Vol.2012-HPC-136 No.25
2012/10/4

Tofuネットワークにおけるプロセス配置形状による集団通信アルゴリズムの性能解析

南里 豪志[1,2,a)]　深沢 圭一郎[1,3,2,b)]

概要：スーパーコンピュータの大規模化に伴って，ノード間インターコネクトネットワークとして，コストの低い多次元メッシュ/トーラストポロジを採用したものを用いる事例が増えている．多次元メッシュ/トーラスは，使用するノード数が同じでも，プロセスが配置されるノード群の形状によって性能が大きく変動する．本研究では，京コンピュータや，その互換機である Fujitsu PRIMEHPC FX10 で用いられている Tofu インターコネクトネットワークを対象として，プロセス配置の形状による集団通信アルゴリズムの性能への影響を計測した．得られた性能を，Tofu インターコネクトの性能解析ツールを用いて取得した通信衝突による転送待ち時間と比較したところ，プロセス配置形状による変動がどちらもほぼ同じ傾向を示すことを明らかにした．これらの結果から，集団通信アルゴリズムの選択において，プロセス配置の形状を考慮した性能見積もりが重要であることを示した．

Performance Dependency Analysis of Algorithms for Collective Communications on Shape of Process Allocation in Tofu Interconnect

Abstract: Multi-dimensional mesh/torus topologies have become popular in the high-end supercomputers, because of their significant number of nodes. On these topologies, even with the same number of nodes, the shape of the group of nodes that are used for allocating processes affects the communication performance. In this paper, the performance dependency of algorithms of collective communications on the shape of process allocation is studied on Tofu interconnect. This is the interconnect used in K-computer and Fujitsu PRIMEHPC FX10. From the results of the experiments, this paper reveled that, as the shape of the process allocation changes, the performance of the algorithms of collective communications changes in similar way as the wait time caused by network collisions. This shows the importance of considering the shape of process allocation on the algorithm selection of collective communications.

1. 背景

現在，スーパーコンピュータと呼ばれる性能を有する計算機は，複数の計算ノードをインターコネクトネットワークで接続したクラスタ型の構成となっている．そのため，計算機の性能向上の手段としては，CPU の高速化やアクセラレータの搭載等によるノード内の性能向上とともに，ノード数の増加が用いられる．例えば，世界のスーパーコンピュータにおける Linpack ベンチマークプログラムの性能を比較した Top500 リストでは，2012 年 6 月の時点で上位 10 システムのうち 6 システムが 1 万ノード以上のノードで構成されている．このように計算ノード数が増えると，従来のクラスタで多く用いられていたファットツリートポロジによるインターコネクトは，ノード数あたりのスイッチ数やリンク数が膨大となり，電力や費用の問題により採用が困難となる．例えば前述の 1 万ノード以上の 6 システム中 5 システムでは，接続スイッチ数やリンク数の少ない多次元メッシュ/トーラストポロジが用いられている．多次元メッシュ/トーラストポロジでは，隣接するノード間の通信を非常に高速に行える反面，複数のメッセージが同じリンクを同時に使用する通信衝突の発生によって性能が低下する．しかもこの通信衝突の発生頻度は，実行時の状

[1] 九州大学情報基盤研究開発センター
Research Institute for Information Technology, Kyushu University
[2] JST CREST
[3] 九州大学国際宇宙天気科学・教育センター
International Center for Space Weather Science and Education, Kyushu University
[a)] nanri@cc.kyushu-u.ac.jp
[b)] fukazawa@cc.kyushu-u.ac.jp

況に依存する．特にプロセスが配置されるノード群の形状は，バイセクションバンド幅や選択される経路に影響するため，衝突の発生頻度への寄与が大きいと考えられる．このプロセス配置の形状は，全ノードによる並列プログラム実行時には一定である．しかし実際のスーパーコンピュータの利用では，プログラム内で複数のプロセスグループに分かれて通信を行ったり，ノード群を複数のジョブに割り当てて実行させることが多く，状況によって形状が異なる．

そこで著者らは，特に集団通信を対象とし，多次元メッシュ/トーラストポロジにおけるプロセス配置の形状と通信性能の関係を調査している．集団通信は多くの科学技術計算で多用され，プログラムの性能に与える影響も大きいため，より高速な実装が求められる．一般に集団通信の実装アルゴリズムは複数用意され，予め決められたメッセージサイズやプロセス数等の閾値に従って選択されることが多いが，前述の通り多次元メッシュ/トーラストポロジでは通信性能がプロセス配置によって変動するため，メッセージサイズやプロセス数が同じでも最適なアルゴリズムが異なる場合がある．また，大規模なスーパーコンピュータ上で想定しうる全てのプロセス配置形状について閾値を設定するのも非現実的である．そのため，プロセスの配置形状に応じて集団通信アルゴリズムの性能を見積もる技術が重要である．

そこで本稿では，そのような技術の開発に向けた予備調査として，京コンピュータや Fujitsu PRIMEHPC FX10 (以下 FX10) で用いられている 6 次元メッシュ/トーラスインターコネクト Tofu を対象として，プロセス配置形状が集団通信アルゴリズムの性能に与える影響を計測し，解析する．まず，同じプロセス数に対して，プロセスを配置する形状を変化させ，それによる性能の変化を計測する．さらに，通信衝突による影響を示す指標として，衝突によって発生する転送待ち時間を用い，プロセス配置形状と通信衝突，集団通信性能の関係を解析する．

2. 集団通信アルゴリズム

集団通信とは，並列プログラムの全ランクが参加してデータの集約や，1 対全，全対全のデータコピー等を行う，科学技術計算で多用される定型の通信パターンである．一般に集団通信は，通信ライブラリの中で，一対一通信等のより基本的な通信を組み合わせて実装される．この実装に用いられるアルゴリズム，すなわち集団通信アルゴリズムとして，様々なものが提案されている．それらの多くは，ネットワークの種類やトポロジに依存しない汎用的なものだが [1], [2]，特定のネットワークを対象としたものもある [3], [4], [5]．Tofu インターコネクト向けにも，いくつかの集団通信について専用のアルゴリズムが用意されている [6]．しかし，基本的にどのアルゴリズムにも長所と短所があり，使用される状況に応じて最適なものは変わる．

本稿では，集団通信のうち特に利用頻度が高く，並列プログラムの性能への影響も大きい，Allgather，Allreduce，Alltoall の実装アルゴリズムについて，Tofu インターコネクトにおけるプロセス配置の形状による性能の変動を計測し，解析する．今回の実験では，Tofu インターコネクト専用アルゴリズムに加え，Tofu インターコネクト用 MPI ライブラリのベースとなっている OpenMPI [8] で提供されているアルゴリズムについても性能を計測した．ただし，実行時のメッセージサイズやプロセス数でアルゴリズムを変えているものについては，その切り替えの閾値が明らかになっていないため，基本的に今回の調査の対象外とした．

以下，調査対象の各集団通信について概要と使用したアルゴリズムを説明する．なお，文中のパラメータ P はプロセス数を示す．また，「ランク」は一般に各プロセスに一意に割り当てられた番号を示すが，以下の説明では簡略化して，「そのランク番号を持つプロセス」のこともランクと表記する．

2.1 Allgather

Allgather は，各プロセスが所有するデータを相互にコピーし合い，最終的に全プロセスのデータをランクの順番に並べた配列を，全プロセスが所有するようにするものである．この通信は，多くの並列プログラムで計算結果の結合に用いられる．

Bruck Alltoall 向けに開発された同名のアルゴリズムを Allgather に適用したものである．各ステップ $i(0 \leq i \leq \lceil \log_2 P \rceil - 1)$ で，ランク r は，ランク $r - 2^i$ から受信し，ランク $r + 2^i$ へ送信する．その際，各プロセスはそれまで自分が受信した全てのデータを送信するため，データ転送量は元々各プロセスが所有していたデータ量 m に対して $m \times 2^i$ となる．

Recursive Doubling 各ステップ $i(0 \leq i \leq \log_2 P)$ で，2^i 離れたランク同士がデータを交換する．Bruck と同様，それまでに自分が受信した全てのデータを送信するため，データ転送量は $m \times 2^i$ となる．なお，OpenMPI の現在の実装では，P が 2 のべき乗で無い場合 Bruck アルゴリズムが呼ばれる．

Ring 各ステップ $i(0 \leq i \leq P - 1)$ で，ランク r は，ランク $r-1$ から，元々ランク $r-i-1$ が所有していたデータを受信し，ランク $r+1$ へ，元々ランク $r-i$ が所有していたデータを送信する．

Neighbor Exchange 各ステップ $i(0 \leq i \leq P/2 - 1)$ で，ランク r はランク $r+1$ もしくはランク $r-1$ とデータを交換する．データ交換する相手は，i が偶数の場合と奇数の場合で切り替える．これにより，双方向の Ring アルゴリズムを実現している．なお，OpenMPI の現在の実装では，P が偶数で無い場合 Ring アルゴリズムが呼ばれる．

Tofu Tofu インターコネクト専用アルゴリズムで，最大 4 個のネットワークインタフェースを同時に利用し，さらに 3 次元トーラス状の隣接通信のみで実装することによって衝突を回避する [6]．アルゴリズムの詳細は公開されていない．このアルゴリズムは，プロセスが配置された形状が 3 次元の直方体であり，送受信データが基本データ型で，各要素が 4 バイト境界に配置されており，1 要素のデータ量が約 64MB 以下の場合に利用できる．

2.2 Allreduce

Allreduce は，全プロセスが所有するデータに対して，任意の演算を適用して集約したものを，全プロセスが所有するようにするものである．この通信は，多くの並列プログラムで計算結果の集約に用いられる．

Basic Linear ランク 0 をルートとし，Reduce 通信で全プロセスが所有するデータを集約後，Broadcast 通信で全プロセスに結果をコピーする．Reduce 通信では，ルート以外のプロセスがルートに対して送信し，ルートはランク番号の逆順にデータを受信しながら演算を適用する．一方 Broadcast 通信では，ルートが全プロセスへの送信を発行し，ルート以外のプロセスがルートから受信する．

Recursive Doubling 各ステップ $i(0 \leq i \leq \log_2 P)$ で，2^i 離れたランク同士がデータを交換しながら演算を適用する．プロセス数が 2 のべき乗で無い場合は最初と最後に調整用のステップを追加する．

Ring 各プロセスは所有するデータをプロセス数分のブロックに分割し，各ブロックについてパイプライン転送を行う．各ステップ $i(0 \leq i \leq P-1)$ でランク r は，ランク $r-1$ から $r-i$ 番目のブロックの非ブロッキング受信を発行し，$r-i+1$ 番目のブロックの非ブロッキング受信の完了を待ち，その受信データと，自分の所有データの $r-i+1$ 番目のブロックに対して演算を適用し，その結果をランク $r+1$ に送信する．これにより，ランク r に r 番目のブロックの Allreduce の結果が格納されるので，最後に Ring アルゴリズムの Allgather で結合する．

Segmented Ring Ring アルゴリズムにおいて，各ブロックをさらに細かいセグメントに分け，セグメント毎にパイプライン転送を行うことにより，データ量が大きい Allreduce における転送バンド幅の向上を図る．なお，最後の Allgather はセグメントに分けず，ブロック単位で行う．

Tofu Tofu インターコネクト専用の Allreduce アルゴリズムで，Trinaryx3 と呼ばれている [7]．このアルゴリズムは，各プロセスが所有するデータを 3 等分し，それぞれについて別の 3 分木を用いて Reduce 通信と Broadcast 通信を行う．各 3 分木は，それぞれ 3 次元トーラス構造上の隣接ノードを接続して構成される．また，どの 3 分木も他の 3 分木の辺と重ならないように辺を選ぶため，通信の衝突を回避できる．このアルゴリズムは，プロセスが配置された形状が 3 次元の直方体であり，送受信データが基本データ型で，各要素が 4 バイト境界に配置されており，OP_MAXLOC，OP_MINLOC を除く定義済み演算を指定している場合に利用可能である．

2.3 Alltoall

Alltoall は，各プロセスが他のプロセスに対して，自分の所有するデータのうち相手のプロセスのランクに対応する部分のコピーを持たせるものである．この通信は，FFT(Fast Fourier Transform) や行列の転置等で頻繁に用いられる．

Basic Linear 各プロセスが，Alltoall 通信に必要な全ての受信と全ての送信を非ブロッキング通信命令で発行し，全通信が完了するのを待つ．

Pairwise 各ステップ $i(1 \leq i \leq P)$ で，各ランク r がランク $r-i$ からの受信とランク $r+i$ への送信を行うことにより，Alltoall を実現する．

Bruck 各ステップ $i(0 \leq i \leq \lceil \log_2 P \rceil - 1)$ で，ランク r は，ランク $r-2^i$ から受信し，ランク $r+2^i$ へ送信する．その際，各ランクはそれまでに自分が受信したデータのうち，送信先のランクが以降のステップで必要とするデータを全て送信する．毎ステップでの通信時にノード内でのメモリコピーが不要なように，アルゴリズムの最初と最後でデータをランク番号分シフトする．

Linear Sync Basic Linear で，一度に発行する非ブロッキング通信命令の数を，指定した値以下に抑えることにより，通信衝突の発生を低減する．

Tofu1，Tofu2 どちらも Tofu インターコネクト専用アルゴリズム [6] で，Tofu1 は比較的小データ量を想定して通信の衝突を許容する代わりに 4 個のネットワークインタフェースを同時に利用する．一方 Tofu2 は，データ量が大きい場合を想定し，通信の衝突を出来るだけ抑止する．アルゴリズムの詳細は公開されていない．どちらのアルゴリズムも，送受信データが基本データ型で，各要素が 4 バイト境界に配置されている必要がある．さらに Tofu2 は，プロセスが配置された形状が 3 次元の直方体であり，コミュニケータが MPI_COMM_WORLD で，1 要素のデータ量が約 32MB 以下の場合に利用できる．

3. Tofu インターコネクト

3.1 Tofu インターコネクトの概要

京コンピュータや FX10 で使用されている Tofu インターコネクトは，2 つの 3 次元メッシュ/トーラスネットワークを組み合わせた，X，Y，Z，A，B，C の 6 次元構造となっている [9]．このうち X，Y，Z 軸の長さは可変であるのに対し，A，B，C 軸の長さはそれぞれ 2，3，2 で固定されている．この A，B，C 軸による 12 ノードで構成される直方体を Tofu ユニットと呼ぶ．Tofu ユニット内では，B 軸のみが両端を接続したトーラス構造となっている．この Tofu ユニットを，各ノードが共通の X，Y，Z 座標を持つように，3 次元トーラス状に並べることにより，全体で 6 次元メッシュ/トーラス構造を形成する．

各ノードには，ネットワークインタフェース，ホストバス，ルータ等からなる ICC(Inter-Connect Controller) が用意されている．このうちネットワークインタフェースは，ホストバスを介して，そのノードが発行する送受信命令を処理する．このネットワークインタフェースが各ノードに 4 個ずつ搭載されているため，最大 4 方向同時に送受信することができる．ネットワークインタフェース 1 個あたりの理論通信バンド幅は 5GB/秒である．一方ルータは，Tofu インターコネクトで接続される 10 方向 (X+，X-，Y+，Y-，Z+，Z-，A，B+，B-，C) の隣接ノード接続用ポートと，自ノードの 4 個のネットワークインタフェース用ポートの，合計 14 ポート間でパケットのルーティングを行う．ルータの各ポートの理論通信バンド幅も 5GB/秒である．

ルータのポートのうち隣接ノード接続用の 10 個のポートには，4 個ずつの VC(Virtual Channel) があり，0 番目は通常パケット，1 番目はトーラスの日付変更線をまたいだり軸が変わったりする通信のパケット，あとの 2 個はリクエストの送信と応答のパケットが，それぞれ使用する．また，各 VC には 8KB の独立した受信バッファが用意されており，このバッファの残量を送信側に通知することで，フロー制御を行っている．

Tofu インターコネクトのルーティングは，3 つのステージで行われる．最初のステージで A，B，C 軸のいずれかの方向に 1 ホップ進み，次のステージで X，Y，Z 軸の順に辿って目的の Tofu ユニットに到達し，最後のステージで目的のノードまで A，B，C 軸を辿る．このうち最初のステージで進む方向は任意に指定することができ，これによって経路の選択肢を広げ，衝突の回避を図る．

3.2 Tofu インターコネクトの利用

Tofu インターコネクトでプログラムを実行する際のトポロジの形状は，ジョブ投入時に x×y×z の形で指定することが出来る．原則として Tofu インターコネクトでは，指定された形状の各軸に対して，物理的な 6 次元 X，Y，Z，A，B，C の軸を 2 つずつ組み合わせた XA，YB，ZC を割り当てる．ただし，故障ノードがあり，この対応ができない場合は，そのノードを迂回した割り当てを適用する．

4. Tofu インターコネクトにおける衝突による通信性能への影響の解析

4.1 Tofu インターコネクトにおける衝突による転送待ち時間

通信ネットワークにおいて，同じ経路を同時に複数の通信が使用する場合，通信衝突が発生する．特にトーラスやメッシュのようにノード数に対してリンク数が少ないネットワークでは，送信ノードおよび受信ノードが全く異なる独立した通信同士が同じ経路を使用することが多いため，通信衝突の発生頻度が高くなる．一般にネットワークで衝突が発生すると，関与した通信の全部，もしくは一部で性能の低下が起こるため，プログラムの性能解析やチューニングでは，通信衝突による性能への影響の解析が重要である．

Tofu インターコネクトにおいて通信衝突が発生した場合，性能への影響はルータにおける転送待ち時間として現れる．前節で説明したとおり Tofu インターコネクトでは，各ノードのルータは，隣接する 10 方向のポートの 4 個の VC それぞれについて，受信バッファの残量を送信側，すなわち各方向の隣接ノードに通知している．送信側は，この残量が 0 である間，その受信バッファに対する転送を待つ．この待ち時間が，通信所要時間の増加，すなわち実効通信バンド幅の低下の要因となる．受信バッファの残量が 0 になる要因は複数考えられるが，通信衝突によるものが主であるため，本稿ではこの待ち時間を通信衝突による性能低下の指標として用いる．

4.2 転送待ち時間情報の取得と集計

Tofu インターコネクトには，送信先の各 VC について，受信バッファの残量が 0 の状態である間の経過時間を累積する PA(Performance Analysis) 機能が用意されている [10]．利用者は，プログラム中の任意のコード区間について，この経過時間の累積値と，送受信パケット数，および転送バイト数を取得できる．以降，本稿では，この受信バッファが 0 である時間の累積値を，転送待ち時間と呼ぶ．なお，ここで取得できるのは隣接ノードとの通信に用いる 10 ポート分のみで，自ノードで発行される送受信が用いる 4 個のネットワークインタフェースについては転送待ち時間を取得できない．

本稿における実験では，集団通信を 1 回実行する区間に対して，転送待ち時間の情報を取得する．この転送待ち時間は，プロセス，ポート，VC それぞれについて個別に出

表 1　実験に使用したトポロジ形状

ノード数	使用したトポロジ形状
12	12×1×1，2×6×1，4×3×1，2×3×2
64	8×8×1，8×4×2，4×4×4
96	12×4×2，12×8×1，8×4×3，8×6×2
1200	20×15×4，16×15×5，8×15×10，16×12×10

力される．すなわち，集団通信 1 回の実行に対して，プロセス数 ×40 個の転送待ち時間情報が得られる．集団通信の実行時間に対するこれらの転送待ち時間の寄与の仕方は，アルゴリズムやプロセス配置の形状，ランクのノードへの割り当て等により異なる．本稿では，最も直接的に実行時間に寄与すると思われる値として，これらの 40 個の転送待ち時間のうちの最大値，すなわち最大待ち時間を，通信衝突の影響を示す指標として用いた．なお，各プロセス，各ポート，各 VC に対する転送待ち時間の分布や，それらの集団通信アルゴリズム性能への影響についての詳細な解析は，今後の課題である．

5. 実験

Tofu インターコネクトにおけるプロセス配置の形状による集団通信アルゴリズムの性能変動を調査するため，FX10 を用いた実験を行った．実験に用いたノード数は 12，64，96，1200 である．このうち 1200 ノードの実験は，東京大学情報基盤センターの Oakleaf-FX を用い，それ以外の実験は，九州大学情報基盤研究開発センターのシステム（以降，九大 FX）を用いた．実験を行った期間は，Oakleaf-FX が 2012 年 5 月 21 日〜23 日の大規模 HPC チャレンジ期間中，九大 FX が 2012 年 7 月〜8 月である．どちらのシステムも，計算ノードには Fujitsu SPARC64 IXfx (1.848GHz, 16 コア) を 1 基と，32GB のメモリを搭載しており，ノード間は Tofu インターコネクトで接続されている．

実験に用いたプログラムは，同じ集団通信を 5 回呼び出すプログラムで，それぞれについて所要時間を計測するとともに，前節で述べたとおり転送待ち時間を取得して，各呼び出しについて，プロセス，ポート，VC で最大の転送待ち時間を抽出した．プログラムは C 言語と MPI で書かれており，実行時のノードあたりプロセス数は 1 とした．

また，3.2 節で述べたとおり，利用者はジョブ投入時に，実行に用いるトポロジの形状を 1〜3 次元で指定することが出来る．今回の実験では，同じノード数に対して表 1 に示すように数通りずつの 1〜3 次元形状を指定した．さらに，同じ形状で 5 回ずつジョブを投入した．

以降の各節では，各アルゴリズムの所要時間と，最大転送待ち時間について，同じ形状で実行した 5 つのジョブの 5 回ずつの呼び出しによる計 25 回分の値の平均値を示す．なお，グラフの数が多いため，本文では Allgather のみの結果を示し，Allreduce，Alltoall の結果は付録に掲載した．

5.1 Allgather 12 ノード

図 1 に，12 ノード，各プロセスの所有データ量 2MB での，Allgather の各アルゴリズムによる所要時間の形状による変化を示す．形状は，一番左の 12×1×1 が最もバイセクションバンド幅が狭く，右に行くにつれて広くなる．Bruck は，バイセクションバンド幅と連動して，徐々に高速になっている．一方，Ring，Neighbor は 2×6×1 だけが他よりも遅い．Tofu 専用アルゴリズムは，1 次元，2 次元では他のアルゴリズムを用いているため，形状による単純な性能比較は出来ない．

一方，この場合の最大転送待ち時間を図 2 に示す．これらの 2 つの図を比較すると，各アルゴリズムについて，形状による最大転送待ち時間の変動の傾向が，所要時間の変動の傾向とほぼ一致することが分かる．そこで，各アルゴリズムで最も遅かった場合の形状と最も速かった場合の形状の間で，所要時間の差と最大転送待ち時間の差を比較する．Bruck の場合，最も遅かった 12×1×1 と最も速かった 2×3×2 の間の所要時間の差は約 0.01 秒であった．一方，同じ形状の組み合わせでの最大転送待ち時間の差は約 0.009 秒で，所要時間の差とほぼ一致する．他のアルゴリズムでも，Ring では所要時間の差が約 0.004 秒であるのに対し，最大転送待ち時間の差は約 0.006 秒，Neighbor では所要時間の差が約 0.003 秒であるのに対し，最大転送待ち時間の差は約 0.004 秒と，近い値を示している．

一方，プロセスあたりの所有データ量を 8KB とした場合の所要時間を図 3，最大転送待ち時間を図 4 にそれぞれ示す．データ量が少ない場合，所要時間に対して最大転送待ち時間が相対的に非常に小さい値となる．これが，各アルゴリズムで形状による性能の変動が見られない原因であると考えられる．

5.2 Allgather 64 ノード，96 ノード

ノード数を 64 とし，各プロセスの所有データ量を 2MB とした場合の，各アルゴリズムの所要時間と最大転送待ち時間の形状による変化を，図 5，6 にそれぞれ示す．なお，

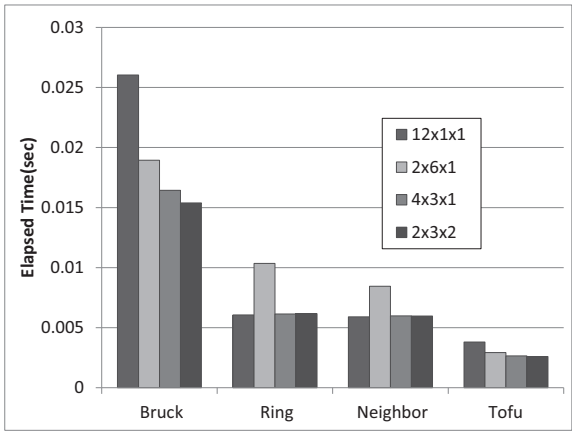

図 1　Allgather(12 ノード，2MB) の所要時間

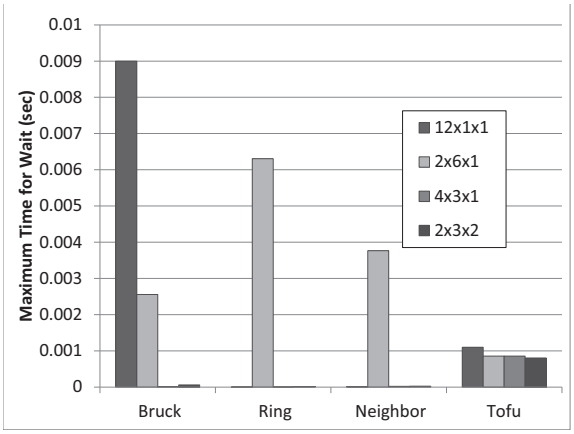

図 2　Allgather(12 ノード，2MB) の衝突による最大転送待ち時間

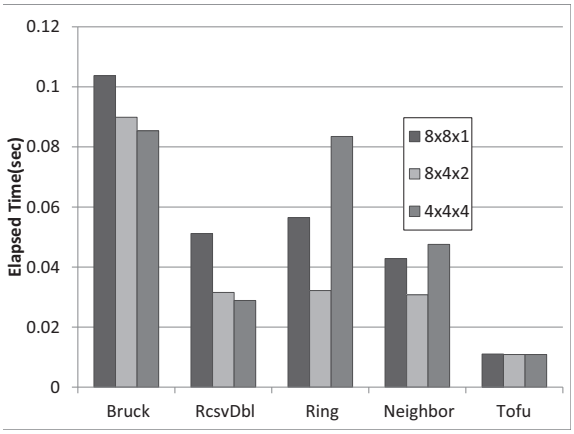

図 5　Allgather(64 ノード，2MB) の所要時間

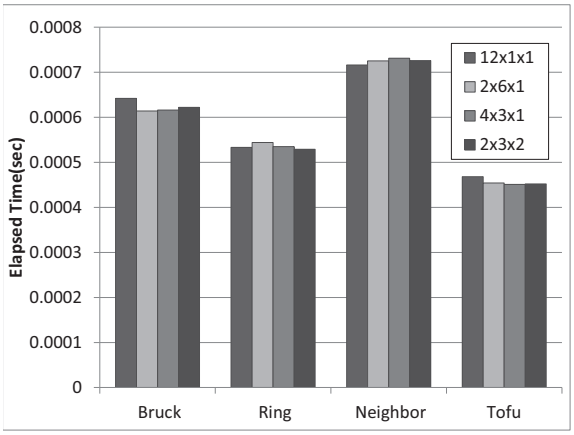

図 3　Allgather(12 ノード，8KB) の所要時間

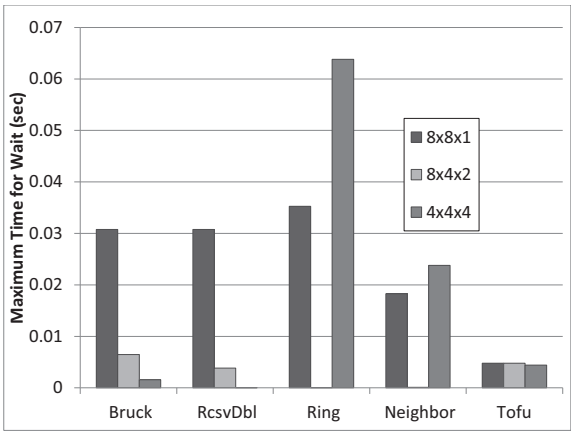

図 6　Allgather(64 ノード，2MB) の衝突による最大転送待ち時間

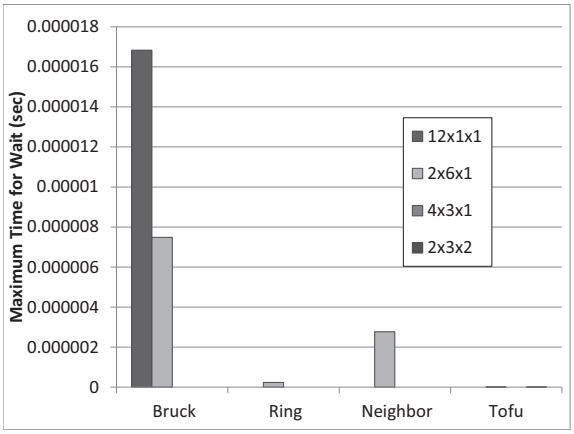

図 4　Allgather(12 ノード，8KB) の衝突による最大転送待ち時間

64 ノードの場合，Recursive Doubling (RcsvDbl) アルゴリズムも利用可能である．

12 ノードの場合と同様に，各アルゴリズムで形状による所要時間の変動と，最大転送待ち時間の変動は，ほぼ同じ傾向を示した．一方，最も遅かった形状と最も速かった形状の所要時間の差を各アルゴリズムについて計算すると，それぞれ Bruck 約 0.018 秒，RcsvDbl 約 0.022 秒，Ring 約 0.051 秒，Neighbor 約 0.017 秒であった．これに対し，最大転送待ち時間の差は，Bruck 約 0.029 秒，RcsvDbl 約 0.03 秒，Ring 約 0.064 秒，Neighbor 約 0.024 秒であり，所要時間の差との相違が大きいアルゴリズムがあった．このように，プロセス配置の形状による所要時間と最大転送待ち時間の変動について，傾向は近いものの絶対値が異なる理由としては，特定のリンクにおける衝突では無くトポロジ全体の衝突が影響している，ということが考えられる．

一方，ノード数を 96 とした場合の，各アルゴリズムの所要時間と最大転送待ち時間の形状による変化を，図 7，8 にそれぞれ示す．こちらも，形状による所要時間と最大転送待ち時間で，変動の傾向はほぼ同じであった．ただし，形状による性能の変動が見られたのは Bruck のみであった．Bruck における，最も遅い形状と最も速い形状での所要時間の変動と最大転送待ち時間の変動は，それぞれ約 0.2 秒と約 0.17 秒であった．

5.3　Allgather 1200 ノード

ノード数が 1200 で，各プロセスの所有データ量が 2MB と 2KB の場合の各アルゴリズムの所要時間の形状による変化を，図 9，10 にそれぞれ示す．この場合も，12，64，96 ノードの時と同様に形状による性能の変化が見られる．

なお，この計測を行った時点では Tofu インターコネクトの性能解析ツールを利用していなかったため，衝突によ

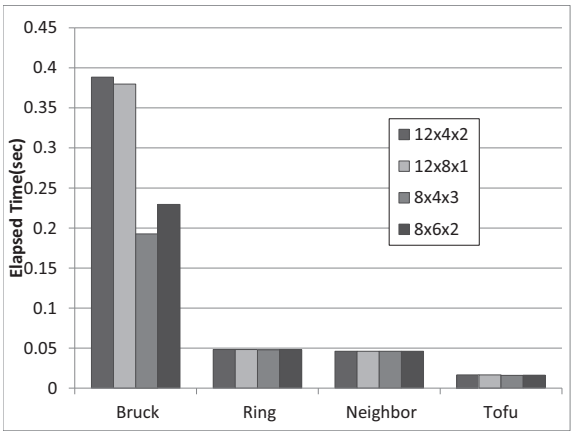

図 7　Allgather(96 ノード，2MB) の所要時間

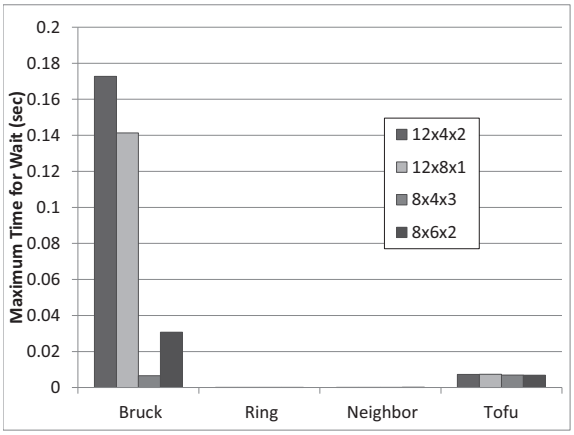

図 8　Allgather(96 ノード，2MB) の衝突による最大転送待ち時間

る影響の解析は次の機会が得られた場合に行う．

6. 考察

6.1 プロセス配置の形状と集団通信アルゴリズムの性能

今回の実験により，多くの場合でプロセス配置の形状によって集団通信アルゴリズムの性能が変動することが分かった．さらに，例えば図 5 のように，同じノード数でも形状によって最適なアルゴリズムが変動することがあるため，アルゴリズムの選択時に形状を考慮することの必要性

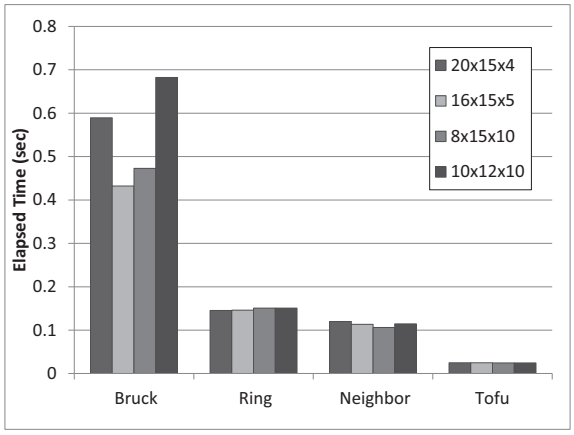

図 9　Allgather(1200 ノード，2MB) の所要時間

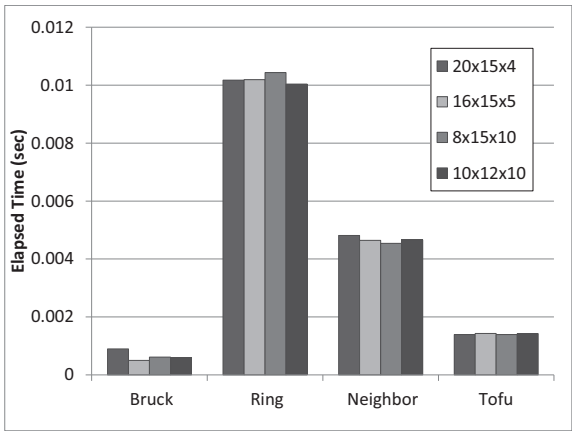

図 10　Allgather(1200 ノード，2KB) の所要時間

を示した．

ただし，形状に応じた性能の予測は単純では無い．ほとんどの場合で通信衝突による性能変動がアルゴリズムの性能変動に影響していることが分かったが，通信衝突の性能変動は，バイセクションバンド幅と同じ傾向を示すものもあれば，図 1 や図 5 の Ring，Neighbor のように全く異なる傾向を示すものもある．さらに，例えば付録の図 A·7 の Basic Linear のように，アルゴリズムの性能が衝突の性能と無関係に変動するものもある．

そのため，Tofu インターコネクト上でのアルゴリズムの性能を見積もるためには，プロセス配置の形状だけで無く，アルゴリズムの特性やノードへのランクの割り当ても加味する必要があると考えられる．

6.2 Tofu インターコネクト専用アルゴリズム

2 節で記述したとおり，Tofu インターコネクト専用アルゴリズムは，インターコネクトの形状を考慮し，通信衝突の回避や複数ネットワークインタフェースの活用等により高速化を図っている．これらのうち，特に衝突を回避する方式のアルゴリズムは，データ量が多い場合に非常に高速であることが，今回の実験で示された．一方，図 10 や付録の図 A·10 で示されているように，データ量が少ない場合は，既存のアルゴリズムの方が高速となる．そのため，各アルゴリズムの特性を活かし，状況に応じて適切にアルゴリズムを切り替えることが重要である．

なお，Tofu インターコネクト専用の Trinaryx3 アルゴリズムでは，基本的に隣接通信しか行わず，しかも経路はそれぞれ独立であるため，通信衝突が発生しないはずであるが，付録の図 A·5 等で示すとおり，転送待ち時間が発生している．この原因については，今後，転送待ち時間の分布の解析等により調査する予定である．

7. むすび

京コンピュータや Fujitsu PRIMEHPC FX10 で用いられている Tofu インターコネクトを対象として，実行時の

プロセス配置の形状による集団通信アルゴリズムの性能を解析した．実験の結果，プロセス配置の形状によって集団通信アルゴリズムの性能が変動することを示した．また，衝突によって生じる転送待ち時間の傾向と比較することにより，ほとんどの場合で，集団通信アルゴリズムの性能の変動が衝突の影響によるものであることを明らかにした．今後は，プロセス配置の形状だけでなく，アルゴリズムの特性，ノードへのランクの割り当て等も考慮した，詳細な性能解析を行う予定である．

参考文献

[1] Thakur, R., Rabenseifner, R. and Gropp, W.: Optimizing of Collective Communication Operations in MPICH, *Mathematics and Computer Science Division*, Argone National Laboratory, ANL/MCS-P1140-0304, (2004).
[2] Rabenseifner. R.: Optimization of Collective Reduction Operations, *International Converence on Computational Science*, LNCS 3036, pp. 1–9 (2004).
[3] Hamid, A. and Coddington, P.: Analysis of Algorithm Selection for Optimizing Collective Communication with MPICH for Ethernet and Myrinet Networks, *8th International Conference on Parallel and Distributed Computing*, Applications and Technologies, pp. 133–140 (2007).
[4] Zhang, P. and Deng, Y.: Design and Analysis of Pipelined Broadcast Algorithms for the All-Port Interlaced Bypass Torus Networks, *IEEE Transactions on Parallel and Distributed Systems*, (2012).
[5] Almasi, G., Heidelberger, P., Archer, C.J., Martorell, x., Chris Erway, C, Moreira, J.E., Steinmacher-Burow, B. and Zheng, Y.: Optimization of MPI collective communication on BlueGene/L systems, *Proceedings of the 19th annual international conference on Supercomputing*, pp. 253–262 (2005).
[6] FX10 利用者ガイド〜MPI 編〜, 講習会 PowerPoint 資料, 富士通株式会社, (2012).
[7] 松本 幸, 安達 知也, 住元 真司, 曽我 武史, 南里 豪志, 宇野 篤也, 黒川 原佳, 庄司 文由, 横川 三津夫, MPI Allreduce の「京」上での実装と評価, 先進的計算基盤システムシンポジウム (SACSIS2012) (2012).
[8] OpenMPI, `http://www.open-mpi.org`
[9] Ajima, Y., Inoue, T., Hiramoto, S. Shimizu, T. and Takagi, T.: The Tofu Interconnect *The 19th Annual Symposium on High-Performance Interconnects*, pp. 87–94 (2011).
[10] プロファイラ仕様手引書 (PRIMEHPC FX10 用) 富士通株式会社, (2012).

付　　録

A.1　Allreduce の実験結果

Allreduce の各アルゴリズムについて，プロセス配置形状に対する所要時間と最大転送待ち時間の変動を以下に示す．

A.1.1　Allreduce 12 ノード

ノード数が 12 で，データ量が 2MB の場合の結果を，図 A·1, A·2 に示す．Recursive Doubling(RcsvDbl)，Ring，Segmented Ring(RingSeg) では，最も速い形状と遅い形状の所要時間の差がそれぞれ約 0.0012 秒，約 0.00061 秒，約 0.00058 秒であったのに対し，最大転送待ち時間の差は，ぞれぞれ約 0.0017 秒，約 0.00067 秒，約 0.00070 秒となり，Allgather の場合と同様に形状による変動の傾向がほぼ同じとなった．ただし，Basic Linear は，どの形状でも転送待ち時間が 0 であったが，所要時間には形状による変動が見られた．この変動の原因はまだ分かっていない．なお，Tofu 専用アルゴリズムは 3 次元形状以外では内部で Recursive Doubling を用いている．

また，データ量が 8KB の場合の結果を図 A·3，A·4 に示す．これも Allgather の場合と同様に，データ量が少なくなり，形状による性能の変動が見られなくなっている．

A.1.2　Allreduce 64 ノード，96 ノード

データ量が 2MB で，ノード数が 64 ノードの場合と 96 ノードの場合の結果を，図 A·5，A·6，A·7，A·8 に示す．RcsvDbl において，形状による所要時間の変動が 64 ノードで約 0.0013 秒，96 ノードで約 0.0036 秒であったのに対し，最大転送待ち時間の変動は 64 ノードで約 0.0012 秒，96 ノードで約 0.0040 秒であり，ほぼ同じ傾向を示した．ただし，12 ノードの場合と同様に Basic Linear では，最大転送待ち時間とは異なる変動の傾向を示した．

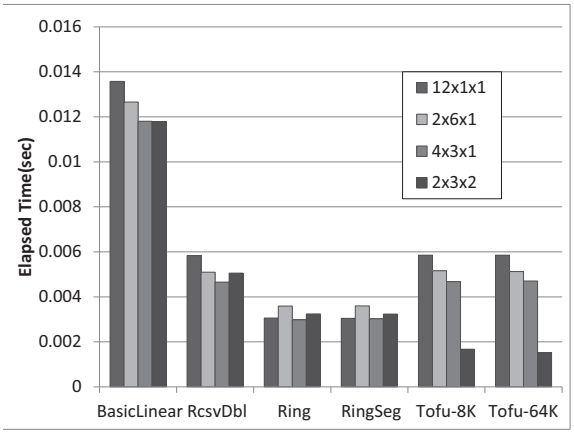

図 **A·1**　Allreduce(12 ノード，2MB) の所要時間

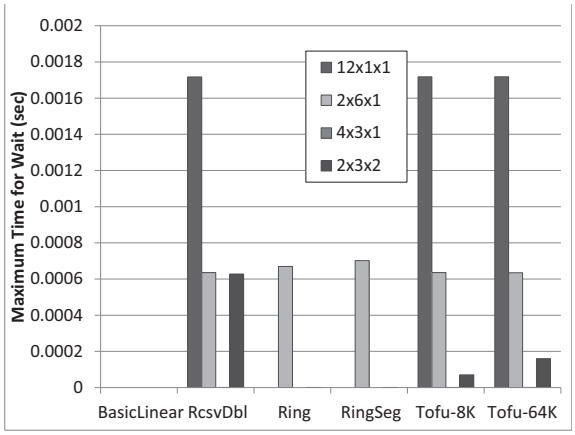

図 **A·2**　Allreduce(12 ノード，2MB) の衝突による最大転送待ち時間

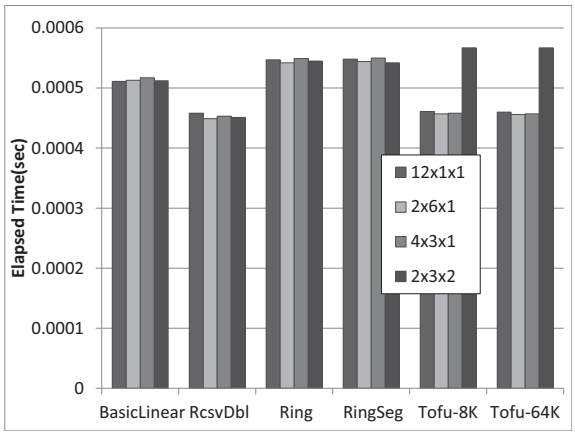

図 **A·3**　Allreduce(12 ノード，8KB) の所要時間

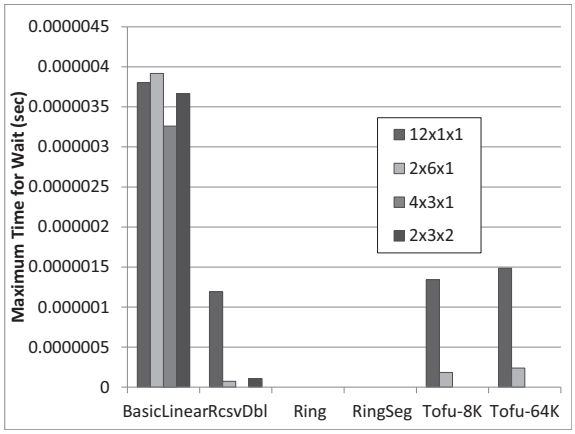

図 **A·4**　Allreduce(12 ノード，8KB) の衝突による最大転送待ち時間

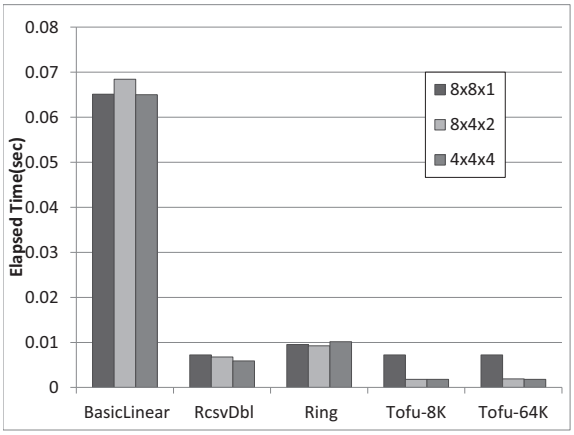

図 **A·5** Allreduce(64 ノード，2MB) の所要時間

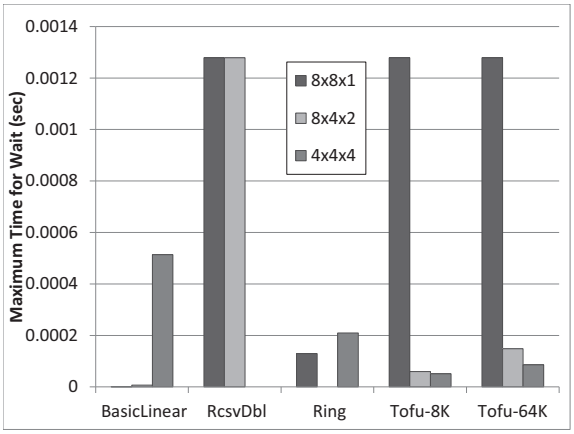

図 **A·6** Allreduce(64 ノード，2MB) の衝突による最大転送待ち時間

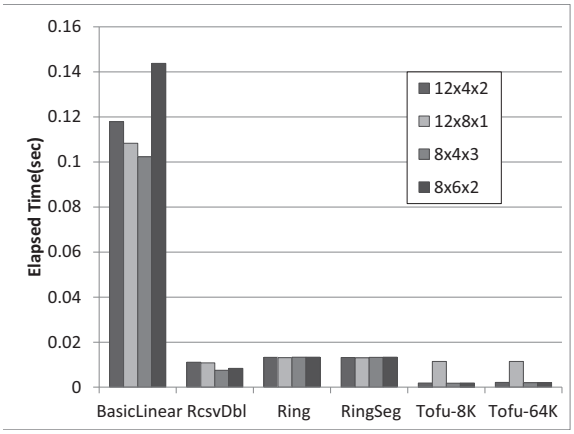

図 **A·7** Allreduce(96 ノード，2MB) の所要時間

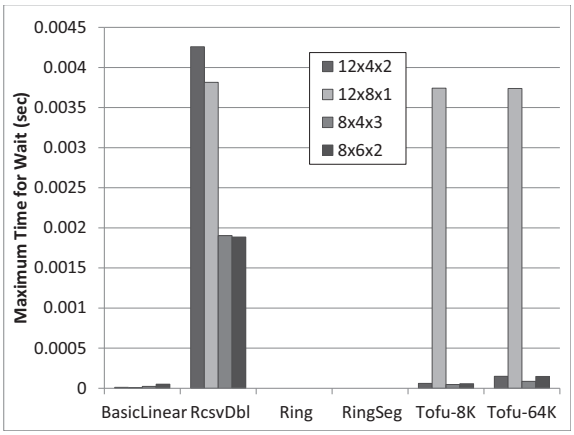

図 **A·8** Allreduce(96 ノード，2MB) の衝突による最大転送待ち時間

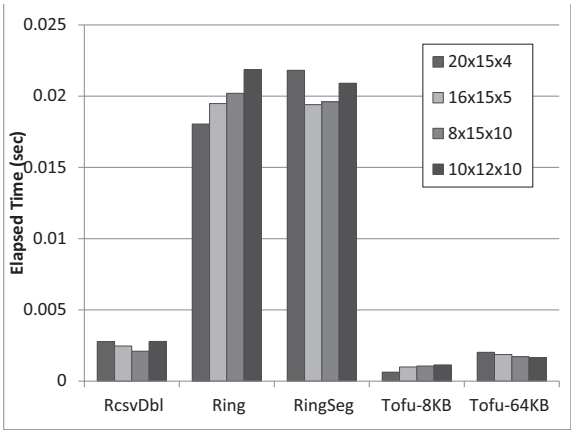

図 **A·9** Allreduce(1200 ノード，2MB) の所要時間

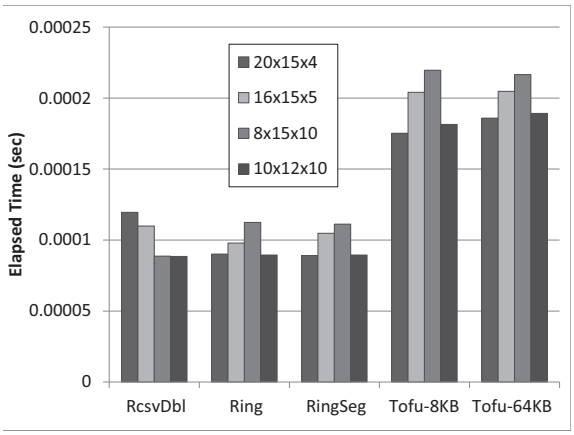

図 **A·10** Allreduce(1200 ノード，2KB) の所要時間

A.1.3 Allreduce 1200 ノード

ノード数が 1200 で，各プロセスの所有データ量が 2MB と 2KB の場合の各アルゴリズムの所要時間の形状による変化を，図 A·9，A·10 にそれぞれ示す．やはり，各アルゴリズムで形状による性能の変動が見られる．

A.2 Alltoall の実験結果

Alltoall の各アルゴリズムについて，プロセス配置形状に

対する所要時間と最大転送待ち時間の変動を以下に示す．

A.2.1 Alltoall 12 ノード

ノード数が 12 で，データ量が 2MB の場合の結果を，図 A·11，A·12 に示す．各アルゴリズムで，最も速い形状と遅い形状の所要時間の差が約 0.0055〜0.0087 秒であるのに対し，最大転送待ち時間の差がどれも約 0.007 秒となり，全体的な傾向もほぼ同じであった．

また，データ量が 8KB の場合の結果を図 A·13，A·14 に示す．これも Allgather の場合と同様に，データ量が少なくなり，形状による性能の変動が見られなくなっている．

A.2.2 Alltoall 64 ノード，96 ノード

データ量が 2MB で，ノード数が 64 ノードの場合と 96 ノードの場合の結果を，図 A·15，A·16，A·17，A·18 に示す．64 ノードでは，多くのアルゴリズムで形状による変動がほとんど見られなかった．一方 96 ノードでは，形状による所要時間と最大転送待ち時間の変動は，一見するとほぼ同じ傾向を示しているが，最も速い形状と遅い形状の差を計算すると，Basic Linear で所要時間の差が約 0.037 秒であったのに対し，最大転送待ち時間の差は約 0.020 秒，Pairwise で所要時間の差が約 0.094 秒であったのに対し，

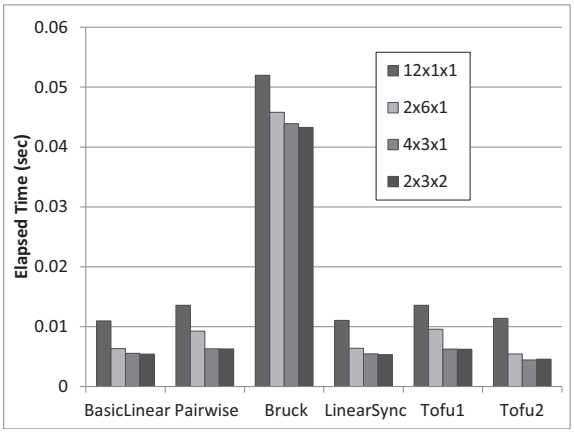

図 **A·11** Alltoall(12 ノード，2MB) の所要時間

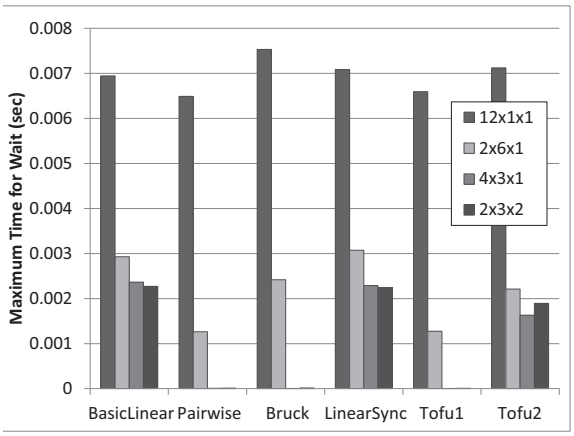

図 **A·12** Alltoall(12 ノード，2MB) の衝突による最大転送待ち時間

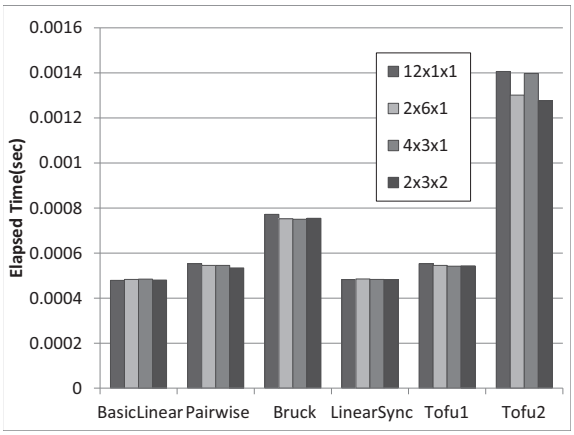

図 **A·13** Alltoall(12 ノード，8KB) の所要時間

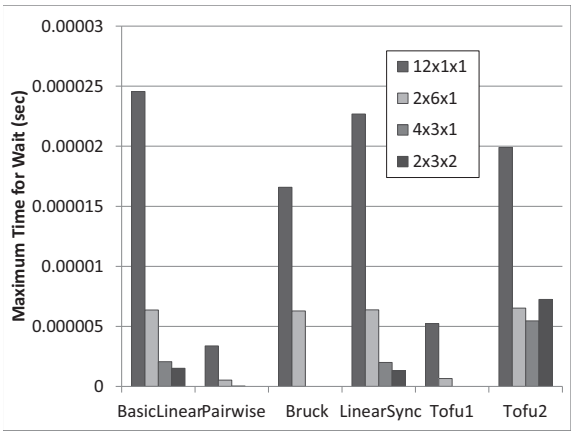

図 **A·14** Alltoall(12 ノード，8KB) の衝突による最大転送待ち時間

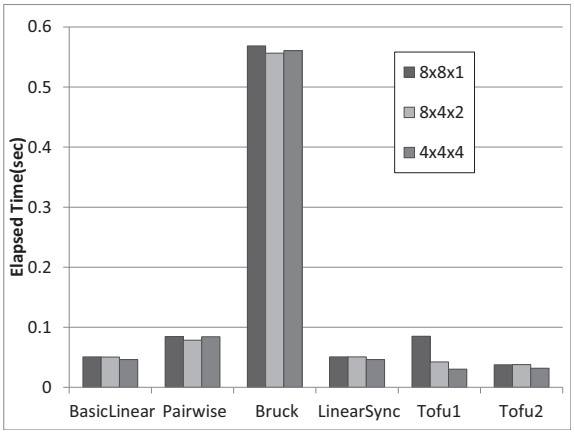

図 **A·15** Alltoall(64 ノード，2MB) の所要時間

最大転送待ち時間の差は約 0.018 秒と，値が大きく異なっている．これらのアルゴリズムでは，特定の経路では無く，全体的な衝突の影響が積算されている可能性があり，今後詳細な解析を行う予定である．

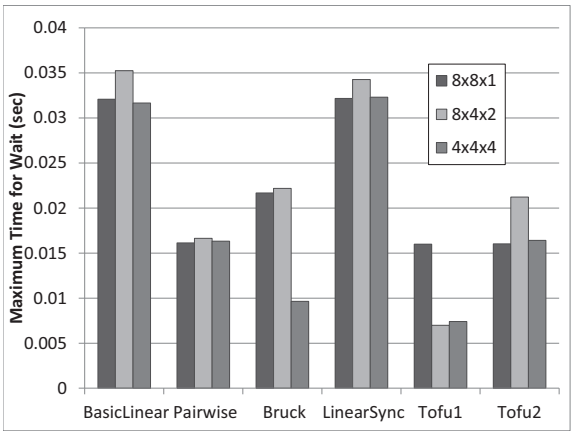

図 **A·16** Alltoall(64 ノード，2MB) の衝突による最大転送待ち時間

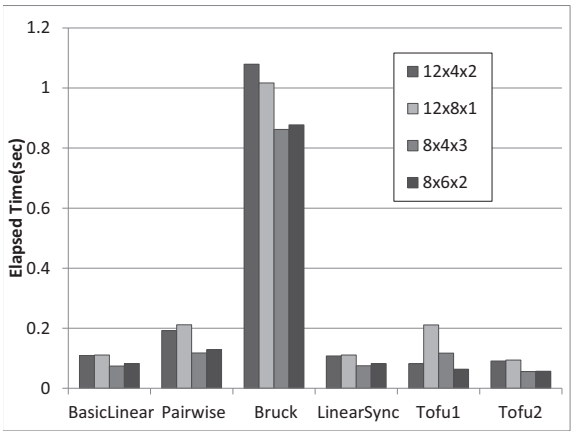

図 **A·17** Alltoall(96 ノード，2MB) の所要時間

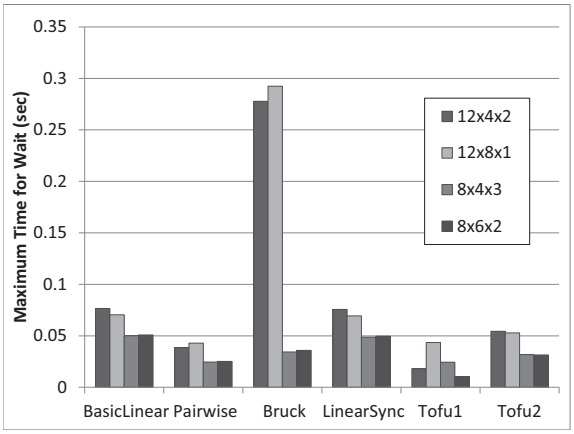

図 **A·18** Alltoall(96 ノード，2MB) の衝突による最大転送待ち時間

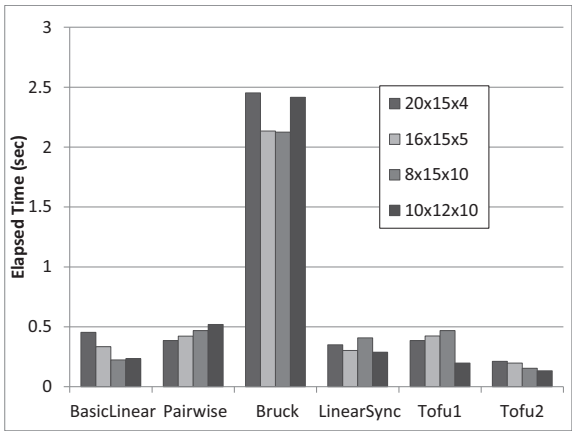

図 **A·19** Alltoall(1200 ノード，2MB) の所要時間

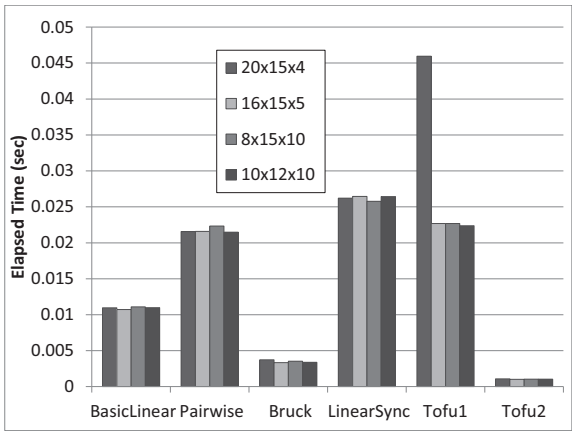

図 **A·20** Alltoall(1200 ノード，2KB) の所要時間

A.2.3 Alltoall 1200 ノード

ノード数が 1200 で，各プロセスの所有データ量が 2MB と 2KB の場合の各アルゴリズムの所要時間の形状による変化を，図 A·19，A·20 にそれぞれ示す．やはり，各アルゴリズムで形状による性能の変動が見られる．