Vol.2011-ARC-197 No.6
Vol.2011-HPC-132 No.6
2011/11/28

MPI_Allreduceの「京」上での実装と評価

松本 幸[†1] 安達知也[†1] 田中 稔[†1]
住元真司[†1] 曽我武史[†2] 南里豪志[†2]
宇野篤也[†3] 黒川原佳[†3]
庄司文由[†3] 横川三津夫[†3]

本報告では，8 万台以上のノードを直接網で結合した「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 集団通信アルゴリズムは，間接網向けのアルゴリズムが主体であり，これを直接網に適用してもメッセージの衝突のため効率的な通信ができない．このため，高い通信性能を得るためには直接網を意識した集団通信アルゴリズムが必須となる．そこで我々は，トーラス向け Allreduce アルゴリズム Trinaryx3 Allreduce を設計し，「京」向けの MPI ライブラリに実装した．Trinaryx3 Allreduce は，「京」の特長である複数 RDMA エンジンを同時に活用することができる．実装を評価した結果，既存の間接網向けアルゴリズムと比較して，5 倍程度バンド幅が向上することを確認した．

Implementation and Evaluation of MPI_Allreduce on the K computer

Yuki Matsumoto,[†1] Tomoya Adachi,[†1]
Minoru Tanaka,[†1] Shinji Sumimoto,[†1]
Takeshi Soga,[†2] Takeshi Nanri,[†2] Atsuya Uno,[†3]
Motoyoshi Kurokawa,[†3] Fumiyoshi Shoji[†3]
and Mitsuo Yokokawa [†3]

This paper reports a method of speeding up MPI collective communication on the K computer, that consists of more than 80 thousand computing nodes connected by direct network. Almost all existing MPI libraries only implement algorithms optimized for indirect network. However, such algorithms perform poor on direct network because of collisions of the messages. Thus, in order to achieve high performance on direct network, it is necessary to implement collective algorithms optimized for the network topology. In this paper, Trinaryx3 Allreduce algorithm is designed and implemented in the MPI library for the K computer. The algorithm is optimized for torus network and enables utilizing multiple RDMA engines, one of the strengths of the K computer. The evaluation result shows that the new implementation achieves five times higher bandwidth than existing one, optimized for indirect 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[15] 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」は，8 万台ノード以上の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ーフェースは，MPI 規格[13),14)] によって定められている．MPI 規格では，一対一通信のほかに，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ため，数多くの研究がなされている[6),9)–11)]．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[2),3),8),20)]．

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．間接網向けのアルゴリズムは，プロセスに対してトポロジー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性能が伸び悩んでしまう．性能を引き出すためには，直接網向けの集団通信アルゴリズムの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，

†1 富士通株式会社
Fujitsu Ltd.
†2 九州大学
Kyushu University
†3 理化学研究所
RIKEN

Vol.2011-ARC-197 No.6
Vol.2011-HPC-132 No.6
2011/11/28

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[16]．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，本報告では，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，それを利用して Allreduce アルゴリズム Trinaryx3 Allreduce を設計する．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[1] の特長である複数の RDMA エンジンを活用することで，低レイテンシ，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 由来の既存の間接網向け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．

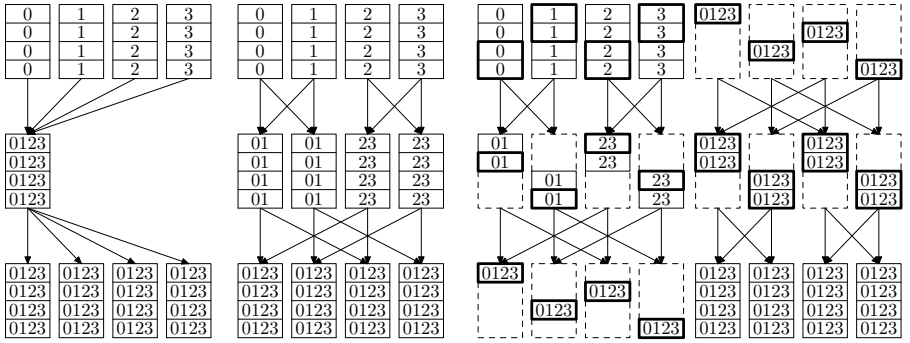
2. MPI_Allreduce の概要と既存アルゴリズム

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，本報告では，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．

本報告では，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予め定義されているほか，ユーザー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，以下の二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常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．



(a) Reduce + Bcast　(b) Recursive Doubling　(c) Reduce_scatter + Allgather

図 1　Allreduce アルゴリズム

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三種類紹介し，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，一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[3]．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(a)** に示す．

通信は，プロセスをグラフの頂点とみ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一対一で直接結ぶ方法や，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ー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．一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，本報告では，このアルゴリズムを「京」向けの MPI ライブラ

Vol.2011-ARC-197 No.6
Vol.2011-HPC-132 No.6
2011/11/28

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 1(b)**）．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[17]．

このアルゴリズムは，各ステップにおいて，全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，一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[19] である（Rabenseifner のアルゴリズム[17] 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 1(c)** 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，一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，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極めて巨大な，限られた範囲でしか有効でないと考えられる．

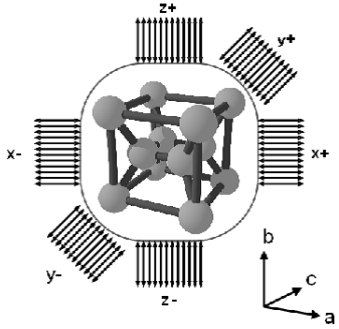


図 2 Tofu のネットワーク (a,b,c 軸)

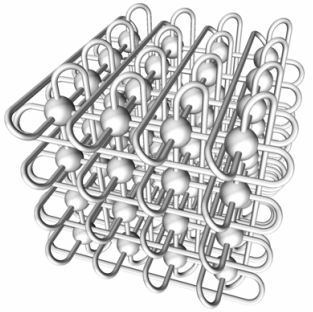


図 3 Tofu のネットワーク (X,Y,Z 軸)

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[1] について述べる．「京」は 8 万台以上のノードで構成されており，そのネットワークは，3 次元トーラスを拡張した，X,Y,Z,a,b,c の 6 次元からなるメッシュ/トーラス構造である． **図 2** のような a, b, c 軸が 2x3x2 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 **図 3** のように組み合わせることで 6 次元メッシュ/トーラスとなる．

利用者はジョブの性質に合わせて，ジョブを 1,2,3 次元トーラスに割り付けて実行することが可能である．6 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．

各ノード内には，**図 4** のようなインターコネクトコントローラが存在し，4 つの RDMA エンジンが搭載されている[21]．このインターコネクトコントローラでは，10 方向に対しての送受信が可能で，RDMA エンジンでは 1 度に最大 16MiB 弱の転送が可能である．また，この 4 個の RDMA エンジンを使用して，最大 4 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 20GB/sec[21] である．

Vol.2011-ARC-197 No.6
Vol.2011-HPC-132 No.6
2011/11/28

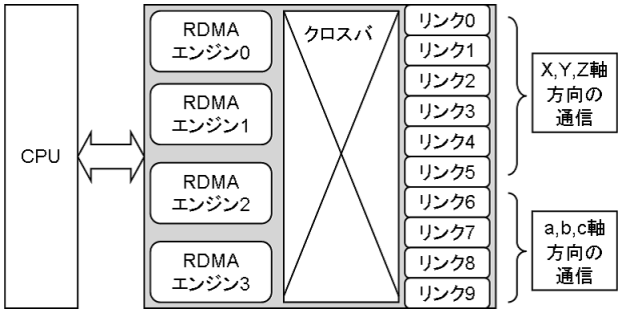


図 4 ノード内のインターコネクトコントローラ概要

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三つの辺素な全域木を構成し，三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，通信は全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 5** に示す．tree 0 では，まず root プロセスから x 軸方向へ枝を伸ばす．次に，新しく接続されたプロセスから y 軸方向へ枝を伸ばしていく．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枝を張ることで補完する．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，z 軸方向の順に接続して木を構成する例を**図 6** 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，三つの辺素な全域木が得られる．

通信は，メッセージを三分割し，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，これらの通信は全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，一経路を用いた場合の三倍になることが期待される．

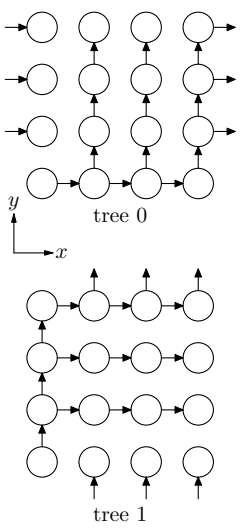


図 5 Trinaryx3 Bcast

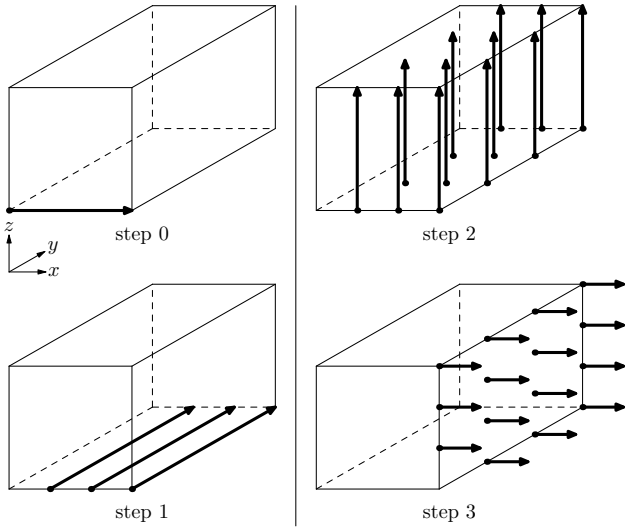


図 6 三次元トーラスでの構成

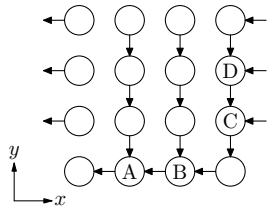


図 7 複数方向からの受信

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．二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．

Vol.2011-ARC-197 No.6
Vol.2011-HPC-132 No.6
2011/11/28

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．**図 7** における A,B,C,D のようなプロセスである．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．全プロセスが一斉に Reduce を始める前提で考えると，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．図 7 の例では，A と B は y 軸方向，C と D は x 軸方向のプロセスからのメッセージを優先して処理する．

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．

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，一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ー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ー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 通信によるオーバーヘッドがあり，特にパイプラ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．

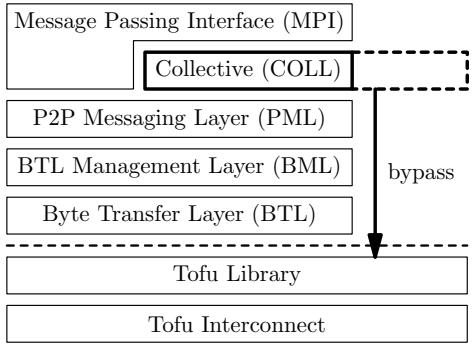


図 8 Open MPI 通信層の階層構造

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一回のみで，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，ユーザーインターフェースを提供する層である．PML 層は一対一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

Vol.2011-ARC-197 No.6
Vol.2011-HPC-132 No.6
2011/11/28

```
 1: Bcast-loop():
 2:   infinite-loop:
 3:     for-each tree, for-each child:
 4:       if there exists an unsent message:
 5:         Send message to child;
 6:
 7:     for-each tree:
 8:       Check message from parent;
 9:       if arrived:
10:         for-each child:
11:           Send message to child;
12:
13:     if done for all trees:
14:       break;
```

図 9 複数の木を使用した Bcast

```
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 10 演算部擬似コード

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 という三層を通ることになり，ソフトウェアオーバー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，これら三層をバイパスし，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ー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．

5.3 パイプライン転送のループ構成

Trinaryx3 Allreduce では，複数の木を使用して，同時並行で通信を行う．複数方向からの受信を並列に処理し，複数方向へ送信する機構が必要となる．

一般的な MPI ライブラリでは，パイプライン転送の受信待ち処理は MPI_Recv や MPI_Wait 相当のブロッキング受信で実装されている．これは，実装されているアルゴリズムが一つの経路しか使用しないためと考えられる．しかし，Trinaryx3 Allreduce では，複数経路を通って非同期にメッセージが到着する．そのため，ブロッキング受信ではなくノンブロッキング受信で実装する必要がある．「京」向けの MPI ライブラリでは，RDMA エンジンごとに用意された完了キューを順にポーリングすることで実現する．

図 9 に Bcast のパイプライン転送ループの擬似コードを示す．3 行目から 5 行目が，root プロセスからの送信処理である．7 行目から 11 行目は，残りのプロセスにおける転送処理である．各々の木に対し，8 行目で到着確認を行い，もしメッセージが到着していたら，11 行目で送信する．全ての送受信が終わるまで繰り返し，14 行目でループを抜ける．

受信処理が全て終わってから送信処理をすることにすると，9 行目から 11 行目は削除することができる．しかし，その場合は，メッセージの到着から送信までに，他の木について到着確認を行うレイテンシが余分にかかってしまう．よって，到着確認後すぐに送信を行う実装としている．

5.4 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ー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[5] は，HPC-ACE 拡張により，二並列の SIMD 命令が使用可能となっている．

演算処理は，**図 10** に示すような二種類の関数で行う．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優位性を確認するため，既存の間接網におけるアルゴリズムとの性能比較を「京」上で行った．次に，間接網に特化したアルゴリズムと Trinaryx3 Allreduce においてメッセージの衝突を検証する．さらに Trinaryx3 Allreduce の性能解析を行う．

6.1 測定環境と測定方法

「京」の測定環境は，**表 1**，**表 2** の通りである． 9216 ノードを使用して測定を行った．比較するアルゴリズムは，Trinaryx3 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，以下本報告でも 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

Vol.2011-ARC-197 No.6
Vol.2011-HPC-132 No.6
2011/11/28

表 1 ノード内の環境

CPU	SPARC64 VIIIfx
CPU 周波数	2.0GHz
メモリ	16GB/ノード

表 2 インターコネクトの環境

インターコネクト	Tofu インターコネクト
ネットワーク性能	リンクあたり 5GB/s（双方向）
測定に使用した形状	48x6x32(3 次元トーラス)

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 バンド幅測定

メッセージは 16B から 1GiB まで測定を行った．**図 11** は，16B から 32MiB までのアルゴリズム別のバンド幅を，**図 12** は，16B から 1GiB までのアルゴリズム別のバンド幅を示している．本報告では，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 11 より，Trinaryx3 Allreduce は，短メッセージにおいては，既存の間接網アルゴリズムである Recursive Doubling より性能が劣る．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，メッセージを中継する回数（最大ホップ数）が性能に大きく影響する．Recursive Doubling では，プロセス数 P を超えない 2 べきの数を Q とすると，最大ホップ数は $1+\log Q+1$ である．9216 プロセスで実行したため，最大ホップ数は $1+\log(8192)+1=15$ 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，$48+6+32-2=84$ 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 より性能がよ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．

一方，図 12 より，長メッセージにおいて，間接網に特化したアルゴリズムは，より高速な Ring アルゴリズムでも最大 1.4GB/s 程度であるが，Trinaryx3 Allreduce は最大で 7.1GB/s 程度のバンド幅であり，5 倍向上している． Trinaryx3 Allreduce は 5 章で述べたとおり，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ー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，

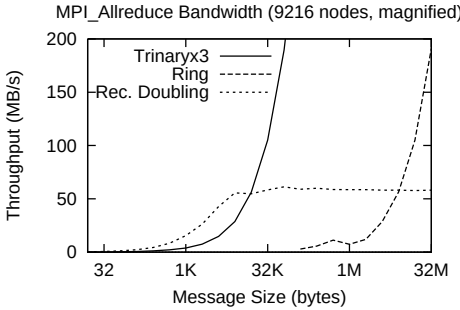


図 11 Allreduce のアルゴリズム別バンド幅（32MiB まで）

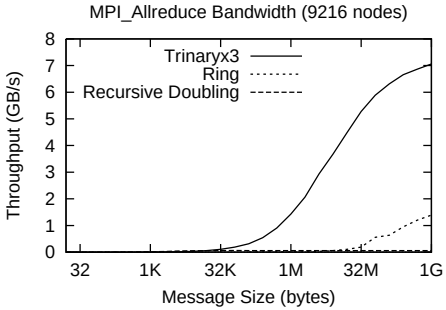


図 12 Allreduce のアルゴリズム別バンド幅（1GiB まで）

表 3 転送待ち時間

項目	Trinaryx3 Allreduce	Recursive Doubling
転送待ちが検出されたリンクの割合	1.98%	35.8%
転送待ちのべ時間	673us	108s
転送待ち平均時間	185ns	1.63ms
転送待ち最大時間	15.7us	19.9ms

間接網に特化した 2 つのアルゴリズムはともに，1 つの 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．したがって，Trinaryx3 Allreduce は,「京」上において，中程度から長メッセージで，他の間接網に特化したアルゴリズムと比較して，高速なアルゴリズムと言える．

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 の二つである．Ring は，2 章で述べたとおり，一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 1MiB とした．

一回の MPI_Allreduce の実行における，全プロセスのリンクごとの転送待ち時間を集計したものが**表 3** である．転送待ち時間が計上される要因は複数あるが，二つ以上のメッセー

Vol.2011-ARC-197 No.6
Vol.2011-HPC-132 No.6
2011/11/28

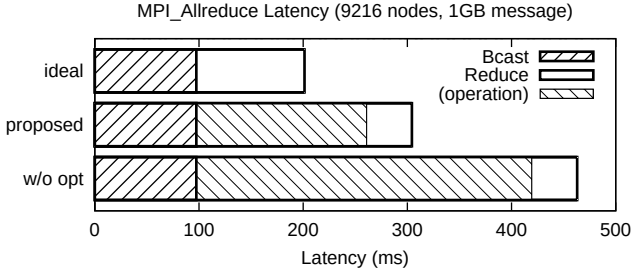


図 13 Allreduce の性能解析

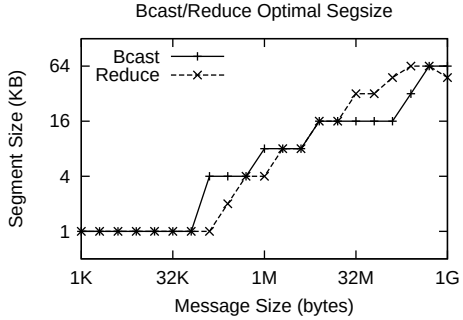


図 14 Trinaryx3 Allreduce の最適なセグメントサイズ

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の指標として有効である．

Recursive Doubling では，三分の一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 が直接網のことを意識し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 2%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 のコスト

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，1GiB の MPI_Allreduce における Bcast と Reduce のコストを積み上げグラフで示したものである．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の測定結果，コンパイラによる演算の最適化が有効な場合，演算の最適化が無効な場合の測定結果である． 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，白い部分と斜線の部分の合計からなり，そのうち，演算に要した時間を左側の斜線の部分で示している．

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 79%程度であり，Allreduce 全体の 53%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 1.76 倍，Allreduce 全体は 1.51 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 6.57GB/s である．すなわち，シングルスレッドでの STREAM[12] Add 相当の性能は約 19.7GB/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，46.6GB/s となっている[22]．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，1KiB から 1GiB 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．この値は，Reduce と Bcast に分けて測定結果より求めた値である．図より，数十 KiB までは，Bcast，Reduce 共に固定の値であるが，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．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．

7. 関連研究

7.1 EDF broadcast

Simmen[18] と Barnett ら[4] 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 と全く同一である．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[20] 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挙げられている．「京」のネットワークでは，複数方向からの送受信が全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[2]．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 に実装された間接網向け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，全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 120MB/s という結果が報告されている（本報告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

本稿では，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，既存の間接網向けのアルゴリズムと性能比較を行った結果，既存の間接網向けの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，既存の間接網向け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言える．

「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 53%を占めていた．しかし，現在のシステムでは演算部分の更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 共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．今後は，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．

参考文献

1) 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).
2) 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. of ICS 2005*, pp.253–262 (2005).
3) Barnett, M., Littlefield, R.J., Payne, D.G. and vande Geijn, R.A.: Global Combine on Mesh Architectures with Wormhole Routing, *Proc. of IPPS '93*, pp.156–162 (1993).
4) 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).
5) Fujitsu: SPARC64 VIIIfx Extensions, `http://www.fujitsu.com/downloads/TC/sparc64viiifx-extensions.pdf`.
6) Graham, R.L. and Shipman, G.: MPI Support for Multi-core Architectures: Optimized Shared Memory Collectives, *Proc. of EuroPVM/MPI 2008*, pp.130–140 (2008).

Vol.2011-ARC-197 No.6
Vol.2011-HPC-132 No.6
2011/11/28

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. of HeteroPar 2006* (2006).
8) Johnsson, S.L. and Ho, C.-T.: Optimum Broadcasting and Personalized Communication in Hypercubes, *IEEE Transactions on Computers*, Vol.38, No.9, pp. 1249–1268 (1989).
9) Karonis, N.T., de Supinkski, B.R., Foster, I., Gropp, W., Lusk, E. and Bresnahan, J.: Exploiting Hierarchy in Parallel Computer Networks to Optimize Collective Operation Performance, *Proc. of IPDPS 2000*, pp.377–384 (2000).
10) Mamidala, A.R., Kumar, R., De, D. and Panda, D.K.: MPI Collectives on Modern Multicore Clusters: Performance Optimizations and Communication Characteristics, *Proc. of CCGrid 2008*, pp.130–137 (2008).
11) Matsuda, M., Kudoh, T., Kodama, Y., Takano, R. and Ishikawa, Y.: Efficient MPI Collective Operations for Clusters in Long-and-Fast Networks, *Proc. of Cluster 2006* (2006).
12) McCalpin, J.D.: Memory Bandwidth and Machine Balance in Current High Performance Computers, *IEEE TCCA Newsletter*, pp.19–25 (1995).
13) Message Passing Interface Forum: MPI: A Message-Passing Interface Standard, `http://www.mpi-forum.org/` (1995).
14) Message Passing Interface Forum: MPI-2: Extensions to the Message-Passing Interface, `http://www.mpi-forum.org/` (1997).
15) MPICH2: `http://www.mcs.anl.gov/research/projects/mpich2/`.
16) Rabenseifner, R.: Automatic MPI Counter Profiling of All Users: First Results on a CRAY T3E 900-512, *Proc. of MPIDC '99*, pp.77–85 (1999).
17) Rabenseifner, R.: Optimization of Collective Reduction Operations, *Proc. of ICCS 2004*, pp.1–9 (2004).
18) Simmen, M.: Comments on broadcast algorithms for two-dimensional grids, *Parallel Computing*, Vol.17, No.1, pp.109–112 (1991).
19) van de Geijn, R.A.: Efficient Global Combine Operations, *Proc. of DMCC '91*, pp. 291–294 (1991).
20) 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).
21) 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,`http://www.ssken.gr.jp/MAINSITE/activity/sectionmeeting/sci/2011-1/program.html` (2011).
22) 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,`http://www.ssken.gr.jp/MAINSITE/download/newsletter/2010/20101020-sci-2/index.html` (2010).