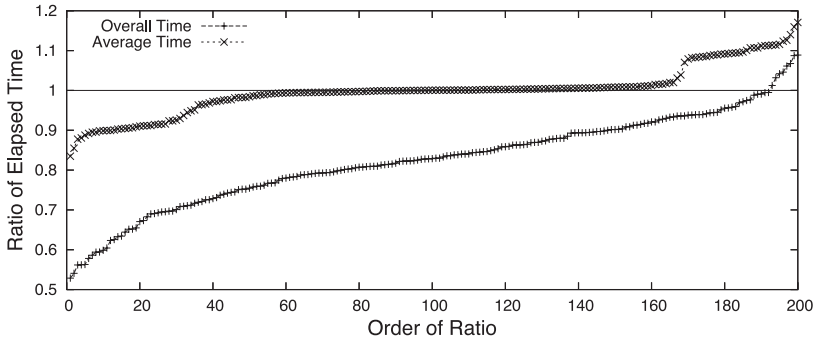

図 13　通信時間比分布（メッセージ長=1 byte）
Fig. 13　The distribution of ratio of elapsed time (messagesize = 1 byte).

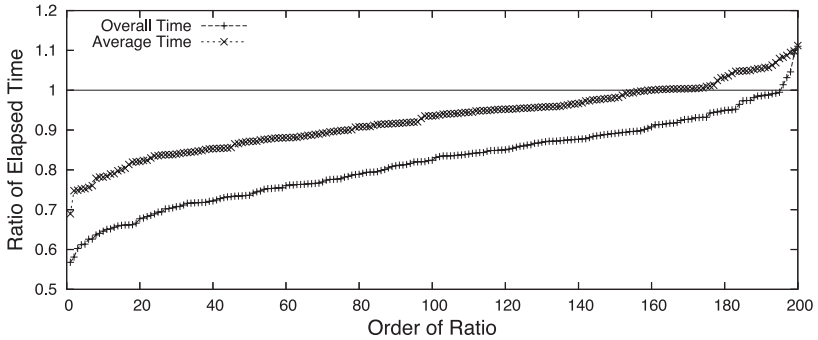

図 14　通信時間比分布（メッセージ長=1 M byte）
Fig. 14　The distribution of ratio of elapsed time (messagesize = 1 megabyte).

を中心にほぼ点対称の分布を示しているからも分かる．いい換えれば，この分布図から動的最適化によるオーバヘッドは通信の遅延を引き起こしていないことが分かる．

メッセージ長 1 メガバイト時の集団通信時間分布では 58%の測定値が 0.95 以下の時間比を示しており 6.5%の測定値が 1.05 以上の時間比となる．

平均通信時間分布ではどちらのメッセージ長においても同様の傾向となって，1 バイトでは 89%，1 メガバイトでは 90%の測定値が 0.95 以下の時間比を示し，1.05 以上の時間比となるのは 1 バイトで 2%，1 メガバイトで 1%であった．

平均通信時間は図 10 にあるように，各プロセスに与えられる負荷の差が十分あればメッセージ長にかかわらず動的最適化の効果が現れるので，メッセージ長が 1 バイトであっても性能は改善される．また，あるプロセスへの外乱によって集団通信時間が伸びた場合であってもそのプロセスからの遅延の伝播が少なければ，すべての通信経路が評価対象となる平均通信時間には影響がほとんど現れないため，そのような場合においては集団通信時間比は悪化を示し平均通信時間比は改善を示す．したがって平均通信時間で比較した場合は，集団通信時間で比較した場合に比べ，動的最適化による性能改善の可能性が高くなる．

これらの結果は，本動的最適化手法がランダムなデータに対しても高確率で通信性能を改善させることができることを示している．しかしながら，本実験のみでは 1 メガバイトのバイト長での測定において，集団通信時間比で最悪 10%程度性能を悪化させている原因が外乱によるものなのか，本手法が作用した可能性があるのかは判断がつかなかった．この判別のためには毎回の通信内容をより詳細にとることが必要である．

5.2　疎行列乗算

本節では本手法を疎行列乗算に適用した実験の結果を示す．この実験では CCS 形式（Compressed Column Storage format）[10] で圧縮された疎行列の乗算を行い，動的最適化を行った場合と行わなかった場合および仮想ランクの割当てのみ最適化を行った場合における実行時間を測定した．

CCS 形式では，疎行列を以下の column-pointer，row-index，val の 3 つの配列に格納している．

- `column-pointer`：各列における先頭 row-index，val の位置を示す．
- `row-index`：各非ゼロ要素のある行位置を示す．
- `val`：各非ゼロ要素の値を示す．

CCS 形式によって圧縮された行列間の乗算を行うために行要素と列要素を抽出する場合，列要素の抽出は colomn-pointer[i] から column-pointer[i+1]−1 に対応する row-index および val を連続的に取り出すため高速で行えるが，行要素の抽出には全 row-index から該当する行位置の要素を探索する必要があるため低速である．そこで，本実験で用いた並列疎行列積では，ルートプロセスで行要素の抽出を行い，抽出した行要素を MPI_Bcast によって全プロセスに分配し，行要素を受け取った各プロセスは自プロセスに割り当てられた列の範囲において乗算を実行する手法によって実施した．

実験では同一の正方行列どうしの乗算を行った．実験に用いる行列は行列サイズ（N），非ゼロ要素割合（R）および列間の非ゼロ要素傾き（C）の 3 要素より生成した．ここで列間の非ゼロ要素傾きは，第 $N/2$ 列に含まれる非ゼロ要素数と，第 1 列（および第 N 列）に含まれる非ゼロ要素数との割合を示しており，第 1 列から第 $N/2$ 列および第 $N/2$ 列から第 N 列までの列に含まれる非ゼロ要素数 $NZ(i)$ は両端を結ぶ一次関数で与えられる．

$$NZ(i) = C \times NZ(\frac{N}{2}) + (i-1)\frac{(1-C)NZ(N/2)}{(N/2)-1}$$

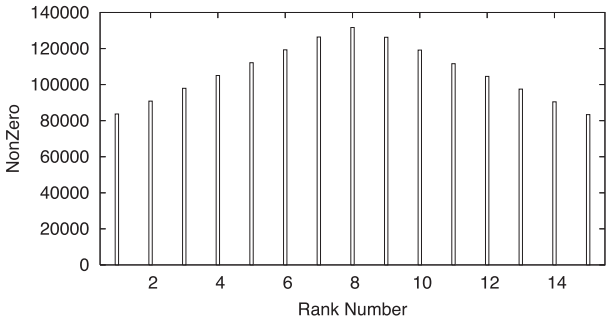

図 15　非ゼロ分布（$N = 4000$，$R = 0.1$，$C = 0.6$）
Fig. 15　The distribution of non-zeros ($N = 4000$, $R = 0.1$, $C = 0.6$).

例として $N = 4000$，$R = 0.1$，$C = 0.6$ の条件を与えて生成した行列に対して 16 プロセスを使って行列積計算をする場合に，各実ランクに割り当てられる非ゼロ要素数を**図 15** に示す．

この実験での最適化フェイズにおける最適化アルゴリズム実行間隔は，初期値を 16 とし，最適化アルゴリズム実行時に全体の待ち時間を集計した結果が以前の集計に比べて 25%以下の差であった場合には次回実行間隔を 2 倍に増やしている．

さらに，プロファイルフェイズでは，評価値は測定値が 3 回連続で 6.25%以下の誤差であった場合にはその平均値をとり，それ以外の場合には前回評価値と測定値の加重平均をとる．もし，新しい評価値が前回送信評価値と 50%以上の差があった場合には新しい評価値を送信する．

測定はそれぞれの条件につき 50 回行った．**表 3** に行列サイズ 4000，14000，非ゼロ要素傾きを 0.2，0.6，非ゼロ比率を 0.1，0.05，0.01 の各パラメータを組み合わせて作成した行列を入力行列としたときに，1 行の処理に要する平均ワークロード時間および平均通信時間を示す．

表中の Matrix は疎行列生成時に与えた条件を示しており，項内の N は行列サイズ，R

表 3　各入力行列に対する平均通信時間と性能比
Table 3　The average time of broadcast for each matrix.

Matrix		N=4000,R=0.1,C=0.2														
Type		Work Load Time					Broadcast Time					Total Time				
		Ori	Opt	Opt/Ori	Ide	Ide/Ori	Ori	Opt	Opt/Ori	Ide	Ide/Ori	Ori	Opt	Opt/Ori	Ide	Ide/Ori
Number	8	1261	1278	1.013	1278	1.013	428.4	100.9	0.235	79.25	0.185	1689	1379	0.816	1357	0.803
of	16	650.0	654.9	1.008	664.1	1.022	322.2	163.2	0.507	88.88	0.276	972.2	818.2	0.842	752.9	0.774
Process	32	297.7	300.0	1.008	307.1	1.032	347.5	245.3	0.706	220.2	0.634	645.1	545.3	0.845	527.4	0.817
	64	111.8	112.2	1.004	109.8	0.983	433.8	440.4	1.015	441.1	1.017	545.6	552.6	1.013	551.0	1.010
Matrix		N=4000,R=0.1,C=0.6														
Type		Work Load Time					Broadcast Time					Total Time				
		Ori	Opt	Opt/Ori	Ide	Ide/Ori	Ori	Opt	Opt/Ori	Ide	Ide/Ori	Ori	Opt	Opt/Ori	Ide	Ide/Ori
Number	8	1253	1261	1.007	1278	1.020	216.1	151.0	0.699	80.38	0.372	1469	1412	0.962	1358	0.925
of	16	634.5	650.9	1.026	649.9	1.024	196.2	132.9	0.677	111.1	0.566	830.8	783.9	0.944	761.0	0.916
Process	32	299.7	306.5	1.023	316.4	1.056	230.3	173.1	0.752	127.8	0.555	530.0	479.7	0.905	444.2	0.838
	64	103.4	105.8	1.023	101.1	0.977	332.1	334.8	1.008	337.6	1.017	435.5	440.6	1.012	438.6	1.007
Matrix		N=4000,R=0.05,C=0.2														
Type		Work Load Time					Broadcast Time					Total Time				
		Ori	Opt	Opt/Ori	Ide	Ide/Ori	Ori	Opt	Opt/Ori	Ide	Ide/Ori	Ori	Opt	Opt/Ori	Ide	Ide/Ori
Number	8	674.6	666.4	0.988	682.5	1.012	276.9	71.25	0.257	80.62	0.291	951.5	737.6	0.775	763.1	0.802
of	16	311.6	318.6	1.022	326.1	1.047	274.2	168.8	0.615	131.4	0.479	585.8	487.3	0.832	457.6	0.781
Process	32	115.0	114.1	0.993	110.1	0.957	334.9	347.5	1.038	334.4	0.998	449.9	461.6	1.026	444.5	0.988
	64	43.36	43.83	1.011	42.17	0.973	455.1	457.5	1.005	458.4	1.007	498.4	501.3	1.006	500.5	1.004
Matrix		N=4000,R=0.05,C=0.6														
Type		Work Load Time					Broadcast Time					Total Time				
		Ori	Opt	Opt/Ori	Ide	Ide/Ori	Ori	Opt	Opt/Ori	Ide	Ide/Ori	Ori	Opt	Opt/Ori	Ide	Ide/Ori
Number	8	665.5	688.9	1.035	678.9	1.020	161.4	78.38	0.486	91.38	0.566	826.9	767.2	0.928	770.2	0.932
of	16	317.4	327.8	1.033	338.1	1.065	194.0	137.4	0.708	107.2	0.553	511.4	465.2	0.910	445.2	0.871
Process	32	106.0	109.9	1.037	108.8	1.026	357.5	337.7	0.945	356.8	0.998	463.5	447.7	0.966	465.6	1.005
	64	41.55	41.77	1.005	41.44	0.997	457.8	464.0	1.014	464.5	1.015	499.4	505.8	1.013	506.0	1.013
Matrix		N=4000,R=0.01,C=0.2														
Type		Work Load Time					Broadcast Time					Total Time				
		Ori	Opt	Opt/Ori	Ide	Ide/Ori	Ori	Opt	Opt/Ori	Ide	Ide/Ori	Ori	Opt	Opt/Ori	Ide	Ide/Ori
Number	8	105.5	99.75	0.945	98.50	0.934	184.6	169.5	0.918	175.4	0.950	290.1	269.2	0.928	273.9	0.944
of	16	43.94	43.62	0.993	42.69	0.972	273.3	269.2	0.985	267.8	0.980	317.2	312.9	0.986	310.5	0.979
Process	32	21.12	21.59	1.022	21.44	1.015	325.6	342.8	1.053	338.8	1.040	346.8	364.4	1.051	360.2	1.039
	64	10.61	10.69	1.007	10.78	1.016	372.9	380.8	1.021	376.5	1.010	383.5	391.5	1.021	387.3	1.010
Matrix		N=4000,R=0.01,C=0.6														
Type		Work Load Time					Broadcast Time					Total Time				
		Ori	Opt	Opt/Ori	Ide	Ide/Ori	Ori	Opt	Opt/Ori	Ide	Ide/Ori	Ori	Opt	Opt/Ori	Ide	Ide/Ori
Number	8	99.75	101.9	1.021	98.62	0.989	217.0	214.1	0.987	215.4	0.993	316.8	316.0	0.998	314.0	0.991
of	16	45.00	45.00	1.000	45.25	1.006	302.9	304.1	1.004	312.7	1.032	347.9	349.1	1.003	357.9	1.029
Process	32	22.31	22.53	1.010	22.59	1.013	365.0	372.9	1.022	372.2	1.020	387.3	395.4	1.021	394.8	1.019
	64	11.42	11.55	1.011	11.41	0.999	420.6	426.2	1.013	425.8	1.012	432.0	437.7	1.013	437.2	1.012

Matrix		N=14000,R=0.1,C=0.2														
Type		Work Load Time					Broadcast Time					Total Time				
		Ori	Opt	Opt/Ori	Ide	Ide/Ori	Ori	Opt	Opt/Ori	Ide	Ide/Ori	Ori	Opt	Opt/Ori	Ide	Ide/Ori
Number	8	14700	14780	1.005	14760	1.004	4467	532.2	0.119	256.1	0.057	19170	15310	0.799	15020	0.783
of	16	7424	7448	1.003	7473	1.007	2683	754.3	0.281	269.2	0.100	10110	8202	0.811	7743	0.766
Process	32	3748	3780	1.009	3823	1.020	1743	741.9	0.426	283.7	0.163	5491	4522	0.824	4106	0.748
	64	1889	1900	1.006	1946	1.030	1477	941.9	0.638	649.9	0.440	3366	2842	0.844	2596	0.771
Matrix		N=14000,R=0.1,C=0.6														
Type		Work Load Time					Broadcast Time					Total Time				
		Ori	Opt	Opt/Ori	Ide	Ide/Ori	Ori	Opt	Opt/Ori	Ide	Ide/Ori	Ori	Opt	Opt/Ori	Ide	Ide/Ori
Number	8	14640	14750	1.008	14680	1.003	1935	773.4	0.400	257.9	0.133	16570	15520	0.937	14930	0.901
of	16	7377	7416	1.005	7455	1.011	1278	836.9	0.655	275.1	0.215	8655	8253	0.954	7730	0.893
Process	32	3752	3780	1.007	3839	1.023	1018	600.1	0.590	313.9	0.308	4770	4380	0.918	4152	0.871
	64	1887	1898	1.006	1936	1.026	970.4	707.6	0.729	499.6	0.515	2857	2606	0.912	2435	0.852
Matrix		N=14000,R=0.05,C=0.2														
Type		Work Load Time					Broadcast Time					Total Time				
		Ori	Opt	Opt/Ori	Ide	Ide/Ori	Ori	Opt	Opt/Ori	Ide	Ide/Ori	Ori	Opt	Opt/Ori	Ide	Ide/Ori
Number	8	7501	7584	1.011	7527	1.003	2382	364.0	0.153	251.1	0.105	9883	7948	0.804	7778	0.787
of	16	3851	3840	0.997	3873	1.006	1601	504.1	0.315	272.6	0.170	5452	4344	0.797	4145	0.760
Process	32	1931	1958	1.014	2007	1.039	1248	682.4	0.547	392.1	0.314	3179	2640	0.830	2399	0.754
	64	970.0	973.7	1.004	976.2	1.006	1363	1072	0.786	1070	0.785	2333	2046	0.877	2046	0.877
Matrix		N=14000,R=0.05,C=0.6														
Type		Work Load Time					Broadcast Time					Total Time				
		Ori	Opt	Opt/Ori	Ide	Ide/Ori	Ori	Opt	Opt/Ori	Ide	Ide/Ori	Ori	Opt	Opt/Ori	Ide	Ide/Ori
Number	8	7518	7564	1.006	7546	1.004	1137	557.1	0.490	256.9	0.226	8655	8121	0.938	7802	0.902
of	16	3819	3863	1.012	3910	1.024	914.1	434.8	0.476	280.7	0.307	4733	4298	0.908	4190	0.885
Process	32	1940	1965	1.013	1993	1.027	837.9	584.3	0.697	412.2	0.492	2778	2550	0.918	2405	0.866
	64	980.4	978.6	0.998	977.1	0.997	1208	1142	0.946	1168	0.967	2188	2121	0.969	2145	0.981
Matrix		N=14000,R=0.01,C=0.2														
Type		Work Load Time					Broadcast Time					Total Time				
		Ori	Opt	Opt/Ori	Ide	Ide/Ori	Ori	Opt	Opt/Ori	Ide	Ide/Ori	Ori	Opt	Opt/Ori	Ide	Ide/Ori
Number	8	1697	1739	1.025	1730	1.019	854.0	284.8	0.333	260.5	0.305	2551	2024	0.793	1990	0.780
of	16	855.0	865.6	1.012	854.7	1.000	941.2	693.5	0.737	698.9	0.743	1796	1559	0.868	1554	0.865
Process	32	423.4	425.9	1.006	425.7	1.005	1275	1256	0.985	1267	0.994	1699	1681	0.990	1693	0.997
	64	198.7	199.3	1.003	193.7	0.975	1616	1615	1.000	1619	1.002	1814	1814	1.000	1813	0.999
Matrix		N=14000,R=0.01,C=0.6														
Type		Work Load Time					Broadcast Time					Total Time				
		Ori	Opt	Opt/Ori	Ide	Ide/Ori	Ori	Opt	Opt/Ori	Ide	Ide/Ori	Ori	Opt	Opt/Ori	Ide	Ide/Ori
Number	8	1709	1762	1.031	1738	1.017	522.6	249.1	0.477	295.4	0.565	2232	2011	0.901	2033	0.911
of	16	850.4	844.8	0.993	850.9	1.001	779.6	716.2	0.919	731.6	0.938	1630	1561	0.958	1582	0.971
Process	32	434.4	438.2	1.009	433.2	0.997	1299	1274	0.981	1294	0.996	1734	1713	0.988	1727	0.996
	64	197.6	199.5	1.010	196.4	0.994	1664	1672	1.005	1656	0.995	1861	1872	1.006	1852	0.995

Ori:Original Opt:Optimized Ide:Ideal　　Time = (us)

は非ゼロ比率，C は非ゼロ要素傾きの各条件に対応している．Type は測定対象を示しており，Work Load Time はワークロードの時間，Broadcast Time はトランスロード時間およびアイドル時間の和，Total Time は Load Time と Broadcast Time の和が測定対象であることを示している．また，Ori は最適化を行わなかった場合の時間，Opt は最適化を行った場合の時間，Ide は最適な仮想ランク割当てを行った場合の時間を示しており，単位はマイクロ秒，Opt/Ori は Ori に対する Opt の比率，Ide/Ori は Ori に対する Ide の比率を示している．

図 16 に行列サイズの変化に対する最適化の効果の変化を示す．

全体性能では最大 20%弱の改善が見られる．ルートプロセスによる行の抽出に必要な時間はプロセス数と関係なく決まる．それに対して，ルート以外のプロセスによる行列積の計算時間は処理する非ゼロ要素数に比例するのでプロセス数に反比例する．そのため，プロセス数が大きくなるにつれ，行の抽出時間と行列積計算時間が近づいていくため改善効果は低下する．$N = 4000$ を 64 プロセスで実行した場合では，すべての行列積計算が行抽出時間より早く終わるため最適化の効果は見られなくなる．$N = 14000$ のときに Bcast 時間の改善効果が低下しても全体性能の改善率は変わっていないが，これはワークロードに対する通信時間の割合が高くなっているためである．

図 17 に非ゼロ要素率傾きの変化に対する最適化の効果の変化を示す．

$C = 0.6$ のときは全体性能では最大 10%弱の改善が見られる．また，$C = 0.2$ のときに比べ $C = 0.6$ のときはプロセス数が増えても改善率の低下が少なくなっているのは，全体の非ゼロ要素数が同じであれば非ゼロ要素率傾きが少ない，すなわち C が 1 に近いほど各プロセスに分配される非ゼロ要素数は平均化されるため，分配された非ゼロ要素数が最も少

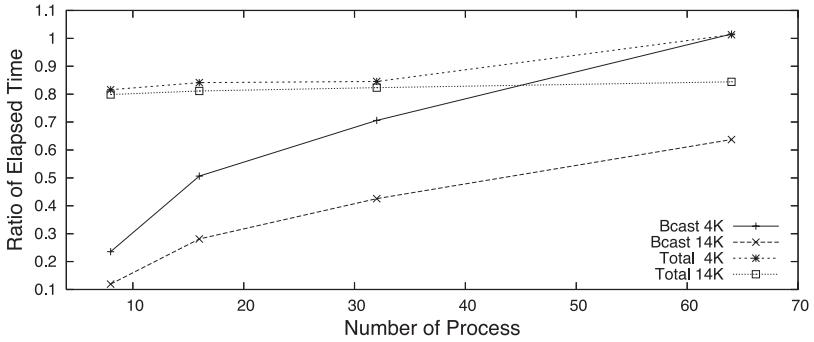

図 16　行列サイズごとの性能比
Fig. 16　Elaps ratio of optimization for each matrix size.

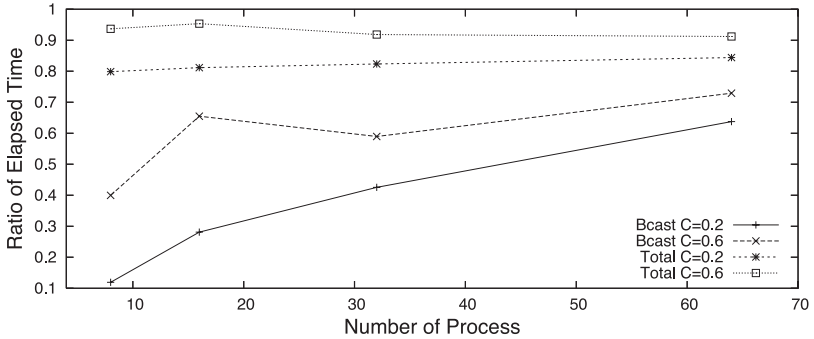

図 17　各列の非ゼロ率傾きごとの性能比（$N = 14000$，$R = 0.1$）
Fig. 17　Elaps ratio of optimization for each slope of column ratio.

ないプロセスどうしを比較した場合，C が 1 に近いほど分配される非ゼロ要素数は増加し行列積計算時間は増加することから，$C = 0.6$ のほうが，プロセス数が増えても行列積計算時間が行抽出時間以下になるプロセスの数が増えないことが理由となっている．

図 18 に非ゼロ要素率の変化に対する最適化の効果の変化を示す．

64 プロセスでの実行時には非ゼロ要素率 5%の時に約 10%の改善効果が見られるが，1%のときにはほとんど改善効果は見られていない．非ゼロ要素率 5%と 10%ではプロセス数が 16 以下のときに Bcast 時間の改善率がほとんど変わっていないのは，負荷不均衡のパターンが改善率を決定する主要因であるためで，プロセス数が 32 を超えていくと行抽出より短時間に終わる行列積計算を行うプロセスの数がそれぞれの非ゼロ要素率によって異なってくるため，改善率が低下していく．非ゼロ要素率 1%のときは，8 プロセス時においてもいくつかのプロセスでの行列積計算時間は行抽出時間より早くなっているため，Bcast 時間の改善率が他の非ゼロ要素率に比べて低下している．32 プロセスではすべての行列積計算時間が行抽出時間より早くなっているために改善効果は見られない．

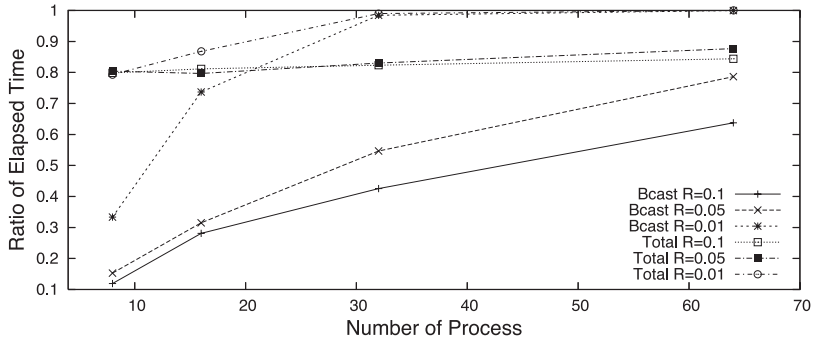

図 18　非ゼロ率ごとの性能比（$N = 14000$，$C = 0.2$）
Fig. 18　Elaps ratio of optimization for each slope of column ratio.

表 3 より疎行列乗算を理想的な配置で実行した場合と動的最適化の元で実行した場合の平均通信時間を比較すると，ほとんどの場合で理想的な配置が動的最適化を上回る，しかしながら $N = 4000$，$R = 0.05$，$C = 6$ の疎行列を 8 プロセスで実行したとき等の，プロセス数が少なくかつ乗算を行うプロセス間における演算量の差が小さい場合には，動的最適化が理想的な配置を上回る場合がある．

プロセス数が大きい場合には，再構成フェイズにおいて 1 組のみの仮想ランク交換を行う本実験における方法では理想的な配置に近づくまでに長い時間を要すること，および 2 点間通信において送信側と受信側の同期をとらない突き放し通信が行われているため，唯一受信を行わないルートプロセスにおける送信処理のみが，全プロセスからのプロファイルを獲得したときには大きく進んでいるため，有効な最適化を行える時期が遅くなることが，動的最適化が不利となる理由としてあげられる．

それに対して，プロセス数が少ない場合には，動的最適化は短期間にプロファイル結果を得られるため早期に有効な仮想ランク入れ替えが可能になり，また理想的な配置となるまで

の手数も少なくなる．さらにプロセス間の演算量差が小さい場合には，送信側プロセスにおけるトランスロードが相対的に大きくなり，理想的な仮想ランク配置を行った場合においても受信側のプロセスに待ち時間が発生するが，このトランスロードを動的最適化によって分散させることでより効率的にブロードキャスト通信を行いうることが，動的最適化が理想的な配置を上回る理由としてあげられる．

それ以外にも，突き放し通信による送信メッセージが受信側のメモリに蓄積される影響や，動的最適化による経路変更によってネットワーク上で送信メッセージが衝突する場合が発生することや，経路変更によってメモリバス上で発生していた受信メッセージとプロセッサによるメモリアクセスとの衝突が回避されること，等も性能に影響を与えていると考えられるが，これらの解析には 2 点間通信における MPI 通信ライブラリやプロセッサの挙動を調査する必要があると考えられる．

6. ま と め

本論文ではブロードキャスト通信に動的最適化を適用することによって，負荷不均衡を緩和してブロードキャスト通信を含めたプログラムの実行時間を短縮する手法を提案した．ブロードキャスト通信の待ち時間を減少させるために，仮想ランクを用いて集団通信内でのランク割当てを可変にし，仮想ランク割当てを実行中に動的に行うことによりブロードキャスト通信の最適化を自動化する手法について述べた．本手法を実装した MPI_Bcast 関数を用いて行った実験では，本手法の最大効果を測定するためのベンチマークから平均通信時間で最大 90%以上の改善がなされることが分かり，疎行列乗算ベンチマークからは平均通信時間で最大 88%の改善がなされ，この改善率は負荷バランスのパターン依存が主であることが示された．

本論文における実験では，外乱がある場合にも統計的には本動的最適化手法が効果的であることを示しているが，さらに進めて外乱による全体通信性能の低下に対して動的最適化がどのような影響を与えるのか調査を行っていきたい．また，外乱要素を排除可能なプロファイル技術を得るための研究も進めていく必要があると考えている．さらに，二項木以外のアルゴリズムに対する評価や MPI 標準によって定義されている他の集団通信に対する評価についても行っていきたい．

謝辞　本研究は文部科学省 “次世代 IT 基盤構築のための研究開発，研究開発領域「将来のスーパーコンピューティングのための要素技術の研究開発” による．また，本研究では理化学研究所の “RIKEN Super Combined Cluster（RSCC)” を使用した．

参 考 文 献

1) Mathematics and Computer Science Division Argone National Laboratory: *MPICH2 User's Guide*.
2) OpenMPI. http://www.open-mpi.org
3) Vadhiyar, S.S., Fagg, G.E. and Dongarra, J.: Automatically Tuned Collective Communications, *ACM/IEEE conference on Supercomputing*, p.3, ACM (2000).
4) Tipparaju, V. and Nieplocha, J.: Optimizing All-to-All Collective Communication by Exploiting Concurrency in Modern Networks, *Proc. ACM/IEEE SC 2005 Conference*, pp.46–46, ACM (2005).
5) Faraj, A. and Yuan, X.: Automatic generation and tuning of MPI collective communication routines, *Proc. 19th annual international conference on Supercomputing*, pp.393–402, ACM (2005).
6) ABC lib. http://www.abc-lib.org
7) Faraj, A., Yuan, X. and Lowenthal, D.: STAR-MPI: Self Tuned Adaptive Routines for MPI Collective Communications, *Proc. 20th annual international conference on Supercomputing*, pp.199–208, ACM (2006).
8) Buntinas, D., Panda, D.K. and Brightwell, R.: Application-Bypass Broadcast in MPICH over GM, *Proc. Cluster Computing and Grid Conference* (*CCGrid*), pp.2–9, ACM (2003).
9) Message Passing Interface Forum: *MPI: A Message-Passing Interface Standard*.
10) Golub, G.H. and Loan, C.F.V.: *Matrix Computations*, 3rd edition, Johns Hopkins University Press, Baltimore, Maryland (1996).

(平成 20 年 5 月 9 日受付)
(平成 20 年 8 月 28 日採録)

曽我　武史

1993 年京都大学工学部電子工学科卒業．同年富士通（株）入社，主にスーパコンピュータ開発に従事．2003 年（財）福岡県産業科学技術振興財団研究員．2008 年（財）福岡市先端科学技術研究所特任研究員．

栗原　康志

2007 年九州大学工学部電気情報工学科卒業．2009 年九州大学大学院システム情報科学府情報理学専攻修了予定．

南里　豪志（正会員）

1995 年九州大学システム情報科学研究科修士課程修了．1996 年 6 月九州大学大型計算機センター助手．2001 年 1 月九州大学情報基盤センター助教授．2007 年 4 月より九州大学情報基盤研究開発センター准教授．計算機科学，特に並列計算システムに関する研究に従事．博士（情報科学）．

黒川　原佳（正会員）

2002 年北陸先端科学技術大学院大学博士後期課程修了．同年から理化学研究所に勤務．博士（情報科学）．主に並列 CFD，並列分散処理，システム評価に興味を持つ．スーパコンピュータ・システムの設計，運用管理に従事．IEEE.CS，日本機械学会各会員．

村上　和彰（正会員）

1960 年生．1984 年京都大学大学院工学研究科情報工学専攻修士課程修了．同年富士通（株）入社．汎用大型計算機の研究開発に従事．1987 年九州大学助手．1994 年九州大学助教授．現在，九州大学大学院システム情報科学研究院情報理学部門教授，情報基盤研究開発センター長，情報統括本部長．計算機アーキテクチャ，並列処理，システム LSI 設計技術，等に関する研究に従事．工学博士．1991 年情報処理学会研究賞，1992 年情報処理学会論文賞，1997 年坂井記念特別賞，2000 年日経 BP 社 IP アワード，2000 年情報処理学会創立 40 周年記念論文賞，2002 年電子情報通信学会業績賞をそれぞれ受賞．