情報処理学会論文誌　コンピューティングシステム　Vol. 3　No. 3　166–177 (Sep. 2010)

マルチコアPCクラスタ向け All-to-all通信アルゴリズムの提案と評価

成瀬　彰[†1]　中島耕太[†1]
住元真司[†1]　久門耕一[†1]

本稿では，マルチコア PC クラスタ上での All-to-all 通信性能の最適化について述べる．既存の All-to-all 通信アルゴリズム（Ring アルゴリズム）を使用してマルチコア PC クラスタで All-to-all 通信を行うと，シングルコア使用時と比較して実効通信バンド幅が低下する．All-to-all 通信中の詳細な挙動調査を実施した結果，その原因がネットワークスイッチ内で発生する Head-of-line（HoL）ブロッキングであることが明らかとなった．この HoL ブロッキング発生を回避する 2-Level Ring アルゴリズムを提案する．24 台の IA サーバを 1 台の InfinBand スイッチで接続したマルチコア PC クラスタ上で性能を評価した結果，2-Level Ring アルゴリズムで All-to-all 通信中の実効通信バンド幅は最大 24%向上することが分かった．さらに，NPB FT と IS の処理性能は最大 7%向上することを確認した．

A Proposal and Evaluation of All-to-all Algorithm for Multi-core PC Cluster Systems

Akira Naruse,[†1] Kohta Nakashima,[†1]
Shinji Sumimoto[†1] and Kouichi Kumon[†1]

In order to realize high network bandwidth during all-to-all communication on multi-core PC cluster system, an existing all-to-all algorithm (ring algorithm) is not optimum, even inadequate. According to our study, it is highly possible that using ring algorithm on multi-core PC cluster system causes Head-of-line (HoL) blocking in network switch. We propose 2-level ring algorithm that can reduce the chance of HoL blocking significantly. The experimental results show that 2-level ring algorithm realizes higher network bandwidth compared to ring algorithm by 24% at maximum and improves the performance of NPB FT and IS by 7% at maximum.

1. はじめに

プロセッサのマルチコア化にともない，マルチコアプロセッサ搭載サーバで構成されるマルチコア PC クラスタシステムが一般化しつつある．マルチコア PC クラスタシステム上で並列プログラムを実行する場合，各サーバ上にはコア数相当のプロセスが生成されることが多い．T2K オープンスパコン[1] のように，各サーバに複数のネットワークインタフェース（NIC）を搭載するシステムもあるが，一般的にはコア数と比べて NIC 数は少ない．したがって，マルチコア PC クラスタシステムでは，複数プロセス間で NIC 等のネットワーク資源が共有されることになる．

MPI[2] は多くの PC クラスタシステム向け並列プログラムで採用されている通信インタフェースであり，事実上の業界標準である．MPI では 1 対 1 通信や集合通信として様々な通信パターンが定義されている．1 対 1 通信には通信アルゴリズムによる性能向上の余地があまりないが，集合通信は通信アルゴリズム次第で性能が変動することが知られている．それは，集合通信では各プロセスが複数のプロセスとメッセージ送受を行うため，たとえば複数メッセージを束ねて通信回数を削減する等の最適化が可能だからである．そのため，これまでに様々な集合通信向け通信アルゴリズムが提案されている[7]．そして，近年はマルチコア PC クラスタシステムに適した通信アルゴリズムが関心を集めている[11]．

本稿では，集合通信の中でも通信負荷の重い All-to-all 通信（MPI_Alltoall）に着目する．特に，メッセージサイズの大きい All-to-all 通信をターゲットとする．メッセージサイズの大きい All-to-all 通信は通信バンド幅で性能が律速されることが多いため，高い実効通信バンド幅を実現する通信アルゴリズムが求められている．

一般に，メッセージサイズの大きい All-to-all 通信では Ring アルゴリズムが使用されることが多い．しかし，マルチコア PC クラスタ上で Ring アルゴリズムの性能を測定したところ，シングルコア使用時と比較して実効通信バンド幅が低下することが分かった．All-to-all 通信中の挙動を詳細に調査した結果，その原因はネットワークスイッチ内で発生する Head-of-line（HoL）ブロッキング[6] であることが明らかとなった．

マルチコア PC クラスタシステム上で Ring アルゴリズム使用時に HoL ブロッキングが発生するのは，あるサーバ上の複数プロセスが同時期にそれぞれ別サーバにメッセージを送

†1 株式会社富士通研究所
Fujitsu Laboratories Ltd.

信するためである．そこで，マルチコア PC クラスタ向けの All-to-all 通信アルゴリズムとして，2-Level Ring アルゴリズムを提案する．提案アルゴリズムでは，あるサーバ上の複数プロセスは同時期には同一のサーバにメッセージを送信するので，ネットワークスイッチ内で HoL ブロッキングが発生しない．

提案アルゴリズムの評価は，24 台の IA サーバを 1 台の DDR InfiniBand [12] ネットワークスイッチで接続したマルチコア PC クラスタシステム上で実施した．評価の結果，2-Level Ring アルゴリズムはつねに Ring アルゴリズムより性能が良いことが確認できた．具体的には，All-to-all 通信中の実効通信バンド幅は最大 24%向上し，NPB [5] FT と IS の処理性能は最大 7%向上した．

以下，2 章で All-to-all 通信の概要と Ring アルゴリズムに関して説明し，3 章でマルチコア PC クラスタシステムと Ring アルゴリズムの組合せで HoL ブロッキングが発生する仕組みを明らかにし，4 章でマルチコア PC クラスタシステム向け All-to-all アルゴリズムとして 2-Level Ring アルゴリズムを提案，5 章で 2-Level Ring アルゴリズムを評価し，6 章で関連研究について述べ，7 章でまとめる．

2. All-to-all 通信の概要と Ring アルゴリズム

本章では，All-to-all 通信の概要と，メッセージサイズが大きい場合に使用される All-to-all 通信アルゴリズムである Ring アルゴリズムについて説明する．

2.1 All-to-all 通信の概要

All-to-all 通信はすべてのプロセスがすべてのプロセスとメッセージ送受を行う集合通信である．各プロセス間で送受されるメッセージの内容がそれぞれ異なるため，バケツリレー的な通信最適化が難しいという特徴を持つ．All-to-all 通信は，メッセージサイズやネットワークトポロジにより適切なアルゴリズムが違うことが知られている．たとえば，メッセージサイズが小さい場合には通信遅延・通信回数がボトルネックになるが，メッセージサイズが大きい場合には通信バンド幅がボトルネックになる．そのため，これまでに様々なアルゴリズムが提案・評価されており [7]，状況に応じて適切なアルゴリズムを自動選択する研究も行われている [8]．

本稿では，All-to-all 通信の中でも通信バンド幅で性能が律速されるメッセージサイズが大きい場合をターゲットとする．メッセージサイズが大きい場合には Ring アルゴリズム [7] が使われることが多く，代表的な MPI 実装である OpenMPI [3] や MVAPICH [4] でも，メッセージサイズが大きい場合には Ring アルゴリズムが使用されている．

```
----------------------------------------
for (i = 0; i < Np, i++) {
   Ip_to   = (my_Ip + i + Np) % Np;
   Ip_from = (my_Ip - i + Np) % Np;
   Comm("send to Ip_to, rec from Ip_from");
}
//  Np: 総プロセス数
//  Ip: プロセス ID (0 <= Ip < Np)
----------------------------------------
```

図 1　Ring アルゴリズムの疑似コード
Fig. 1　Pseudo code of Ring algorithm.

2.2 Ring アルゴリズム

Ring アルゴリズムは，Np 回の通信ステップで構成されるアルゴリズムである（Np は総プロセス数）．**図 1** に Ring アルゴリズムの疑似コードを示す．あるプロセスのプロセス ID を my_Ip とすると，通信ステップ i（$0 \leq i < Np$）のとき，そのプロセスはプロセス ID が my_Ip + i のプロセスにメッセージを送付し，プロセス ID が my_Ip − i のプロセスからメッセージを受信する（プロセス ID はラップアラウンド）．したがって，Ring アルゴリズムには，各通信ステップにおいて，どのプロセスも複数のプロセスに同時にメッセージを送信することがなく，どのプロセスも複数のプロセスから同時にメッセージを受信することがない，という特徴がある．

3. マルチコア PC クラスタ上での All-to-all 通信性能調査と考察

本章では，マルチコア PC クラスタ上で Ring アルゴリズムで All-to-all 通信を行う場合に，シングルコア使用時と比べて実効バンド幅が低下する理由を明らかにする．

3.1 測 定 環 境

測定環境を**表 1** に示す．本稿で示す測定値は，すべてこの環境下で測定された結果である．測定環境は 24 台の IA サーバを InfiniBand ネットワークで接続したマルチコア PC クラスタである．各サーバは，2 ソケットの 2.93 GHz Xeon プロセッサ，24 GB の DDR3-1333 メモリ，DDR 対応の InfiniBand HCA を搭載している．計 24 枚の HCA は，すべて 1 台の InfiniBand スイッチに接続している．ある程度の規模の PC クラスタでは，Fat-tree [13] 等の多段スイッチネットワークが一般的である．しかし，多段スイッチネットワークでは，

表 1　測定環境
Table 1　Testing environment.

Servers	24 × 2-socket Nehalem server
CPU	2 × Intel Xeon X5570 (Nehalem 2.93 GHz)
M/B	SuperMicro X8DTT
Mem	24 GB (6 × 4 GB DDR3-1333 DIMM)
NIC	Mellanox MHGH29-XTC (DDR-IB, PCIe2)
OS	RHEL5.4 (64 bit)
MPI	OpenMPI 1.3.1
Compiler	Intel C/Fortran Compiler 11.0 (compile option: -O3)
Switch	Flextronics F-X430044 (DDR-IB, 24 ports)

$$Alltoall\_bandwidth = \frac{Msg \times (Np - Nlp) \times Nlp}{T}$$

`Msg`: 各プロセス対の送受メッセージサイズ
`Np`: 総プロセス数
`Nlp`: ローカルプロセス数
`T`: `All-to-all` 実行時間（秒）

図 2　All-to-all バンド幅の定義
Fig. 2　Definition of All-to-all bandwidth.

使用サーバ数やサーバの組合せ次第でネットワーク内でホットスポットが発生し，性能が変動することが報告されている[14]．多段スイッチネットワーク環境下ではネットワーク性能を定量的に評価するのが難しいため，本稿ではホットスポット等の影響を考慮する必要のない，単一スイッチに全サーバを接続したシステム構成で性能測定を行うこととする．

3.2　All-to-all 通信の性能指標

All-to-all 通信の性能指標には All-to-all バンド幅を用いる．All-to-all バンド幅の定義を**図 2** に示す．All-to-all 通信時のネットワーク利用効率を確認することが主目的であるため，サーバ内通信量は計算式から除外した．具体的には，各プロセスがネットワークに送出する総メッセージ量（$Msg \times (Np - Nlp)$）にローカルプロセス数（Nlp）を掛け，それを実行時間（T）で割った値を All-to-all バンド幅とした．

なお，実行時間（T）にはサーバ内通信時間も含まれており，この計算式による算出値は厳密には実効ネットワークバンド幅を示していない[⋆1]．しかし，メモリ帯域はネットワーク帯域と比べて十分に高速であり，サーバ内通信時間が実行時間に占める割合は十分に低いと考えられる．したがって，本稿では「All-to-all バンド幅 ≃ 実効ネットワークバンド幅」と見なすことにする．

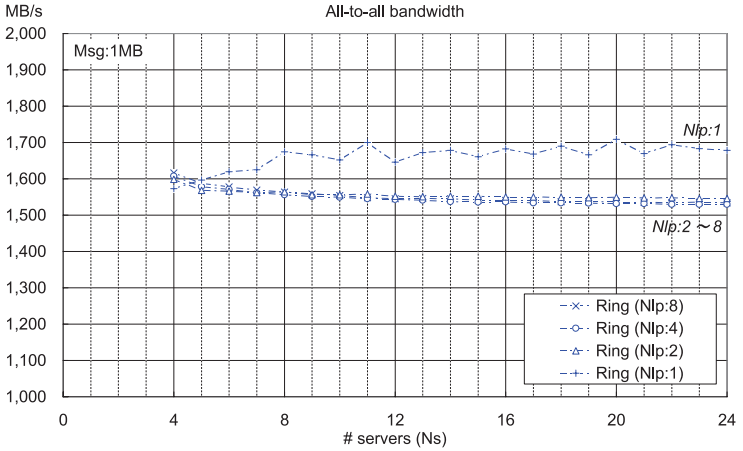

図 3　Ring アルゴリズムの All-to-all バンド幅測定結果
Fig. 3　All-to-all bandwidth measurement result of Ring algorithm.

3.3　Ring アルゴリズムの性能調査

Ring アルゴリズムとマルチコア PC クラスタシステムの適合性を確認するため，All-to-all 通信時の実効ネットワークバンド幅を測定した．**図 3** に測定結果を示す．図 3 は，横軸がサーバ数，縦軸が All-to-all 通信時のサーバあたり実効ネットワークバンド幅（以下，All-to-all バンド幅）を示している．測定条件は，サーバ数が 4〜24，サーバあたりプロセス数（以下，ローカルプロセス数）が 1〜8，各プロセス間の送受メッセージサイズは 1 MB である．

図 3 より，ローカルプロセス数 1 の場合の All-to-all バンド幅は 1,600〜1,700 MB/s 程度である．一方，ローカルプロセス数 2 以上の場合は 1,500〜1,600 MB/s 程度であり，ローカルプロセス数 1 の場合より性能が低い．ローカルプロセス数 2 以上の場合は，複数プロセスでネットワークを共有しており，それが性能低下の原因と考えることもできる．しかし，文献 11) によれば，複数プロセスで同時に通信を行った方がネットワーク利用効率は高くな

⋆1 特にサーバ数が少ないときは誤差が大きい．

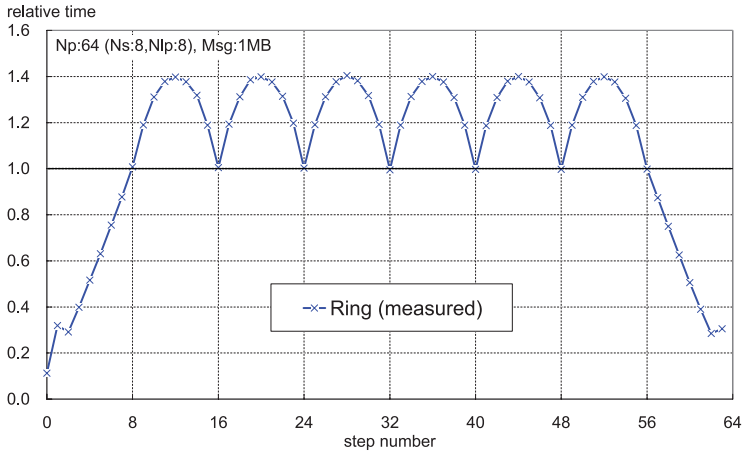

図 4　Ring アルゴリズムの各通信ステップの経過時間測定
Fig. 4　Measurement result of elapsed time in each communication step of Ring algorithm.

るとの結果が出ている．その結果に従えば，ローカルプロセス数 2 以上の場合の All-to-all バンド幅は向上するはずである．実際には All-to-all バンド幅は低下しており，これはマルチコア PC クラスタシステムと Ring アルゴリズムの組合せが不適切である可能性を示唆している．

3.4　Ring アルゴリズムの各通信ステップの実行時間調査

All-to-all 通信時の詳細状況を確認するため，Ring アルゴリズムの各通信ステップの実行時間を測定した．各通信ステップの実行時間測定は，通信ステップ間にバリア（MPI_Barrier()）を挿入した状態で実施し，通信ステップごとに，全プロセスの中で最も実行時間の長い結果をその通信ステップの実行時間とした．**図 4** に測定結果を示す．図 4 は，横軸が通信ステップ番号，縦軸が実行時間である．実行時間は相対時間であり，通信ステップ番号がローカルプロセス数の倍数のときの時間を基準に正規化している．測定条件は，総プロセス数が 64（サーバ数：8 × ローカルプロセス数：8），各プロセス間の送受メッセージサイズは 1 MB である．

本測定条件で Ring アルゴリズムを実行すると，通信ステップ番号が 8 以上 56 以下の場合は，どのプロセスも他サーバのプロセスとメッセージ送受を行う．つまり，通信ステップ番号がこの範囲内であれば，各サーバがネットワークに送出するメッセージ量はつねに等しいので，実行時間も等しくなることが期待される．しかし，図 4 より，実際には通信ステップごとに実行時間が違うことが分かる．具体的には，通信ステップ番号がローカルプロセス数の倍数の場合は実行時間が短く，それ以外の場合には実行時間が増加している．実行時間の違いには規則性があり，合理的な理由が存在する可能性が高い．

なお，通信ステップ番号が 8 未満 56 超の場合の実行時間が短いのは，一部もしくはすべてのプロセスがサーバ内通信を行っており，通信ステップ番号が 8 以上 56 以下の場合と比べてネットワークに送出するメッセージ量が少ないためである．以後，サーバ内通信を含む通信ステップは考慮の対象外とする．

3.5　ローカルプロセス数 2 以上での性能低下原因の考察

Ring アルゴリズムでローカルプロセス数 2 以上の場合に，All-to-all バンド幅が低下する原因を考える．図 4 より，通信ステップ番号がローカルプロセス数の倍数の場合は実行時間が短く，それ以外の場合は実行時間が長くなることが分かった．All-to-all 通信はプロセス単位でメッセージ送受が行われるが，プロセス単位ではなくサーバ単位で通信相手を考えると，両者には大きな違いがある．

通信ステップ番号がローカルプロセス数の倍数の場合，どのサーバも 1 つのサーバに対してのみメッセージを送信（もしくは受信）している．一方，それ以外の場合，どのサーバも 2 つのサーバに対してメッセージを送信（もしくは受信）している．後者の場合には，ネットワークスイッチ内で Head-of-Line（HoL）ブロッキング[6] が発生，パケット転送が遅延されて実行時間が増加している可能性がある．

HoL ブロッキングとは，ネットワークスイッチ内の出力ポート競合を契機に，該当出力ポートへのパケット転送がブロックされるだけでなく，その後続パケットもブロックされて，ネットワーク利用効率が低下する現象である．**図 5** に HoL ブロッキングのイメージ図を示す．図 5 は，サーバ B への出力ポートで競合が発生，それを契機にサーバ C からサーバ B へのパケットがブロックされ，さらに，その後続パケット（サーバ C からサーバ D へのパケット）までブロックされる様子を示している．

HoL ブロッキングが発生するのは，ある出力ポートに対して複数の入力ポートから同時にパケットを転送するときである．つまり，あるサーバに対して複数サーバからメッセージを送信するときには，ネットワークスイッチ内で HoL ブロッキングが発生する可能性がある．通信ステップ番号がローカルプロセス数の倍数でない場合，あるサーバに対して複数サーバから同時にメッセージを送信することがある．Ring アルゴリズムは，HoL ブロッキングの発生条件を満たしてしまう．

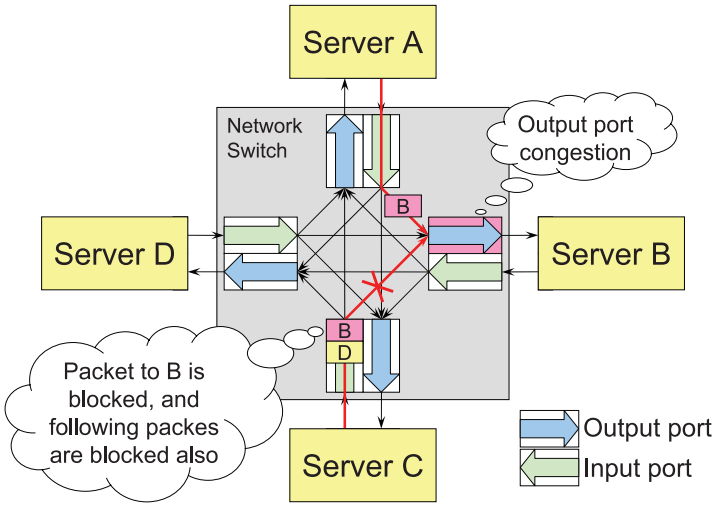

図 5　HoL ブロッキングのイメージ図
Fig. 5　Image of HoL blocking.

表 2　3 サーバ間のメッセージ送受比率例（$0 \leq \alpha \leq 1$）
Table 2　Message sending and receiving ratio among three servers.

Sender	Receiver		
	Server 0	Server 1	Server 2
Server 0	0	α	$1-\alpha$
Server 1	$1-\alpha$	0	α
Server 2	α	$1-\alpha$	0

3.6　HoL ブロッキングによる性能低下のモデル化と検証

HoL ブロッキングにより実行時間が具体的にどの程度増加するのかを考える．議論を簡略化するため，3 台のサーバで Ring アルゴリズムを実行，各サーバが他 2 サーバに対してメッセージを送付する通信ステップを事例とする．なお，このときの 3 台のサーバ間のメッセージ送受比率は**表 2** のとおりとする（$0 \leq \alpha \leq 1$）．

サーバ 0 がサーバ 1 にパケットを送出する確率は α である．一方，サーバ 2 もサーバ 1 に $1-\alpha$ の確率でパケットを送出するので，サーバ 1 への出力ポートでは $\alpha(1-\alpha)$ の確率で競合が発生する．このとき，サーバ 0 もしくはサーバ 2 のどちらかのパケットがブロックされることになる．どのパケットも優先度は同じと仮定すると，サーバ 0 が送出するパケットは，サーバ 1 への出力ポートで $\alpha(1-\alpha)/2$ の確率でブロックされる．同様に，サーバ 0 が送出するパケットは，サーバ 2 への出力ポートでも $\alpha(1-\alpha)/2$ の確率でブロックされる．このとき，サーバ 1 への出力ポート，サーバ 2 への出力ポートで同時に競合が発生することはないので，サーバ 0 が送出するパケットは合計で $\alpha(1-\alpha)$ の確率でブロックされることになる（パケットブロック率）．

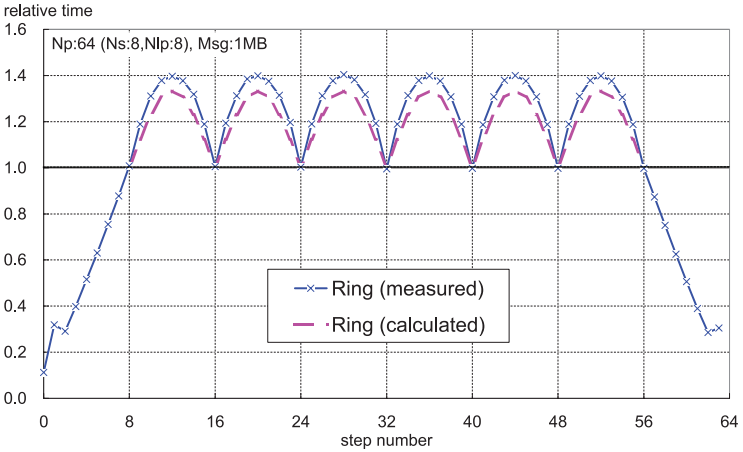

図 6　Ring アルゴリズムの各通信ステップの経過時間予測
Fig. 6　Calculation result of elapsed time in each communication step of Ring algorithm.

さらに，パケットブロック率が時系列に対して独立であると仮定すると，競合未発生時のパケット転送時間を基準とした平均パケット転送時間（Ta）は以下計算式で算出できる．

$$Ta = \frac{1}{1 - packet\_blocked\_ratio} = \frac{1}{1-\alpha(1-\alpha)}$$

この計算式に基づいて，総プロセス数が 64（サーバ数：8 × ローカルプロセス数：8）の場合に，Ring アルゴリズムの各通信ステップの実行時間を算出した結果を**図 6** に示す．図 6 は実線が実測結果，点線が算出結果を示している．HoL ブロッキングによる転送遅延を考慮した実行時間（算出値）は，実測値とおおむね一致していることが分かる．したがって，マルチコア PC クラスタで Ring アルゴリズムを使用する場合にネットワーク性能が低下する原因は，ネットワークスイッチ内で発生する HoL ブロッキングである可能性が非常に高

いといえる．

4. マルチコア PC クラスタ向け All-to-all アルゴリズムの提案

マルチコア PC クラスタ向け All-to-all 通信アルゴリズムとして 2-Level Ring アルゴリズムを提案する．Ring アルゴリズムは各通信ステップの通信相手をプロセス ID に基づいて決定するが，2-Level Ring アルゴリズムはサーバ ID とローカルプロセス ID に基づいて通信相手を決定する．**図 7** に 2-Level Ring アルゴリズムの疑似コードを示す．以下，2-Level Ring アルゴリズムの通信相手決定手順である．

(1)　［通信相手のサーバ ID の決定］
通信相手のサーバ ID は外部ループで決定する．外部ループ番号 j（0 ≤ j < Ns）のとき，あるプロセスのサーバ ID を my_Is とすると，そのプロセスはサーバ ID が my_Is + j のプロセス（Is_to）にメッセージを送付し，サーバ ID が my_Is − j のプロセス（Is_from）からメッセージを受信する（サーバ ID はラップアラウンド）．

(2)　［通信相手のローカルプロセス ID の決定］
通信相手のローカルプロセス ID は内部ループで決定する．内部ループ番号 k（0 ≤ k < Nlp）のとき，あるプロセスのローカルプロセス ID を my_Ilp とすると，そのプロセスはローカルプロセス ID が my_Ilp + k のプロセス（Ilp_to）にメッセージを送付し，ローカルプロセス ID が my_Ilp − k のプロセス（Ilp_from）からメッセージを受信する（ローカルプロセス ID はラップアラウンド）．

(3)　［通信相手のプロセス ID の決定］
通信ステップごとに一意に定まる通信相手のサーバ ID，ローカルプロセス ID に基づき，通信相手のプロセス ID（送信相手は Ip_to，受信相手は Ip_from）を決定する．

```
-----------------------------------------
for (j = 0; j < Ns, j++) {
   Is_to   = (my_Is + j + Ns) % Ns;
   Is_from = (my_Is - j + Ns) % Ns;
   for (k = 0; k < Nlp, k++) {
      Ilp_to   = (my_Ilp + k + Nlp) % Nlp;
      Ilp_from = (my_Ilp - k + Nlp) % Nlp;
      Ip_to    = (Is_to   * Nlp) + Ilp_to;
      Ip_from  = (Is_from * Nlp) + Ilp_from;
      Comm("send to Ip_to, rec from Ip_from");
   }
}
//  Ns:  サーバ数
//  Nlp: ローカルプロセス数
//  Np:  総プロセス数 (Np = Ns * Nlp)
//  Is:  サーバ ID (0 <= Is < Ns)
//  Ilp: ローカルプロセス ID (0 <= Ilp < Nlp)
//  Ip:  プロセス ID (Ip = Is * Nlp + Ilp)
-----------------------------------------
```

図 7　2-Level Ring アルゴリズムの疑似コード
Fig. 7　Pseudo code of 2-Level Ring algorithm.

図 8 に 2-Level Ring アルゴリズムによる具体的な通信例を示す．図 8 は，総プロセス数が 16（サーバ数：4 × ローカルプロセス数：4）の場合の事例である．図 8 上は，外部ループ番号 j が 1〜3 の場合のサーバ間の送受関係を示している．図 8 下は，外部ループ番号 j

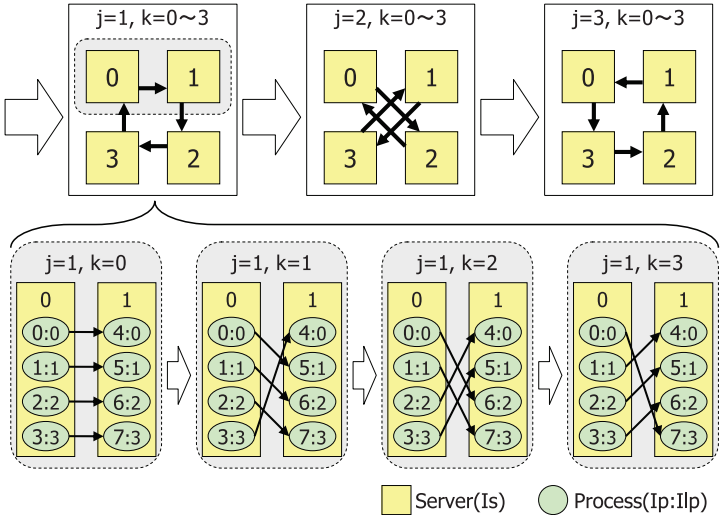

図 8　2-Level Ring アルゴリズムの通信例（総プロセス数：16，サーバ数：4，ローカルプロセス数：4）
Fig. 8　Example of communication pattern of 2-Level Ring algorithm.

が 1，内部ループ番号 k が 0〜3 のときに，ノード 0 の 4 プロセス（Ip:0〜3）とノード 1 の 4 プロセス（Ip:4〜7）の間の送受関係がどうなるかを示している．このように，2-Level Ring アルゴリズムでは，どの通信ステップにおいても各サーバの送信相手（もしくは受信相手）は，必ず 1 つのサーバに限定される．つまり，2-Level Ring アルゴリズムでは，あるサーバに対して同時に複数のサーバからメッセージを送信することがないので，原理上，ネットワークスイッチ内で HoL ブロッキングが発生しない．

なお，2-Level Ring アルゴリズムではサーバあたりのプロセス数が，どのサーバでも同じであることを仮定している．ある程度の並列数の MPI ジョブに対しては，サーバ単位で計算資源を割り当てることが多く，この仮定は一般的には妥当と考えられる．しかし，計算資源の割当てがたとえばコア単位で行われ，サーバあたりのプロセス数がサーバ間で異なる場合には 2-Level Ring アルゴリズムを適用できない．

5. 2-Level Ring アルゴリズムの評価

本章では，マルチコア PC クラスタ上での 2-Level Ring アルゴリズムの挙動・性能を，表 1 の測定環境で評価した結果について述べる．

5.1 各通信ステップの実行時間評価

2-Level Ring アルゴリズムの各通信ステップの実行時間測定結果を**図 9** に示す[⋆1]．図 9 は，横軸が通信ステップ番号，縦軸が実行時間を示している．実行時間は相対時間であり，通信ステップ番号がローカルプロセス数の倍数のときの時間を基準に正規化している．測定条件は，総プロセス数は 64（サーバ数：8 × ローカルプロセス数：8）で，各プロセス間の送受メッセージサイズは 1 MB である．

Ring アルゴリズムでは，通信ステップ番号がローカルプロセス数の倍数でない場合に実行時間が長くなる現象が観測されたが，2-Level Ring アルゴリズムではそのような時間増加は観測されなかった．期待どおりの結果である．

5.2 各種サーバ数での All-to-all バンド幅評価

各種サーバ数（4〜24），各種ローカルプロセス数（1〜8）で Ring アルゴリズムと 2-Level Ring アルゴリズムの性能を測定した結果を**図 10** に示す．図 10 は，横軸がサーバ数，縦軸が All-to-all バンド幅を示している．メッセージサイズは 1 MB である．

⋆1 2-Level Ring アルゴリズムの通信ステップ番号 i は，図 7 の 2 つのループ変数 j，k から式 i = j * Nlp + k で算出される値とした．

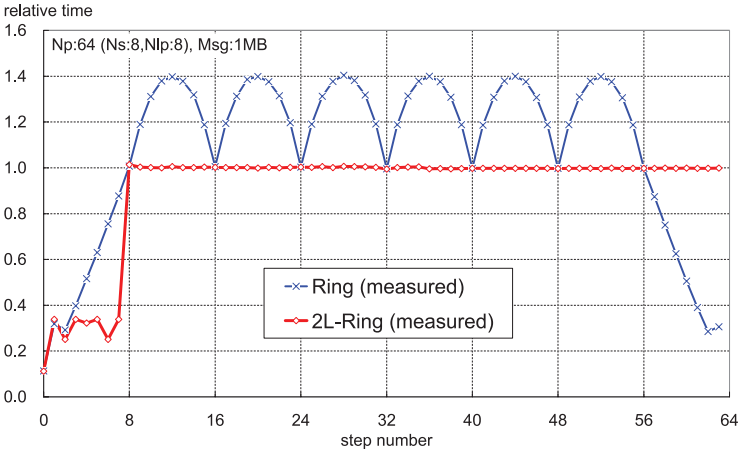

図 9　2-Level Ring アルゴリズムの各通信ステップの経過時間測定結果
Fig. 9　Measurement result of elapsed time in each communication step of 2-Level Ring algorithm.

図 10 より，Ring アルゴリズムではローカルプロセス数が複数のときに All-to-all バンド幅が低下する現象が観測されたが，2-Level Ring アルゴリズムでは反対に All-to-all バンド幅が向上することが分かる．文献 11) の結果と同様の傾向を示しており，妥当な結果といえる．

最も性能向上率が高いのは，総プロセス数：192（サーバ数：24 × ローカルプロセス数：8）の場合である．Ring アルゴリズムでは 1,534 MB/s だった All-to-all バンド幅が，2-Level Ring アルゴリズムでは 1,904 MB/s に向上している．性能向上率は 24.1%である．通信バンド幅の理論上限は 2,000 MB/s であり（ネットワークは DDR-IB），このときのネットワーク利用効率は 95%を超えていたことになる．

なお，ローカルプロセス数 1 のときに，Ring アルゴリズムと 2-Level Ring アルゴリズムの性能が一致しているが，これは，ローカルプロセス 1 の場合，両アルゴリズムは等価だからである．

5.3 各種メッセージサイズでの All-to-all バンド幅評価

各種メッセージサイズ（10 KB〜1 MB）で Ring アルゴリズムと 2-Level Ring アルゴリズムの性能を測定した結果を**図 11** に示す．図 11 は，横軸がメッセージサイズ，縦軸が All-to-all バンド幅を示している．サーバ数は 24 である．

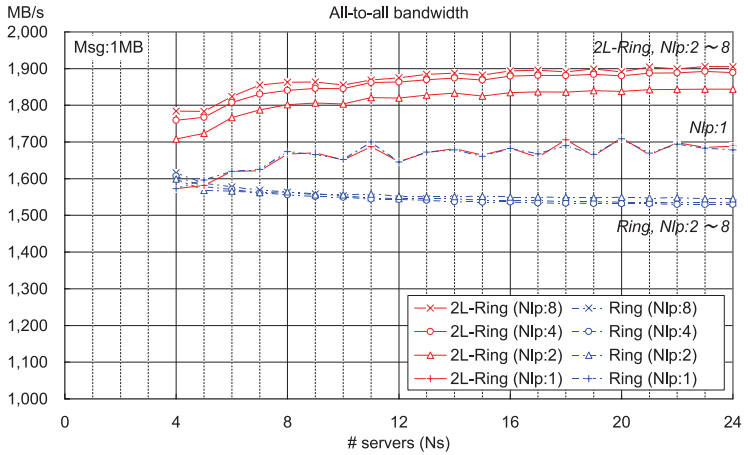

図 10　各種サーバ数での All-to-all バンド幅測定結果
Fig. 10　All-to-all bandwidth measurement result at various server counts.

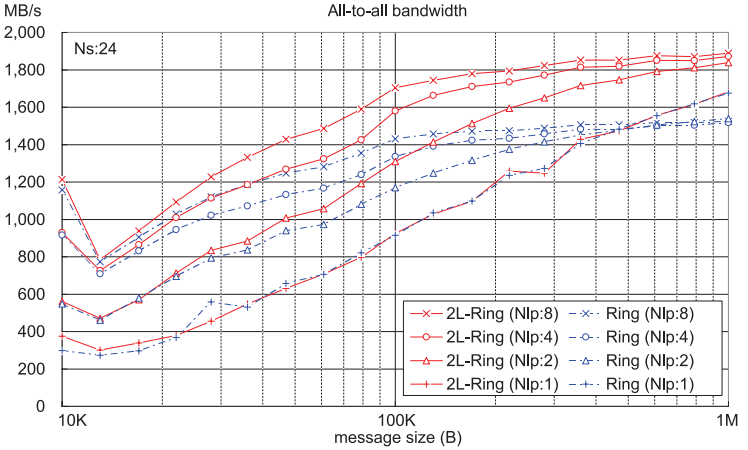

図 11　各種メッセージサイズでの All-to-all バンド幅測定結果
Fig. 11　All-to-all bandwidth measurement result at various message size.

図 11 より，メッセージサイズが数十 KB を超えると，ローカルプロセス数 2 以上の場合の All-to-all バンド幅が向上することが分かる．2-Level Ring アルゴリズムは，ネットワークバンド幅ネックの場合に有効な方法であり，メッセージサイズが大きいときに効果が大きいのは妥当な結果といえる．

注目すべきは，どのメッセージサイズでも Ring アルゴリズムと比較して性能が低下しないことである．また，2-Level Ring アルゴリズムにはメモリ使用量が増える等の副作用もない．したがって，これまで Ring アルゴリズムを選択していたケースであれば，安全に 2-Level Ring アルゴリズムを使用することができる．

5.4　SA アルゴリズムとの性能比較

各種メッセージサイズ（10 KB〜1 MB）で Ring アルゴリズム，2-Level Ring アルゴリズム，SA（Send-side Aggregation）アルゴリズム[11] の性能を測定した結果を**図 12** に示す．図 12 は，横軸がメッセージサイズ，縦軸が All-to-all バンド幅を示している．サーバ数は 24，ローカルプロセス数は 8 である．

SA アルゴリズムは，2-Level Ring アルゴリズム同様，マルチコア PC クラスタ向けの All-to-all 通信アルゴリズムである[11]．SA アルゴリズムは，主にメッセージサイズが小さい場合を対象としたアルゴリズムであり，サーバ間通信の回数を削減できることが大きな特長である．具体的には，Ring アルゴリズムや 2-Ring リングアルゴリズムでは，プロセスあたり $(Ns-1)\times Nlp$ 回のサーバ間通信が必要だが，SA アルゴリズムではそれが $Ns-1$ 回となり，サーバ間通信回数は $1/Nlp$ に減少する．それにともないサーバ間通信 1 回あたりのメッセージサイズは Np 倍に大きくなるので，元々のメッセージサイズが小さい場合にも相対的に高いバンド幅を得やすいという特長を持っている．

一方，SA アルゴリズムではサーバ間通信を行う前に，同一サーバ内のプロセス間で通常より多量のメッセージを交換する必要がある．Ring アルゴリズムや 2-Level Ring アルゴリズムでは，プロセスあたりのサーバ内通信量は合計で $Msg\times(Nlp-1)$ であるが，SA アルゴリズムではそれが $Msg\times(Nlp-1)\times Ns$ となる．つまり，SA アルゴリズムではサーバ内通信量が増加するため，サーバ内のメモリコピー負荷が大きくなるという問題がある．

図 12 より，メッセージサイズが約 30 KB 以下の場合には SA アルゴリズムの性能が良いことが分かる．これは，SA アルゴリズムにより，サーバ間通信回数が減り，それと同時にサーバ間通信 1 回あたりのメッセージサイズが大きくなった効果と考えられる．

一方，メッセージサイズが約 30 KB を超える場合には 2-Level Ring アルゴリズムの性能が良いことが分かる．SA アルゴリズムは Ring アルゴリズム程度の性能にとどまった．SA アルゴリズムも，2-Level Ring アルゴリズムと同様に，ノードレベルでは Ring アルゴリズ

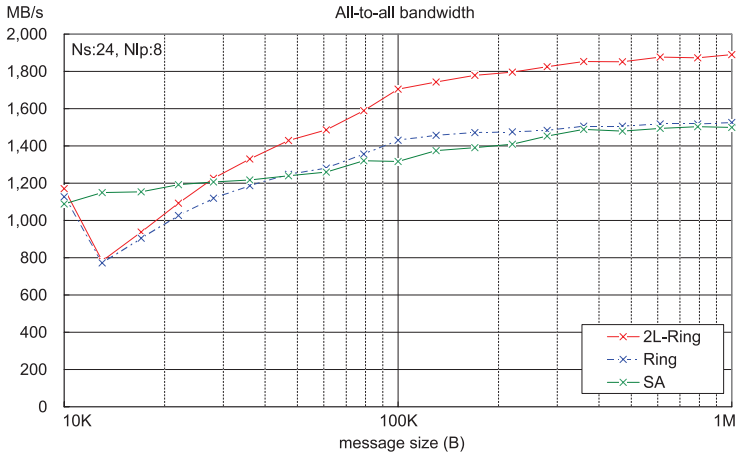

図 12　各種メッセージサイズでの SA アルゴリズムとの性能比較結果
Fig. 12　All-to-all bandwidth comparison of 2-Level Ring algorithm and SA algorithm.

ムに相当するアルゴリズムであり，HoL ブロッキング発生を回避できるという特徴を持っている．2-Level Ring アルゴリズムと SA アルゴリズムの実行時間内訳を確認した結果，メッセージサイズが十分に大きい場合のノード間通信時間は，どちらもほぼ同じであることが分かった．SA アルゴリズムでも HoL ブロッキング発生を回避できていたことになる．したがって，メッセージサイズが大きいときの 2-Level Ring アルゴリズムと SA アルゴリズムの性能差は，サーバ内のメモリコピー負荷が原因である．

SA アルゴリズムはメッセージサイズが小さい場合に有効であり，2-Level Ring アルゴリズムはメッセージサイズが大きい場合に有効であることが確認できたといえる．

5.5　NPB FT/IS での性能評価

NPB [5] で 2-Level Ring アルゴリズムの性能を評価した結果について説明する．評価には，MPI 版の NPB 3.3 に含まれる 9 種のプログラムのうち，All-to-all 通信（MPI_Alltoall(v)）を使用する FT と IS を用いた．問題サイズは 3 種（B，C，D），総プロセス数は 128（サーバ数：16 × ローカルプロセス数：8）とした．評価環境の全 24 サーバを使用しないのは，FT と IS はどちらも総プロセス数が 2 べき数に限定されているためである．

表 3 に測定結果（実行時間と処理性能 (Gop/s)）を示す．表 3 の項目 alltoall msg size は，All-to-all 通信時の各プロセス対の送受メッセージサイズ（平均）を示している．表 3 より，2-Level Ring アルゴリズムにより FT の処理性能は 6.1〜7.1%，IS の処理性能は 1.9〜7.0%向上することが分かる．

表 3　NPB FT/IS の測定結果（総プロセス数：128）
Table 3　Measurement result of NPB FT and IS.

	alltoall msg size	Total time (sec) Ring	2L-ring	Gop/s Ring	2L-ring	
FT						
B	32 KB	1.109	1.177	78.21	82.99	(+6.1%)
C	128 KB	4.691	4.380	84.49	90.49	(+7.1%)
D	2,048 KB	117.2	110.5	76.46	81.14	(+6.1%)
IS						
B	8 KB	0.112	0.110	2.988	3.046	(+1.9%)
C	32 KB	0.486	0.461	2.760	2.912	(+5.5%)
D	512 KB	8.551	7.991	2.511	2.687	(+7.0%)

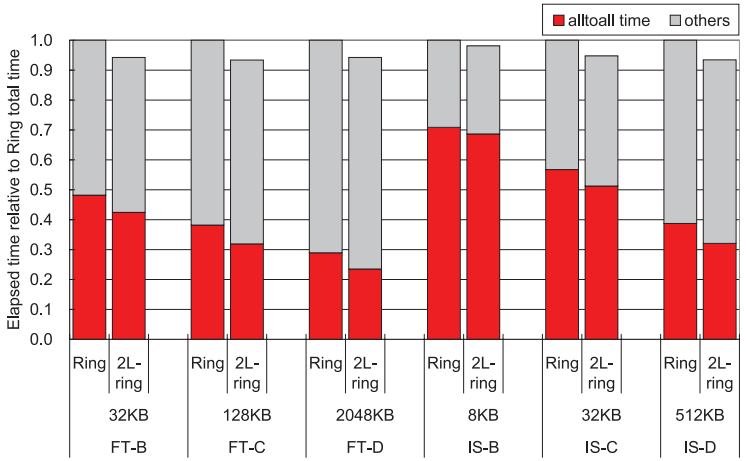

図 13　NPB FT/IS の実行時間内訳（総プロセス数：128）
Fig. 13　Execution time breakdown of NPB FT and IS.

図 13 に実行時間の内訳（All-to-all 通信時間とそれ以外の時間）を示す．図 13 の縦軸は相対時間であり，Ring アルゴリズム使用時の実行時間を基準に正規化している．図 13 より，2-Level Ring アルゴリズムにより All-to-all 通信時間は減少しているが，それ以外の時間はほぼ一定である．したがって，FT と IS の処理性能向上は All-to-all 通信時間減少の効

果であるといえる．

2-Level Ring アルゴリズム使用による All-to-all 通信時間の減少率に着目すると，All-to-all 通信時の送受メッセージサイズが小さい場合には減少率が小さく，大きい場合には減少率が大きい．このメッセージサイズと All-to-all 通信時間減少率との関係は，図 11 に示す各種メッセージサイズでの All-to-all 通信の素性能測定結果とおおむね一致している．したがって，All-to-all 通信を使用する他アプリでも，2-Level Ring アルゴリズムを使用することで図 11 に相当する性能向上（All-to-all 通信時間減少）を期待できる．

6. 関 連 研 究

文献 9) では，All-to-all 通信時の HoL ブロッキングによる性能低下が指摘されているが，多段ネットワークにおける上位・下位スイッチ間でのリンク競合を解消するルーティング方式が主テーマであり，ネットワークスイッチ内の HoL ブロッキング競合を回避する All-to-all 通信アルゴリズムは考慮されていない．

文献 15) と 16) では，1 サーバあたりのプロセス数が複数の場合の All-to-all 通信アルゴリズムとして階層型アルゴリズムが言及，提案されている．これら階層型アルゴリズムの特徴は，ある時間におけるサーバ間通信を行うプロセスを，1 サーバあたりたかだか 1 つに制限していることである．その目的は，NIC が複数プロセスで同時に使用されることでネットワーク性能が低下するのを回避することにあると思われる．これらの階層型アルゴリズムでも，結果的に，ネットワークスイッチ内での HoL ブロッキングは回避される．しかし，現在のプラットフォームでは，複数プロセスが同時に通信を行った方がネットワーク利用効率が高くなることが報告されており[11]，図 10 と図 11 の結果もそれを支持している．つまり，従来の階層型アルゴリズムは現在のプラットフォーム特性に適しておらず，2-Level Ring アルゴリズム相当の実効ネットワークバンド幅の実現は難しい．

文献 10) では，マルチコア向け All-to-all 通信アルゴリズムが提案されているが，サーバ内通信の最適化を目的としたものであり，ネットワーク利用効率の向上に関しては考慮されていない．

文献 11) では，マルチコア向け All-to-all 通信アルゴリズムとして SA（Send-side Aggragation）アルゴリズムが提案されている．SA アルゴリズムは，サーバ間通信を行う前にサーバ内プロセス間でデータ交換することで，サーバ間通信回数を削減することが大きな特長である．SA アルゴリズムはネットワークスイッチ内の HoL ブロッキング回避を意図したアルゴリズムではないが，結果的に HoL ブロッキングを回避することができる．しかし，図 12 に示したとおり，SA アルゴリズムはメッセージサイズが小さい場合に有効なアルゴリズムであり，メッセージサイズ大きい場合にはサーバ内のメモリコピー負荷が増えるため Ring アルゴリズム程度の性能にとどまることが確認された．また，SA アルゴリズムには，サーバ間通信前のサーバ内プロセス間のデータ交換用に，追加でバッファ（メモリ）が必要になるという短所がある．

我々の知る限り，ネットワークスイッチ内で発生する HoL ブロッキングに着目し，HoL ブロッキングの発生を回避してネットワーク利用効率を高めるシンプルかつ実用的なマルチコア PC クラスタシステム向け All-to-all 通信アルゴリズムは，これまでのところ提案・評価されていない．

7. ま と め

本稿では，マルチコア PC クラスタシステム上で従来アルゴリズム（Ring アルゴリズム）を使用して All-to-all 通信を行うと，ネットワーク利用効率が低下するという問題を調査し，その原因がネットワークスイッチ内で発生する Head-of-line（HoL）ブロッキングであることを明らかにした．

この HoL ブロッキング発生を回避するマルチコア PC クラスタシステム向け All-to-all 通信アルゴリズムとして 2-Level Ring アルゴリズムを提案した．提案アルゴリズムの評価は 24 台の IA サーバを 1 台の DDR InfiniBand ネットワークスイッチで接続したマルチコア PC クラスタシステム上で実施した．評価の結果，2-Level Ring アルゴリズムで最大 1,904 MB/s の All-to-all バンド幅（ネットワーク利用効率 95%超）が実現可能であり，Ring アルゴリズムと比較して最大 24%の性能向上が得られることを確認した．また，All-to-all 通信をともなう NPB FT と IS は，提案アルゴリズムにより全体の処理性能が最大 7%向上することを確認し，2-Level Ring アルゴリズムの有効性を明らかにした．

2-Level Ring アルゴリズムは通信バンド幅ネックの場合に効果のある方法であり，メッセージサイズが小さく通信遅延ネックの場合には大きな効果を期待できない．しかし，どのメッセージサイズでも Ring アルゴリズムの性能を下回ることはなかった．またメモリ使用量が増える等の副作用もないので，これまで Ring アルゴリズムを選択していたケースであれば，安全に 2-Level Ring アルゴリズムを使うことができる．

本稿では，24 台のサーバを 1 台の DDR InfiniBand スイッチに接続したマルチコア PC クラスタ上で 2-Level Ring アルゴリズムの有効性を確認した．ネットワークスイッチ内の HoL ブロッキングを回避する 2-Level Ring アルゴリズムは，多数のスイッチから構成され

る Fat-Tree 等の多段スイッチ構成ネットワークでも有効と考えられるが，定量的な評価が必要である．今後，多段スイッチ構成ネットワークでの評価に加え，InfiniBand 以外のネットワークでも評価を行う予定である．

また，現在の 2-Level Ring アルゴリズムは，サーバあたりのプロセス数が同じであることを仮定している．システムの運用方針によっては，サーバあたりのプロセス数が異なる場合も考えられるので，その場合にも適用可能なアルゴリズムを検討する予定である．

参 考 文 献

1) T2K Open Super Computer. http://www.open-supercomputer.org/
2) The Message Passing Interface (MPI) standard. http://www.mpi-forum.org/
3) OpenMPI. http://www.open-mpi.org/
4) MVAPICH. http://mvapich.cse.ohio-state.edu/
5) NAS Parallel Benchmark (NPB). http://www.nas.nasa.gov/Software/NPB/
6) Karol, M., Hluchyj, M. and Morgan, S.: Input Versus Output Queueing on a Space-Division Packet Switch, *IEEE Trans. Communications*, Vol.35, Issue 12, pp.1347–1356 (1987).
7) Faraj, A. and Yuan, X.: Automatic generation and tuning of MPI collective communication routines, *19th Annual international Conference on Supercomputing* (2005).
8) 南里豪志，Mamadou, H.N.，Gu, F.L.，村上和彰：性能モデルによる予測を併用した Alltoall アルゴリズム動的選択技術の評価，情報処理学会研究報告会，Vol.2008-HPC-118, pp.73–78 (2008).
9) Geoffray, P. and Hoefler, T.: Adaptive Routing Strategies for Modern High Performance Networks, *16th IEEE Symposium on High Performance Interconnects* (2008).
10) Mamidala, A.R., Kumar, R., De, D. and Panda, D.K.: MPI Collectives on Modern Multicore Clusters: Performance Optimizations and Communication Characteristics, *2008 8th IEEE International Symposium on Cluster Computing and the Grid* (2008).
11) Kumar, R., Mamidala, A. and Panda, D.K.: Scaling alltoall collective on multi-core systems, *IEEE International Symposium on Parallel and Distributed Processing* (2008).
12) InfiniBand Trade Association: InfiniBand Architecture Specification, Release 1.2 (Oct. 2004).
13) Petrini, F. and Vanneschi, M.: k-ary n-trees: High Performance Networks for Massively Parallel Architectures, *11th International Parallel Processing Symposium* (*IPPS97*) (1997).
14) Hoefler, T., Schneider, T. and Lumsdaine, A.: Multistage switches are not crossbars: Effects of static routing in high-performance networks, *2008 IEEE International Conference on Cluster Computing* (2008).
15) Huse, L.P.: MPI Optimization for SMP Based Clusters Interconnected with SCI, *7th European PVM/MPI User's Gropu Meeting* (2000).
16) Sanders, P. and Traff, J.L.: The Hierarchical Factor Algorithm for All-to-All Communication, *8th International Euro-Par Conference on Parallel Processing* (2002).

(平成 22 年 1 月 26 日受付)
(平成 22 年 5 月 15 日採録)

成瀬　彰（正会員）

1996 年名古屋大学大学院工学研究科修了（情報工学専攻）．同年富士通（株）入社．（株）富士通研究所にて IA サーバ，PC クラスタシステムに関わる研究開発に従事．計算機アーキテクチャ，並列処理，性能最適化に興味を持つ．平成 21 年度情報処理学会山下記念研究賞受賞．SACSIS2010 最優秀論文賞受賞．

中島　耕太（正会員）

2000 年九州大学工学部電気情報工学科卒業．2002 年同大学大学院システム情報科学府情報工学専攻修士課程修了．同年富士通（株）入社．博士（工学）．現在，（株）富士通研究所勤務．高速通信機構に関する研究開発に従事．SACSIS2010 最優秀論文賞受賞．

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（株）富士通入社．（株）富士通研究所にて並列オペレーティングシステム，並列分散システムソフトウェアの研究開発に従事．1997 年より新情報処理開発機構に出向．コモディティネットワークを用いた高速通信機構の研究開発，RWC SCore3 クラスタ等大規模 PC クラスタ開発に従事．2002 年より（株）富士通研究所にて高速通信機構の研究開発，理研スーパーコンバインドクラスタ，PACS-CS 等大規模 PC クラスタ開発等に従事，2007 年より富士通（株）と（株）富士通研究所に所属，PFLOPS 級のスーパコンピュータと PC クラスタ向けのソフトウェア開発に従事．超大規模並列分散システムのアーキテクチャ，システムソフトウェア等に興味を持つ．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（慶應義塾大学大学院理工学研究科）．

久門　耕一（正会員）

1979 年東京大学工学部電気工学科卒業．1981 年同大学大学院電子工学専門課程修士課程修了．1984 年同大学院博士課程中退．同年（株）富士通研究所入社．現在，同社取締役（同社 IT システム研究所所長を兼務）．CPU，メモリ，並列計算機アーキテクチャに関する研究に従事．GCC，Linux カーネル等の改良にも興味を持つ．SACSIS2010 最優秀論文賞受賞．