情報処理学会論文誌　コンピューティングシステム　Vol.5　No.5　152–162 (Oct. 2012)

MPI_Allreduceの「京」上での実装と評価

松本 幸[1,a)]　安達 知也[1]　住元 真司[1]　南里 豪志[2]　曽我 武史[2]　宇野 篤也[3]
黒川 原佳[3]　庄司 文由[3]　横川 三津夫[3]

受付日 2012年3月28日, 採録日 2012年6月9日

概要：本論文では，82,944 台の計算ノードを Tofu インターコネクトと呼ばれる 6 次元の直接網で結合した「京」における MPI 集団通信の高速化について述べる．従来の MPI ライブラリには，トポロジを考慮したアルゴリズムが存在しないため，「京」のような直接網において性能を出すことができない．そのため，Trinaryx3 と呼ばれる Allreduce を設計し，「京」向けの MPI ライブラリに実装した．Trinaryx3 アルゴリズムは，トーラス向けに最適化されており，「京」の特長の 1 つである複数 RDMA エンジンを活用することができる．実装を評価した結果，既存のトポロジを考慮していないアルゴリズムと比較して，5 倍のバンド幅の向上を確認した．

キーワード：京コンピュータ，MPI 集団通信，MPI_Allreduce，トーラスネットワーク

Implementation and Evaluation of MPI_Allreduce on the K Computer

Yuki Matsumoto[1,a)]　Tomoya Adachi[1]　Shinji Sumimoto[1]　Takeshi Nanri[2]
Takeshi Soga[2]　Atsuya Uno[3]　Motoyoshi Kurokawa[3]　Fumiyoshi Shoji[3]
Mitsuo Yokokawa[3]

Received: March 28, 2012, Accepted: June 9, 2012

Abstract: This paper reports a method of speeding up MPI collective communication on the K computer, which consists of 82,944 computing nodes connected by a 6D direct network, named Tofu interconnect. Existing MPI libraries, however, do not have topology-aware algorithms which perform well on such a direct network. Thus, an Allreduce collective algorithm, named Trinaryx3, is designed and implemented in the MPI library for the K computer. The algorithm is optimized for a torus network and enables utilizing multiple RDMA engines, one of the strengths of the K computer. The evaluation results show the new implementation achieves five times higher bandwidth than existing one.

Keywords: K computer, MPI collective communication, MPI_Allreduce, torus network

1. はじめに

分散メモリ型並列計算機における並列計算では，Message Passing Interface（MPI）ライブラリを用いたプログラミングが一般的である．MPI ライブラリの例としては，MPICH2 [12] や Open MPI [7] がある．我々が開発しているスーパーコンピュータ「京」[*1]は，82,944 台の計算ノードを直接網で結合した分散メモリ型並列計算機であり，Open MPI をベースとした「京」向けの MPI ライブラリを提供している．

MPI ライブラリのインタフェースは，MPI 規格 [10], [11] によって定められている．MPI 規格では，1 対 1 通信のほかに，バリア同期や全対全通信といった，集団通信も定義

[1] 富士通株式会社
Fujitsu Ltd., Kawasaki, Kanagawa 211–8588, Japan
[2] 九州大学
Kyushu University, Fukuoka 812–8581, Japan
[3] 理化学研究所
RIKEN, AICS, Kobe, Hyogo 650–0047, Japan
[a)] yuki.matsumoto@jp.fujitsu.com

[*1] 「京」は文部科学省が推進し，理化学研究所と富士通が共同開発した次世代スーパーコンピュータの愛称です．

されている．集団通信は，サーバ構成やネットワーク構成によって最適な通信アルゴリズムが異なる．直接網で結合されたシステムでは，ノード間距離により通信性能が変わるほか，経由する通信路の重なりによって輻輳が起きる．そのため，それらを考慮した直接網向けのアルゴリズムが提案されている [3], [4], [8], [17].

しかしながら，直接網向けのアルゴリズムを実装しているMPIライブラリは非常に少ない．これは，一般に普及しているPCクラスタの大部分が間接網で結合されているためと推測される．既存のトポロジを考慮していないアルゴリズムは，プロセスに対してトポロジ情報に関係なく一次元的に割り振られたランク番号ベースの実装がほとんどであり，「京」のような直接網で結合されたシステムにおいては，通信の衝突が起きやすくバンド幅を活かすことができない．

また，「京」は複数のRDMAエンジンを搭載し，複数方向に同時通信が可能となっているが，MPIライブラリのベースとなっているOpen MPIの集団通信部分においては，複数方向への同時通信を制御することができないため，設計したとおりの性能とならない可能性がある．

性能を引き出すためには，直接網向け集団通信アルゴリズムの複数同時通信の制御を可能とした実装が必須といえる．特に，集団通信の中で，Allreduceは，一般的な並列アプリケーションにおいて高い頻度で利用され，総実行時間に占めるMPI_Allreduce関数の実行時間の割合も高いことが知られている [13]．MPI_Allreduceを高速化することは重要であると考えられる．

MPI_Allreduceの実装方法として，ReduceとBcastを組み合わせる方法がある．そこで，我々は，まずトーラス向けのBcastアルゴリズムであるTrinaryx3 Bcastを提案し，それを利用したAllreduceアルゴリズムTrinaryx3 Allreduceを提案する．そして，「京」向けのMPIライブラリに実装し，その評価を行う．Trinaryx3 Bcastは，複数の通信路を同時に用いるパイプライン転送アルゴリズムで，使用するリンクに重なりがないため，衝突が起きないという特長がある．「京」に搭載されたTofuインターコネクト [2] の特長である複数のRDMAエンジンを活用することで，高バンド幅が実現できることを示す．そして，その実装を「京」上でOpen MPI由来の既存のトポロジを考慮していないアルゴリズム実装と性能比較することにより，トーラスを意識したTrinaryx3 Allreduceの有効性を示す．

2章ではMPI_Allreduceの概要と既存のAllreduceアルゴリズムを紹介する．次に，3章で「京」のネットワークアーキテクチャについて説明する．4章ではTrinaryx3 Bcastとその Trinaryx3 Allreduceへの応用について述べる．5章で「京」上でのTrinaryx3 Allreduceの実装について説明し，続く6章で実装の評価を行う．7章で関連研究を紹介し，最後に，8章で結論を述べる．

2. MPI_Allreduceの概要と既存アルゴリズム

2.1 MPI_Allreduceの概要と要件

Allreduceは，全プロセスが持っているデータ列を集め，演算を行い，その結果を全プロセスが共有する集団通信である．以下，本論文では，プロセス数をP，データ数をN，演算を$\circ$で表す．通信前にプロセスiが持っているデータを$a_{i,j}$（$0 \leq j < N$）とすると，通信後は全プロセスが$a_{0,j} \circ a_{1,j} \circ \cdots \circ a_{P-1,j}$（$0 \leq j < N$）を得ることになる．

我々は，可換な演算を用いたMPI_Allreduceをターゲットとする．MPI規格では，加算や乗算，最大値などの演算があらかじめ定義されているほか，ユーザが自分で演算を定義して使うこともできる．このうち，定義済みの演算は，結合的かつ可換であるため，演算順序の最適化を行うことができる．

ただし，MPI規格では，以下の2点の制約が課されている．全プロセスで完全に結果が一致することと，同じ入力に対してはつねに同じ出力が得られることである．たとえば浮動小数点演算は，計算誤差が発生しうるため，厳密には結合的でないことが知られている．プロセスごとに，あるいは関数の実行ごとに，違う順序で演算を行うようなアルゴリズムは，これらの制約を満たすことができない．

2.2 既存アルゴリズム

本節では，MPIライブラリに実装されているアルゴリズムを3種類紹介し，数万プロセス規模の超並列，長メッセージでの挙動について議論する．

2.2.1 Reduce + Bcast

最もナイーブなアルゴリズムは，1つのプロセスに演算結果を集約したのち（Reduce），その結果を全体にbroadcast（Bcast）するものである [4]．演算結果を集約するプロセスをrootプロセスと呼ぶ．4プロセスでの実行例を**図 1** に示す．

通信は，プロセスをグラフの頂点と見なして作った全域木上で行う．Reduceではleafからrootに向かって通信し，Bcastではrootからleafに向かって通信する．木の構成手法としては，rootプロセスと残りのプロセスとを1対1で直接結ぶ方法や，二分木（binary tree）や二項木（binomial tree）により通信ステップ数を対数オーダで減らす方法が知られている．また，直接網上でのプロセス間の結合を意識した木の構成方法も研究されている [8].

このアルゴリズムは，超並列においても良好なスケーラビリティを示す．1つのプロセスにおける送受信量は，プロセス数にかかわらず一定である．また，レイテンシは木の高さに比例して増えてしまうが，パイプライン転送により，長メッセージでのスループット低下を防ぐことができる．よって，我々はこのアルゴリズムを「京」向けのMPI

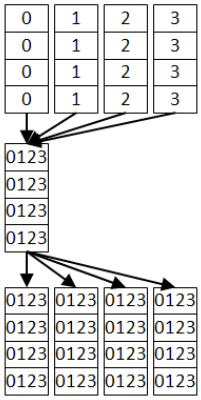

図 1　Reduce + Bcast
Fig. 1　Reduce + Bcast.

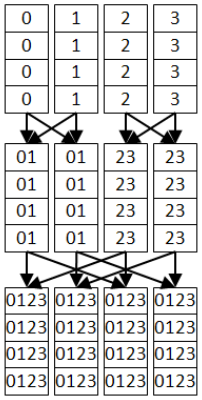

図 2　Recursive Doubling
Fig. 2　Recursive Doubling.

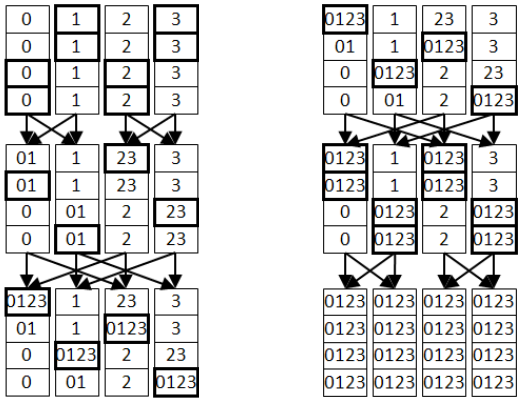

図 3　Reduce_scatter + Allgather
Fig. 3　Reduce_scatter + Allgather.

ライブラリに実装する．

2.2.2　Recursive Doubling

Recursive Doubling は，ネットワークの双方向性を利用し，プロセス数の対数のステップ数で通信を完了させるアルゴリズムである（**図 2**）．i（$0 \leq i < \log P$）ステップ目では，2^i 離れたプロセスと通信する．プロセス数が 2 べキでない場合でも，定数ステップ増やすことで対応する方法が知られている [14]．

このアルゴリズムは，各ステップにおいて，すべてのプロセスがメッセージ全体のやりとりを行っている．ステップ数がプロセス数の対数であるため，1 つのプロセスが通信する量は，メッセージサイズ $\times \log P$ となる．これは，超並列でスループットの低下が避けられないことを示している．

2.2.3　Reduce_scatter + Allgather

Recursive Doubling は，前述のとおり長メッセージには不向きであり，また，複数のプロセスが同じデータに対して演算しているという意味で，非効率な面がある．それに着目した長メッセージ向けのアルゴリズムが，Reduce_scatter + Allgather [16] である（Rabenseifner のアルゴリズム [14] とも呼ばれる）．

まず，各プロセスが演算結果を $1/P$ ずつ持つように，演算と通信を行う（Reduce_scatter）．これには，やりとりするメッセージサイズを半分に減らしていく方法（recursive halving）や，$1/P$ ずつメッセージを区切って隣のノードに転送していく方法（ring）などがある．次に，各プロセスに散らばった演算結果を全プロセスで共有する（Allgather）．これにも，Reduce_scatter の逆で，やりとりするメッセージサイズを倍々に増やしていく方法（recursive doubling）や，順々に隣のノードにメッセージを転送していく方法（ring）など，いくつかの手法がある．recursive halving と recursive doubling による例を**図 3** に示す．太枠で示した部分が，そのステップで通信する対象である．

このアルゴリズムでは，1 つのプロセスにおける総通信量はプロセス数にかかわらず一定となる．しかしながら，

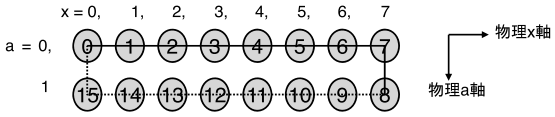

図 4　X 軸と a 軸の仮想三次元の X 軸へのマッピング
Fig. 4　Mapping from X-axis and a-axis to logical X-axis.

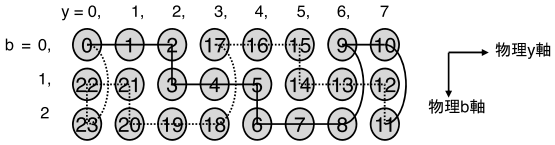

図 5　Y 軸と b 軸の仮想三次元の Y 軸へのマッピング
Fig. 5　Mapping from Y-axis and b-axis to logical Y-axis.

各ステップで通信の対象となるメッセージのサイズはプロセス数に依存する．同じ長さのメッセージを送る場合，プロセス数が増えれば増えるほど，Reduce_scatter 後に各プロセスが持つデータ量は小さくなる．一般に短メッセージでは，通信レイテンシの影響でバンド幅を生かしきれない．超並列においては，メッセージサイズがきわめて巨大な，限られた範囲でしか有効でないと考えられる．

3.　京のネットワークアーキテクチャ

本章では，「京」で使用されている Tofu インターコネクト [2] について述べる．「京」は 82,944 台の計算ノードで構成されており，そのネットワークは，三次元トーラスを拡張した，X，Y，Z，a，b，c の六次元からなるメッシュ/トーラス構造である．a，b，c 軸が $2 \times 3 \times 2$ の 12 ノードが，共通の X，Y，Z 座標を持つノードグループとなる．ノードグループを組み合わせることで六次元メッシュ/トーラスとなる．

利用者はジョブを一，二，三次元トーラスに割り付けて実行することが可能である．六次元メッシュ/トーラスを利用して，システムを一部切り出してもトーラスとなる．さらに，通常のトーラスでは，ノード故障時にトーラスを維持できないが，このインターコネクトでは，故障ノード

を回避するルーティングを行うことで，故障のノードがある場合もトーラスを使用可能という特徴がある．六次元メッシュ/トーラスを三次元トーラスに割り付ける例を示す．**図 4** および**図 5** は，X と a，Y と b，Z と c をそれぞれ 1 つの軸として，仮想的に三次元トーラスにプロセスを割り付けた例である．図中の円に記載された数字は，仮想三次元における割り付けた軸の座標を示している．図 4 は X 軸と a 軸への割付け方法を示している．Z 軸と c 軸の割付けは X 軸と a 軸の割付けと同様である．図 5 は Y 軸と b 軸への割付方法を示している．このように三次元の各軸がトーラスとなるように接続される．仮想三次元の各軸の番号は，その軸における座標を表す．仮想三次元トーラスにおける隣接ノードは，六次元においても隣接ノードであることを保証している．以降，仮想三次元トーラスを前提として議論を進めていく．

各ノード内には，インターコネクトコントローラが存在し，4 個の RDMA エンジンが搭載されている [19]．このインターコネクトコントローラでは，4 個の RDMA エンジンを使用して，最大四方向同時に送受信が可能となる．4 方向同時通信のピークバンド幅は 20 GB/sec [19] である．

4. 三次元トーラス向けアルゴリズム Trinaryx3 Allreduce

2 章で述べたとおり，「京」のような超並列環境では，Reduce + Bcast アルゴリズムを使用したパイプライン転送による MPI_Allreduce の実装が有効と考えられる．本章では，まず三次元トーラス向け Bcast アルゴリズムである Trinaryx3 Bcast を提案し，それを Reduce と Bcast それぞれに活用した Trinaryx3 Allreduce について述べる．

4.1 Trinaryx3 Bcast

Trinaryx3 Bcast は，三次元トーラスネットワーク上に 3 つの辺素（共通の辺を持たない）な全域木を構成し，3 経路を用いた同時通信を行う Bcast アルゴリズムである．トーラス軸を意識した木の構成により，通信はすべて隣接プロセス間で行われ，レイテンシが小さい．また，木は互いに辺素であるので，通信の衝突が起きないという特長がある．

まず，簡単のため二次元トーラス上での木の構成を**図 6** に示す．tree 0 では，まず root プロセスから x 軸方向にあるプロセスを子のノードとする．次に，新しく接続されたプロセスから y 軸方向にあるプロセスを子のノードとする．すると，x 座標が root プロセスと一致するプロセスだけ接続されていないので，それらのプロセスに x 軸方向で隣接するプロセスから接続することで補完する．x 軸と y 軸を入れ替えて同様に構成した木が tree 1 である．これらの木は，互いに辺素な全域木となっている．

この構成方法を三次元トーラスに拡張したものが Trinaryx3 である．三次元トーラス上で，x 軸方向，y 軸方向，

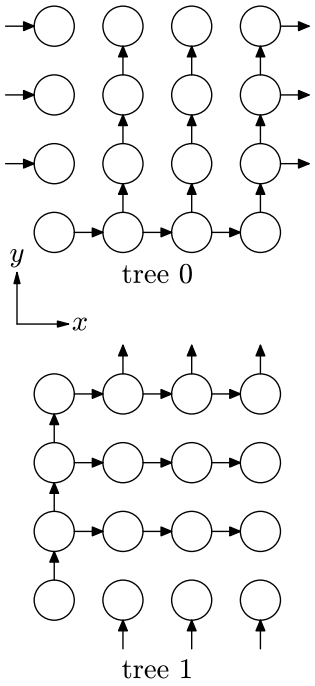

図 6　二次元における木の構成
Fig. 6　Construction of two trees for Trinaryx3 Bcast as 2D.

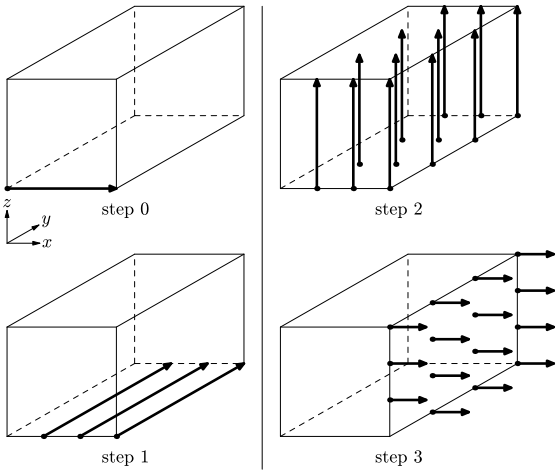

図 7　三次元トーラスにおける木の構成
Fig. 7　Construction of a tree for Trinaryx3 Bcast.

z 軸方向の順に接続して木を構成する例を**図 7** に示す．接続する軸の順番を巡回させて同様に構成すると，3 つの辺素な全域木が得られる．

通信は，メッセージを 3 分割し，それぞれの全域木を用いて転送する．通信資源が複数ある場合，これらの通信はまったく独立に行うことができる．よって，見かけ上のスループットは，1 経路を用いた場合の 3 倍になることが期待される．

4.2 Allreduce への適用

Trinaryx3 Bcast は，通信方向を逆にすることで，Reduce アルゴリズムとしても適用可能である．2 つを組み合わせれば，衝突のない，高バンド幅の Allreduce アルゴリズム

が得られる．

ただし，Reduce への適用では，2 章で述べたとおり，演算順序に気をつける必要がある．それぞれの全域木において，複数のプロセスからメッセージを受け取って演算するプロセスが存在する．複数プロセスからメッセージを受信する場合，一般にメッセージの到着順序は保証されない．そのため，メッセージが到着した順に演算を行う方針では，演算順序に非決定性が生じてしまう．あらかじめ静的に演算順序を決めておき，仮に後で演算するメッセージが先に来てしまった場合は，即座に演算せずに，先に演算するメッセージの到着を待つ設計とする．

演算順序の設定は，プロセスの全域木における高さ，すなわち最も遠い leaf プロセスからの距離に基づいて行う．全プロセスがいっせいに Reduce を始めるという前提で考えると，高さの低いプロセスほど，最初にメッセージを送信するまでの時間が短いと考えられる．よって，複数のプロセスからメッセージを受信し演算する場合，高さの低いプロセスのメッセージから演算することにしておくと，無駄な待ちが発生する可能性が低くなる．

なお，Bcast にも，この考え方を応用することができる．通信レイテンシを抑えるためには，距離の遠い leaf プロセスへの転送を優先するのがよいと考えられる．よって，Bcast で複数プロセスに転送する場合の通信順序は，Reduce で求めた演算順序の逆順とする．

4.3 Trinaryx3 Allreduce の性能モデル

Trinaryx3 Allreduce をパイプライン転送する実装を行った際の，理想的な実行時間のモデルを考える．Trinaryx3 Allreduce は Reduce + Bcast で実装されているため，Reduce と Bcast それぞれの実行時間のモデルを考える．

以下のように変数を定義する．初めに，通信レイテンシを L_{mpi}，通信のバンド幅を BW_{comm}，パイプラインの最大段数を H とする．また，1 度のパイプラインで送信するセグメントの長さを SS，送信するメッセージ長を N，演算のスループットを BW_{comp} とする．

はじめにデータ分割を行わない Bcast の実行時間 T_{bc} は，式 (1) のように表すことができる．

$$\begin{aligned} T_{bc} = {} & H \times (L_{mpi} + SS \div BW_{comm}) \\ & + N \div SS \\ & \times \max(L_{mpi}, SS \div BW_{comm}) + L_{mpi} \end{aligned} \tag{1}$$

データ分割を行わない Reduce の実行時間 T_{rd} は，式 (2) のように表すことができる．ここで，通信の転送と通信の送出・演算は完全にオーバラップしていると仮定する．

$$\begin{aligned} T_{rd} = {} & H \times (L_{mpi} + SS \div BW_{comm} \\ & + SS \div BW_{comp}) \\ & + N \div SS \\ & \times \max(L_{mpi} + SS \div BW_{comp}, \\ & \quad SS \div BW_{comm}) \\ & + L_{mpi} \end{aligned} \tag{2}$$

Trinaryx3 Allreduce の実装では，送信するデータのセグメント分割数を 3 分割する．すなわち，メッセージの転送において，セグメント分割数 $N \div SS$ が 1/3 となる．しかし，メッセージの送出と演算はシングルスレッドで実行するため，1/3 にならない．このため，Trinaryx3 の Bcast と Reduce の実行時間は，式 (3)，式 (4) となる．

$$\begin{aligned} T_{Trix3\_bc} = {} & H \times (L_{mpi} + SS \div BW_{comm}) \\ & + \max(N \div SS \times L_{mpi}, \\ & \quad N \div 3 \div BW_{comm}) \\ & + L_{mpi} \end{aligned} \tag{3}$$

$$\begin{aligned} T_{Trix3\_rd} = {} & H \times (L_{mpi} + SS \div BW_{comm} \\ & \quad + SS \div BW_{comp}) \\ & + \max(N \div SS \times L_{mpi} \\ & \quad + N \div BW_{comp}, \\ & \quad N \div 3 \div BW_{comm}) \\ & + L_{mpi} \end{aligned} \tag{4}$$

5. 「京」向けの MPI_Allreduce の実装

Trinaryx3 Allreduce の実装としては，パイプライン転送を用いる．パイプライン転送では，メッセージを一定サイズのセグメントに区切って転送する．セグメントサイズを適切に調整すると，1 つのセグメントを転送している間に次のセグメントを受信することができ，通信レイテンシとソフトウェア処理にかかるレイテンシを隠蔽することができる．

レイテンシが大きい場合，パイプライン転送を円滑に行うにはセグメントサイズを長くする必要がある．しかしながら，セグメントサイズを長くすれば長くするほど，先頭セグメントの到着に遅れが生じてしまう．この遅れは，実行環境の規模が大きくなるほど，つまり木の高さが高くなるほど影響が大きくなる．よって，特に超並列においては，レイテンシを短くすることが必須である．

本章では，「京」向けの MPI ライブラリにおける，ソフトウェアオーバヘッドの削減について述べる．

5.1 通信レイテンシ

一般的な MPI ライブラリの集団通信は，send/recv モデルの通信で実装されている．これは，多種多様なネットワークインタフェースに対応するためと考えられる．しかしながら，send/recv モデルの通信は，メモリコピーや rendezvous 通信によるオーバヘッドがあり，特にパイプラ

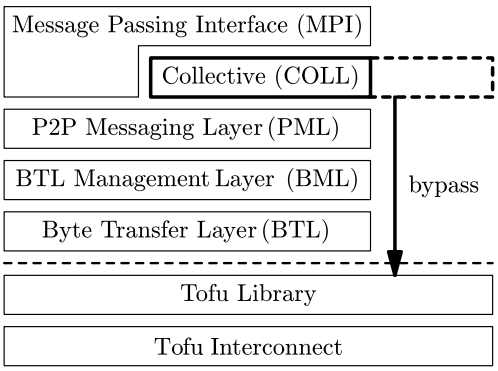

図 8　Open MPI 通信層の階層構造
Fig. 8　Open MPI's communication stack.

イン転送で扱うようなセグメント長においては，通信路の性能を引き出すことができない．そこで，「京」向けの MPI ライブラリでは，send/recv モデルの通信ではなく，Tofu インターコネクトの RDMA を利用する．

Tofu インターコネクトの RDMA で通信する場合，DMA アドレスを用いる．DMA アドレスは，通信バッファのアドレスをシステムに登録することで得られる．RDMA 通信では，リモートプロセスの DMA アドレスを事前に知る必要がある．そのため，実際のメッセージのやりとりを行う前に，DMA アドレスの通知を行う．この通信を制御通信と呼ぶ．一般的には，通信対象となるバッファは通信ごとに異なる．そのため，通信ごとに制御通信を行う必要がある．しかし，セグメント分割された各領域の通信は，オフセットが異なるだけで，同じバッファを対象とする．そのため，バッファ先頭の DMA アドレスを通知する制御通信 1 回のみで，全セグメントの RDMA 転送が可能となる．これにより，実メッセージのやりとりに制御通信が割り込むことで発生する外乱を抑えることができる．

5.2　通信階層のバイパス

「京」向けの MPI ライブラリは，システムへの適合性と拡張性の高さから，Open MPI をベースとしている．Open MPI は，多種多様な環境で動作させることを想定して，ライブラリを階層化し，実行環境に応じて各階層のモジュールを入れ替えて使用することができるようになっている．**図 8** に Open MPI 通信層の階層構造を示す．一番上の MPI 層は，ユーザインタフェースを提供する層である．PML 層は 1 対 1 通信機能を提供する層で，以下 BML 層，BTL 層を経由して，Tofu インターコネクト用の API を提供する Tofu ライブラリに至る．BTL 層までが MPI ライブラリの範囲である．

集団通信は，太枠で示した COLL という層に実装されている．Open MPI の実装では，PML，BML，BTL という 3 層を通ることになり，ソフトウェアオーバヘッドが大きくなる．また，そもそも PML はランク番号ベースの send/recv モデルの通信 API しか提供していない．そこで，これら 3 層をバイパスし，直接 Tofu ライブラリを使用する実装とする．これにより，低レイテンシで RDMA が使用可能になる．また，Tofu ライブラリを用いてトポロジ解析を行うことでコミュニケータの形状を認識し，三次元トーラスを意識した集団通信の実行を可能とする．

```
1: Op2(A[], B[]):
2:   for i = 0 to N-1:
3:     B[i] = B[i] ○ A[i];

1: Op3(A[], B[], C[]):
2:   for i = 0 to N-1:
3:     C[i] = B[i] ○ A[i];
```

図 9　演算部擬似コード
Fig. 9　Pseudocode of reduction operation.

5.3　演算

Reduce においては，演算結果を通信するため，演算処理もソフトウェアオーバヘッドの一部となる．よって，演算処理にかかる時間を削減する必要がある．

「京」向けの MPI ライブラリは，動作環境が固定されているため，CPU アーキテクチャに特化した最適化が有効である．「京」の SPARC64 VIIIfx プロセッサ [6] は，HPC-ACE 拡張により，2 並列の SIMD 命令が使用可能となっている．

演算処理は，**図 9** に示すような 2 種類の関数で行う．どちらも容易に SIMD 化が可能なほか，ループアンローリングやソフトウェアパイプライニングといったループ最適化も適用できる．これらの最適化は，「京」向けのコンパイラを用いて自動的に行う．ただし，Open MPI の既存実装では，ループ内で参照されるポインタ変数に偽の依存が発生しており，最適化が阻害されてしまうため，若干の修正をくわえる．最適化の有無による性能差については，6 章で検証する．

6.　Trinaryx3 Allreduce の評価

本章では，Trinaryx3 Allreduce アルゴリズムの評価について述べる．Trinaryx3 Allreduce アルゴリズムのトーラスでの性能を確認するため，既存のトポロジを考慮していないアルゴリズムとの性能比較を「京」上で行った．次に，既存のトポロジを考慮していないアルゴリズムと Trinaryx3 Allreduce においてメッセージの衝突の有無を検証する．さらに Trinaryx3 Allreduce の性能解析を行う．最後に 4.3 節で作成したモデルとの性能の比較を行った．

6.1　測定環境と測定方法

「京」の 9216 ノード（三次元トーラスで $48 \times 6 \times 32$）を使用して測定を行った．比較するアルゴリズムは，Trinaryx3

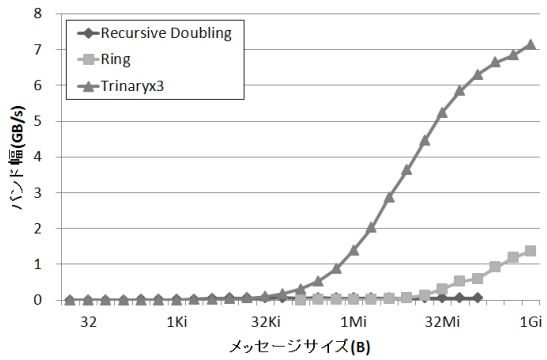

図 10　Allreduce のアルゴリズム別バンド幅（1 GiB まで）
Fig. 10　Allreduce thoughput (up to 1 GiB).

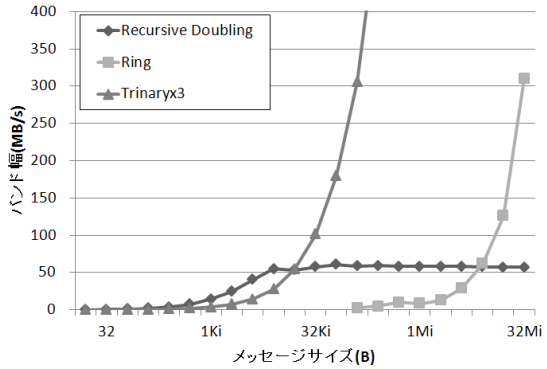

図 11　Allreduce のアルゴリズム別バンド幅（32 MiB まで）
Fig. 11　Allreduce thoughput (up to 32 MiB).

Allreduce および Open MPI で実装されている Recursive Doubling と Reduce_scatter + Allgather アルゴリズムである．Reduce_scatter + Allgather アルゴリズムを Open MPI では Ring アルゴリズムと呼んでおり，以下本論文でも Ring アルゴリズムと呼ぶ．使用する MPI ライブラリは，現在富士通が開発中のものである．性能測定は，データ型は MPI_DOUBLE で演算が MPI_SUM の MPI_Allreduce を行う自作プログラムで実施する．

6.2　バンド幅と実行時間測定

メッセージは 16 B から 1 GiB まで測定を行った．**図 10** は，16 B から 1 GiB までのアルゴリズム別のバンド幅を示している．**図 11** は，16 B から 32 MiB までのアルゴリズム別のバンド幅を，**図 12** は，16 B から 32 KiB までのアルゴリズム別の実行時間を示している．本論文では，バンド幅を集団通信実行時に通過するデータ量 ÷ 実行時間 と規定し，Allreduce のバンド幅は，メッセージの長さ ×2 ÷ 実行時間として求めた．

図 10 より，長メッセージにおいて，既存のトポロジを考慮していないアルゴリズムは，より高速な Ring アルゴリズムでも最大 1.4 GB/s であるが，Trinaryx3 Allreduce は最大で 7.1 GB/s のバンド幅であり，5.1 倍のバンド幅を達成していることが分かる．Trinaryx3 Allreduce は 5 章

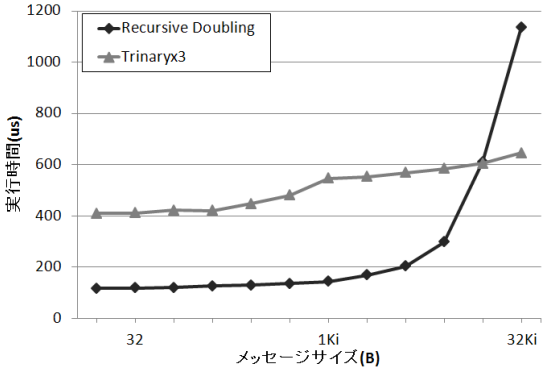

図 12　Allreduce のアルゴリズム別実行時間（32 KiB まで）
Fig. 12　Allreduce latency (up to 32 KiB).

で述べたとおり，複数の RDMA エンジンで多方向に同時に通信することが可能であり，通信と演算をオーバラップさせることができるため，高速な通信を実現することができる．一方，既存のトポロジを考慮していない 2 つのアルゴリズムはともに，1 個の RDMA エンジンしか使用していないことと，通信と演算が逐次的に処理されていることで，Trinaryx3 Allreduce と比較して性能が劣っている．

一方，図 11，図 12 より，Trinaryx3 Allreduce は，短メッセージにおいては，既存のトポロジを考慮していないアルゴリズムである Recursive Doubling より性能が劣ることが分かる．これは，以下のような理由によるものと考えられる．

短メッセージでは，パイプライン通信において，メッセージを中継する回数（パイプラインの最大ホップ数）が性能に大きく影響する．Recursive Doubling では，パイプラインの最大ホップ数は 15 である．

一方，Trinaryx3 Allreduce では，最長経路は木が最も深いところである．4 章より，3 次元トーラスの x，y，z 軸の軸長を X，Y，Z とおくと，木の高さは $X+Y+Z-2$ であるため，84 となる．Reduce と Bcast で leaf-root 間を往復するため，最大ホップ数は 168 となる．このことから，短メッセージでは，最大ホップ数が小さい Recursive Doubling が Trinaryx3 Allreduce より性能が良くなると考えられる．

また数百 KiB 程度では，Ring はバンド幅が伸びていない．理由は Reduce_scatter の後で各プロセスが担当するメッセージが全体のメッセージサイズをプロセス数で割った値となり，その値が数十 B と短いためだと考えられる．これは，2 章で述べた Reduce_scatter+Allgather のアルゴリズムの超並列における傾向と一致する．

図 10，図 11，図 12 の結果より，Trinaryx3 Allreduce は，「京」上において，中程度から長メッセージで，他の既存のトポロジを考慮していないアルゴリズムと比較して，高速なアルゴリズムといえる．

6.3　メッセージの衝突

アルゴリズムごとのメッセージの衝突状況について，インターコネクトコントローラ内の統計情報を用いて調査した．対象とするアルゴリズムは，Trinaryx3 Allreduce と Recursive Doubling の 2 つである．Ring は，2 章で述べたとおり，1 回の通信におけるメッセージサイズが小さいため，衝突が起こりにくいと考え，除外した．メッセージサイズは，Recursive Doubling のバンド幅が停滞している 1 MiB とした．

1 回の MPI_Allreduce の実行における，全リンク数における転送待ちが発生したリンク数の割合を求めた．その結果，Recursive Doubing では，36%であり，Trinaryx3 Allreduce では 0.89%であった．転送待ち時間が計上される要因は複数あるが，2 つ以上のメッセージが同じタイミングで同一リンクを通過しようとする場合に調停によって待たされる時間が主であり，メッセージの衝突具合いの指標として有効である．

Recursive Doubling では，3 分の 1 以上のリンクで転送待ちが検出されており，メッセージ衝突が頻繁に起きていることを示唆している．これは，Recursive Doubling がトポロジを考慮していないアルゴリズムであることと合致している．一方で，Trinaryx3 Allreduce では，転送待ちが検出されたリンクは約 1%と，Recursive Doubling に比べて非常に少ない．Trinaryx3 Allreduce は，隣接プロセスへの通信のみで構成されており，アルゴリズムの性質として，衝突が起きない．ここで計上された転送待ちは，衝突以外の要因によるものと考えられる．

6.4　Trinaryx3 Allreduce の性能解析

6.4.1　Reduce と Bcast の実行時間内訳

Trinaryx3 Allreduce の性能解析のため，Bcast と Reduce に分割してコストを調査した．**図 13** は，1 GiB の MPI_Allreduce における Bcast と Reduce の実行時間を積み上げグラフで示したものである．Bcast のセグメントの長さは 64 KiB であり，Reduce のセグメントの長さは 48 KiB である．図の上から順に，演算を無効化し，通信処理のみとした場合（ideal），コンパイラによる演算の最適化が有効な場合（proposal），演算の最適化が無効な場合（no-opt）の測定結果である．積み上げグラフは左から順に，Bcast の実行時間，Reduce の実行時間である．Reduce の実行時間は，3 つの部分の合計からなり，左から順に，通信に要した時間，通信と演算がオーバラップしている時間，演算の時間である．

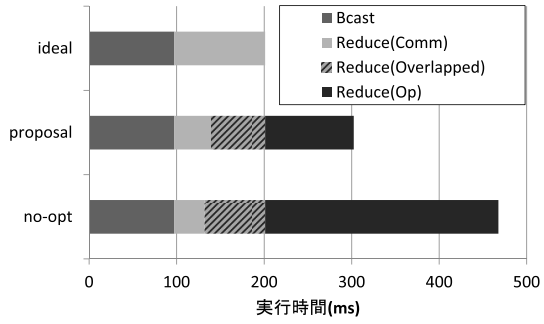

図 13　Trinaryx3 Allreduce の実行時間内訳
Fig. 13　Detail of Trinaryx3 Allreduce execution time.

はじめに，演算を無効化した結果より，Reduce の通信にかかる時間は，Bcast と同程度である．この Reduce の通信にかかる時間が，本来の通信時間と考えられる．しかし，最適化が有効な場合の結果と比較すると，Reduce の実行時間が，Reduce の本来の通信の時間を上回っている．このことより，Reduce の実行時間を演算の実行時間が決定しているように見える．最適化が有効な場合のグラフより，演算が Reduce の実行時間に占める割合は 80%程度であり，Allreduce 全体の 54%程度を占めている．

演算の性能が Allreduce の性能に与える影響が大きいことを確認するために，演算部分を最適化しないライブラリでも同様のデータを採取した（図 13 下のグラフ）．演算を最適化したライブラリと比較すると，演算部分の時間が 1.97 倍となり，Reduce 全体では 1.80 倍，Allreduce 全体は 1.54 倍と悪化した．この結果より，Reduce の演算部分が性能に与える影響は大きい．

最適化した場合の演算のスループットは約 6.58 GB/s である．すなわち，シングルスレッドでの STREAM [9] Add 相当の性能は約 19.8 GB/s である．一方，「京」の STREAM Triad 性能は，マルチスレッドでの実行で，46.6 GB/s となっている [20]．このため，演算部分の改善手法としてはマルチスレッド化が考えられる．しかし，今回はシステムとしての安定動作を優先するため，採用しない．

6.4.2　Trinaryx3 Allreduce のセグメントサイズ

5 章で述べたとおり，セグメントサイズを適切に調整することで，通信レイテンシとソフトウェア処理にかかるレイテンシを隠蔽することが可能である．**図 14** は，1 KiB から 1 GiB までの各メッセージサイズにおける Reduce と Bcast の最適なセグメントサイズである．Allreduce におい

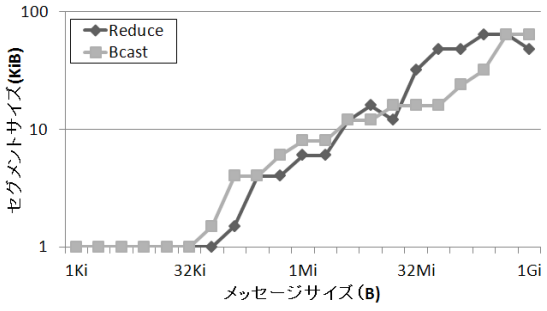

図 14　Trinaryx3 Allreduce の実測より求めた最適なセグメントサイズ
Fig. 14　Optimal segment size of Trinaryx3 Allreduce from measurement.

て Reduce と Bcast のセグメントサイズをそれぞれ 1 KiB から 64 KiB まで値を変更し，測定結果より求めた値である．図より，メッセージ長が大きくなるにつれて，最適なセグメントサイズは増加傾向にあることが分かる．セグメントサイズは演算の有無によらず増加傾向にある．最適なセグメントサイズを決める要因は，演算よりも通信の影響が大きいと考えられる．よって，通信がセグメントサイズに与える影響を調査する必要がある．

6.4.3 Trinaryx3 Allreduce の性能モデルとの比較

4.3 節で定義したモデルに沿って，6.4.1 項の図 13 あるいは参考文献から得られた，パラメータの実測値より評価を行う．6.2 節に記述してあるとおり，パイプラインの最大段数 H は 84 である（Reduce と Bcast でそれぞれ分けて評価するため 84 とする）．3 経路を使用した双方向の通信バンド幅は，11.6 GB/s であり [1]，ここでは 1 経路あたりのバンド幅として $BW_{comm} = 11.6 \div 3 = 3.87$ GB/s を使用する．通信レイテンシ L_{mpi} は，1.27 マイクロ秒である [18]．演算のスループット BW_{comp} は 6.4.1 項の結果より，6.58 GB/s である．Bcast については SS は 64 KiB であり，Reduce については SS は 48 KiB である．4.3 節の式 (3)，式 (4) にそれぞれ値を代入すると以下のようになる．T_{Trix3_bc} は 94.1 ミリ秒であり，T_{Trix3_rd} は 193 ミリ秒である．

Bcast においては，モデル式の 1.04 倍の実行時間，Reduce においては，モデル式の 1.06 倍の実行時間となっている．この結果より我々の実装が設計どおりの性能となっているといえる．

7. 関連研究

7.1 EDF broadcast

Simmen [15] と Barnett ら [5] は，Edge Disjoint Fence（EDF）broadcast と呼ばれるトーラス向け Bcast アルゴリズムを提案している．EDF broadcast は，トーラス軸を意識した複数経路を使用してパイプライン転送を行うアルゴリズムで，木の構成方法は Trinaryx3 とまったく同一である．ただし，Reduce や Allreduce への応用や，通信の順序については触れられていない．

Barnett らは，二次元メッシュ構造の Intel Paragon 上で EDF broadcast の評価を行っている．また，Watts ら [17] は，三次元トーラス構造の Cray T3D 上で評価を行っている．これらの研究においては，EDF broadcast は他アルゴリズムよりも性能が出ないという結論になっている．その原因として，どちらのネットワークにおいても，送受信が複数同時に行えないため，パイプライン転送のコストが大きくなることと，複数のメッセージが同時に到着することによりパイプラインが乱れることがあげられている．「京」のネットワークでは，複数方向からの送受信がまったく独立して行えるため，このような問題は発生しない．

7.2 Blue Gene/L

Blue Gene/L の MPI ライブラリは，三次元メッシュ向けの集団通信を実装している [3]．長メッセージ向けの Allreduce アルゴリズムとして，EDF broadcast をメッシュ向けに適用した Reduce + Bcast アルゴリズムが使われている．これにより，中・長メッセージで MPICH2 に実装された既存のアルゴリズムを凌駕することが示されている．

ただし，Reduce については，複数のプロセスからメッセージを受け取るプロセスがボトルネックとなって性能が出ないという問題点が指摘されている．このため彼らは，すべてのプロセスで入次数と出次数が 1 以下になるような，ハミルトンパスを使用したアルゴリズムを提案し，長メッセージでの有効性を示している．しかしながら，この手法は，レイテンシがプロセス数に比例するため，低並列でのみ有効だと考えられる．

数値としては，512 ノードでの実行で，MPI_Allreduce の最大バンド幅が約 120 MB/s という結果が報告されている（本論文での計算方法に合わせ再計算）．Trinaryx3 Allreduce はその 58 倍のバンド幅だが，これは，リンクの絶対性能の違いによるところが大きい．ただし，Blue Gene/L ではピークバンド幅に対する効率が 26%であるのに対し，Trinaryx3 Allreduce は 47%の効率であり，相対的な効率の面でも Blue Gene/L を上回っている．

8. まとめ

本論文では，トーラス向けの Bcast アルゴリズムである Trinaryx3 Bcast を提案し，それを Allreduce に応用した Trinaryx3 Allreduce アルゴリズムを設計した．「京」上で実装し，既存のトポロジを考慮していないアルゴリズムと性能比較を行った結果，既存のトポロジを考慮していないアルゴリズムと比較して，バンド幅が最大 5 倍程度となった．また，リンクごとの転送待ち時間によりメッセージの衝突状況を調査した結果，既存のトポロジを考慮していないアルゴリズムでは転送待ちが頻繁に発生していた一方で，Trinaryx3 Allreduce では転送待ちはほとんど発生していなかった．この結果により，Trinaryx3 Allreduce は「京」上で有効なアルゴリズムであるといえる．

「京」上で実装した Trinaryx3 Allreduce の実行時間を分析した結果，Reduce の演算時間が性能に与える影響が大きく，Allreduce 全体の 54%を占めていた．しかし，現在のシステムでは演算部分のさらなる性能改善は困難と考えられる．また，最適なセグメントサイズは，メッセージ長が増加するにつれて Reduce，Bcast ともに増加する傾向であり，通信の与える影響が大きいと考えられる．さらに，Trinaryx3 Allreduce の実行時間のモデル化を行い，1 GiB の Allreduce の実測結果との比較を行った．この結果，Bcast 部分はモデル式の 1.04 倍，Reduce 部分は 1.06 倍の実行時間であった．この結果より設計どおりの性能で

ある．

謝辞　本論文を執筆するにあたり，富士通株式会社の三吉郁夫氏ならびに三輪英樹氏には，性能測定に際して有益な助言をいただいた．ここに感謝の意を表する．

参考文献

[1] Adachi, T., Shida, N., Miura, K., Sumimoto, S., Uno, A., Kurokawa, M., Shoji, F. and Yokokawa, M.: The Design of Ultra Scalable MPI Collective Communication on the K Computer, *ISC 2012* (2012).

[2] Ajima, Y., Sumimoto, S. and Shimizu, T.: Tofu: A 6D Mesh/Torus Interconnect for Exascale Computers, *Computer*, Vol.42, No.11, pp.36–40 (2009).

[3] Almási, G., Archer, C., Erway, C.C., Heidelberger, P., Martorell, X., Moreira, J.E., Steinmacher-Burow, B.D. and Zheng, Y.: Optimization of MPI Collective Communication on BlueGene/L Systems, *Proc. ICS 2005*, pp.253–262 (2005).

[4] Barnett, M., Littlefield, R.J., Payne, D.G. and van de Geijn, R.A.: Global Combine on Mesh Architectures with Wormhole Routing, *Proc. IPPS '93*, pp.156–162 (1993).

[5] Barnett, M., Payne, D.G., van de Geijn, R.A. and Watts, J.: Broadcasting on Meshes with Worm-Hole Routing, *Journal of Parallel and Distributed Computing*, Vol.35, No.2, pp.111–122 (1996).

[6] Fujitsu: SPARC64 VIIIfx Extensions, available from ⟨http://www.fujitsu.com/downloads/TC/sparc64viiifx-extensions.pdf⟩.

[7] Graham, R.L., Shipman, G.M., Barrett, B.W., Castain, R.H., Bosilca, G. and Lumsdaine, A.: Open MPI: A High-Performance, Heterogeneous MPI, *Proc. HeteroPar 2006* (2006).

[8] Johnsson, S.L. and Ho, C.-T.: Optimum Broadcasting and Personalized Communication in Hypercubes, *IEEE Trans. Comput.*, Vol.38, No.9, pp.1249–1268 (1989).

[9] McCalpin, J.D.: Memory Bandwidth and Machine Balance in Current High Performance Computers, *IEEE TCCA Newsletter*, pp.19–25 (1995).

[10] Message Passing Interface Forum: MPI: A Message-Passing Interface Standard (1995), available from ⟨http://www.mpi-forum.org/⟩.

[11] Message Passing Interface Forum: MPI-2: Extensions to the Message-Passing Interface (1997), available from ⟨http://www.mpi-forum.org/⟩.

[12] MPICH2: available from ⟨http://www.mcs.anl.gov/research/projects/mpich2/⟩.

[13] Rabenseifner, R.: Automatic MPI Counter Profiling of All Users: First Results on a CRAY T3E 900-512, *Proc. MPIDC '99*, pp.77–85 (1999).

[14] Rabenseifner, R.: Optimization of Collective Reduction Operations, *Proc. ICCS 2004*, pp.1–9 (2004).

[15] Simmen, M.: Comments on broadcast algorithms for two-dimensional grids, *Parallel Computing*, Vol.17, No.1, pp.109–112 (1991).

[16] van de Geijn, R.A.: Efficient Global Combine Operations, *Proc. DMCC '91*, pp.291–294 (1991).

[17] Watts, J. and van de Geijn, R.A.: A Pipelined Broadcast for Multidimensional Meshes, *Parallel Processing Letters*, Vol.5, No.2, pp.281–292 (1995).

[18] 住元真司，川島崇裕，志田直之，岡本高幸，三浦健一，宇野篤也，黒川原佳，庄司文由，横川三津夫：「京」のための MPI 通信機構の設計，*SACSIS 2012* (2012).

[19] 追永勇次：次世代スパコン『京』について，サイエンティフィックソサイエティ研究会 HPC フォーラム 2011 (2011)，入手先 ⟨http://www.ssken.gr.jp/MAINSITE/activity/sectionmeeting/sci/2011-1/program.html⟩.

[20] 林　正和：次世代スパコン『京（けい）』の言語処理系と評価，サイエンティフィックソサイエティ研究会 2010 年度科学技術計算分科会 (2010)，入手先 ⟨http://www.ssken.gr.jp/MAINSITE/download/newsletter/2010/20101020-sci-2/index.html⟩.

松本 幸　（正会員）

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士課程修了．同年富士通（株）に入社．並列計算向け通信ライブラリの開発に従事．

安達 知也

2010 年東京大学大学院理工学系研究科コンピュータ科学専攻修士課程修了．同年より富士通（株）勤務．並列計算用通信ライブラリの研究開発に従事．

住元 真司　（正会員）

1986 年同志社大学工学部電子工学科卒業．同年富士通（株）入社．（株）富士通研究所にて並列分散 OS の研究開発に従事．1997 年より新情報処理開発機構に出向．PC クラスタ向け高速通信機構の研究開発に従事，2002 年より（株）富士通研究所にて高速通信機構，大規模 PC クラスタのシステム開発に従事，2007 年より富士通（株）にてスーパーコンピュータ「京」のシステム開発，MPI 通信ライブラリ，クラスタファイルシステム開発に従事．平成 12 年度情報処理学会論文賞受賞，SACSIS2010 最優秀論文賞受賞，工学博士．

南里 豪志　（正会員）

1995 年九州大学大学院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，現在に至る．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）．

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，主にスーパー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．2011 年九州大学学術研究員．計算機アーキテクチャ，並列計算システム等に関する研究に従事．工学博士．

宇野 篤也　（正会員）

2000 年筑波大学大学院工学研究科博士課程修了．博士（工学）．現在，理化学研究所計算科学研究機構運用技術部門システム運転技術チーム開発研究員．システムソフトウェア関連の研究開発に従事．

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）．主にシステムのデザイン・設計，並列 CFD，並列分散処理，システム評価に興味を持つ．スーパーコンピュータ・システムの設計・運用管理に従事．IEEE，IEEE-CS 各会員．

庄司 文由

1997 年金沢大学大学院自然科学研究科単位取得退学．1998 年広島大学情報教育研究センター助手．2005 年理化学研究所次世代スーパーコンピュータ開発実施本部開発研究員．現在，理化学研究所計算科学研究機構運用技術部門システム運転技術チームチームヘッド．大規模スーパーコンピュータの開発と運用に従事．博士（理学）．

横川 三津夫　（正会員）

1984 年筑波大学大学院理工学研究科修士課程修了．2006 年から理化学研究所次世代スーパーコンピュータ開発実施本部にて，スーパーコンピュータ「京」の開発に従事．現在，理化学研究所計算科学研究機構運用技術部門部門長．工学博士．