Vol.2009-ARC-186 No.4
Vol.2009-HPC-123 No.4
2009/11/30

マルチコアPCクラスタ向けAll-to-all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 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、高いネットワーク性能が得られない。All-to-all 通信中の詳細な挙動調査により、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になった。更に、マルチコア PC クラスタ向け All-to-all アルゴリズムとして 2-Level Ring アルゴリズムを提案し、24 台の 2way Nehalem サーバを InfiniBand ネットワークで接続したマルチコア PC クラスタ上での評価の結果、2-Level Ring アルゴリズムにより All-to-all 通信中の実効ネットワークバンド幅が最大 24%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.

1. はじめに

プロセッサのマルチコア化に伴い、マルチコアプロセッサを搭載した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的になったきた。マルチコア PC クラスタシステムで並列プログラムを実行する場合、各サーバ上では複数のプロセスが動作することが多い。T2K オープンスパコン[1] のように、各サーバにネットワークインタフェース (NIC) を複数搭載するシステムもあるが、一般的にコア数の方が NIC 数より多く、1 プロセス/1 コア状態で全プロセスがサーバ間通信を行う場合には、ネットワーク資源は複数プロセスにより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に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。特に、複数プロセス間のメッセージ送受を伴う集合通信は、メッセージサイズやネットワークトポロジーなど様々な要因で最適な通信手順が異なるため、これまで集合通信を実現する様々な通信アルゴリズムが提案されており[6]、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[10]。

本稿では、集合通信の中でも All-to-all 通信 (MPI_Alltoall) に着目する。特に、ネットワークバンド幅で性能が律速され、通信アルゴリズムによる最適化の困難なメッセージサイズが大きい場合の All-to-all 通信をターゲットとする。マルチコア PC クラスタ上で既存の All-to-all 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、高いネットワーク性能が得られないが、All-to-all 通信中の詳細な挙動調査により、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[5] であることが明らかになった。

マルチコア PC クラスタシステムと Ring アルゴリズムの組み合わせで 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それぞれ別サーバ上の別プロセスにメッセージを送信 (もしくは受信) すること、つまり、あるサーバが同時に複数サーバにメッセージを送信 (もしくは受信) することが原因である。そこで、マルチコア PC クラスタ向けの All-to-all アルゴリズムとして、あるサーバが同時に複数のサーバにメッセージを送信 (もしくは受信) することのない、2-Level Ring アルゴリズムを提案する。2-Level Ring アル

†1 富士通研究所
Fujitsu Laboratories

Vol.2009-ARC-186 No.4
Vol.2009-HPC-123 No.4
2009/11/30

ゴリズムによりネットワークスイッチ内での HoL ブロッキング発生が回避される。

2-Level Ring アルゴリズムの性能を、24 台の 2way Nehalem サーバを DDR InfiniBand ネットワークで接続したマルチコア PC クラスタシステム上で評価した結果、2-Level Ring アルゴリズムは常に Ring アルゴリズムより性能が良く、All-to-all 通信中の実効ネットワークバンド幅は最大 24%向上することを確認した。

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 アルゴリズムの組み合わせで 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 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 通信は全てのプロセスが全てのプロセスとメッセージ送受を行う集合通信であり、各プロセス間で送受されるメッセージの内容はそれぞれ独立であり、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ーにより適切なアルゴリズムが違うことが知られている。例えば、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、様々なアルゴリズムが提案・評価されており[6]、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[7]。

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[6] が使われることが多く、代表的な MPI 実装である OpenMPI[3]・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。

2.2 Ring アルゴリズム

Ring アルゴリズムは、総プロセス数を Np とすると、Np 回の通信ステップで構成されるアルゴリズムである。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≤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)。従って、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。図 1 に Ring アルゴリズムの疑似コードを示す。

```
-------------------------------------------------------------------------
for (i = 0; i < Np, i++) {
    Ip_to   = (my_Ip + i + Np) % Np;
    Ip_from = (my_Ip - i + Np) % Np;
    Communication( "send msg to Ip_to and receive msg 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

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 通信を行う場合に高いネットワーク性能が得られない理由を明らかにする。

3.1 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、表 1 に示す測定環境において、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。サーバ数は 4〜24、サーバあたりプロセス数 (以下、ローカルプロセス数) は 1〜8 で、各プロセス間の送受メッセージサイズは 1MB とした。

性能指標である 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、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

Vol.2009-ARC-186 No.4
Vol.2009-HPC-123 No.4
2009/11/30

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

`Msg`: 各プロセス対の送受メッセージサイズ（バイト）
`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-tl-all バンド幅の定義
Fig. 2 definition of all-to-all bandwidth

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ので、本稿では「All-to-all バンド幅 ≃ 実効ネットワークバンド幅」と考えることにする。

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 1600〜1700MB/s 程度である。一方、ローカルプロセス数 2 以上の場合は 1500〜1600MB/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 10) によれば、複数プロセスの方がネットワーク利用効率は高くなるとの結果が出ており、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 アルゴリズムの組み合わせが不適切である可能性を示唆している。

表 1 測定環境
Table 1 Testing environment

Servers	24 × 2way Nehalem server
CPU	2 × Intel Xeon X5570 (Nehalem 2.93GHz)
M/B	SuperMicro X8DTT
Mem	24GB (6 × 4GB DDR3-1333 DIMM)
NIC	Mellanox MHGH29-XTC (DDR-IB, PCIe2, ConnectX)
OS	RHEL5.4 (64bit)
MPI	OpenMPI 1.3.1
Compiler	gcc 4.1.2 (compile option: -O3)
Switch	Flextronics F-X430044 (DDR-IB, 24ports)

⋆1 特にサーバ数が少ないときは誤差が大きい

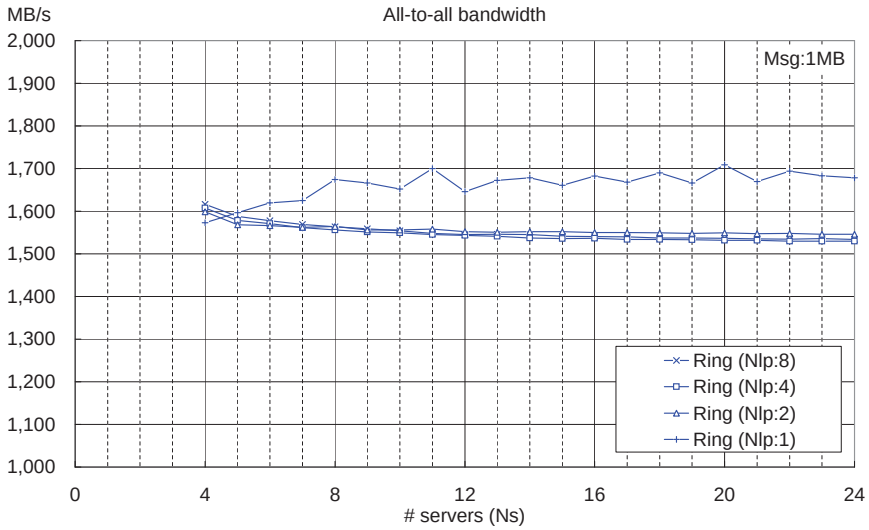

図 3 Ring アルゴリズムの All-to-all バンド幅測定結果
Fig. 3 All-to-all bandwidth of Ring algorithm

3.2 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結果を図 4 に示す。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実行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 1MB 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常に等しいので、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毎に実行時間が違うことが分かった。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が、これら通信ステップ

Vol.2009-ARC-186 No.4
Vol.2009-HPC-123 No.4
2009/11/30

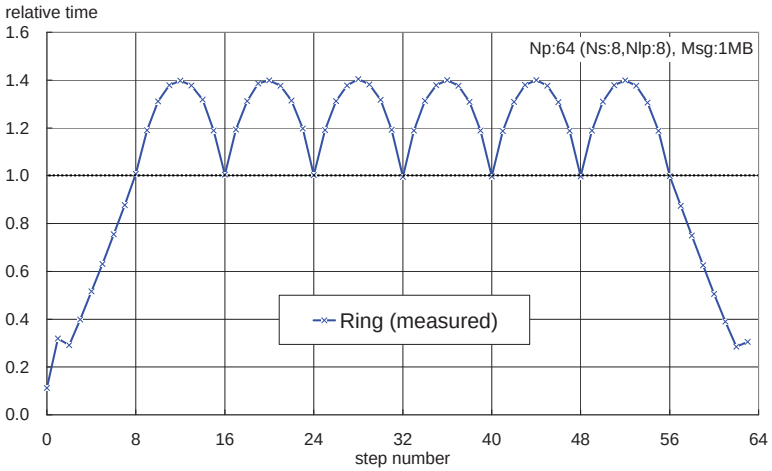

図 4 Ring アルゴリズムの各通信ステップの経過時間測定
Fig. 4 Measured elapsed time of each step in Ring algorithm

では一部もしくは全てのプロセスがサーバ内通信を行い、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されるメッセージ量が少ないことがその理由である。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。

3.3 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) する。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) する。従って、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[5] 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、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、更に、その後続パケット (サーバ C からサーバ D) もブロックされる様子を示している。

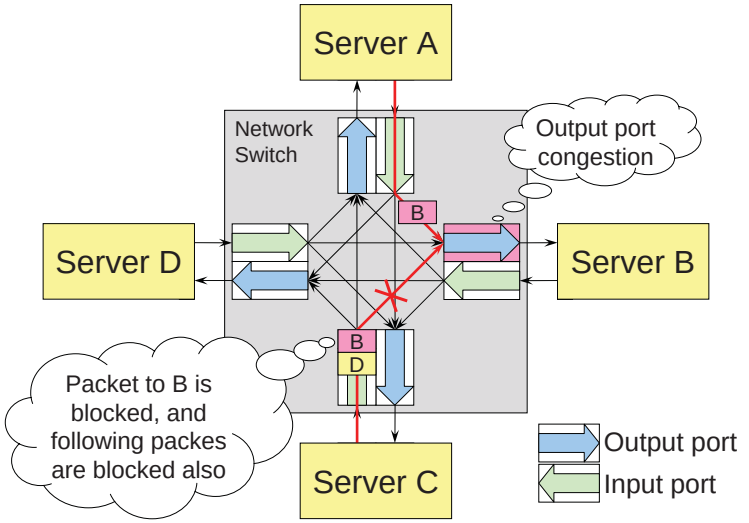

図 5 HoL ブロッキングのイメージ図
Fig. 5 image of HoL blocking

HoL ブロッキングが発生するのは、複数の入力ポートからある出力ポートへ同時にパケットを転送するときである。つまり、複数のサーバからあるサーバへ同時に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、複数のサーバからあるサーバへ同時にメッセージを送信することがあり、HoL ブロッキングの発生条件を満たしている。

3.4 HoL ブロッキングによる性能低下のモデル化と検証

次に、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に考える。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 の通りとする ($0 \leq \alpha \leq 1$)。

Vol.2009-ARC-186 No.4
Vol.2009-HPC-123 No.4
2009/11/30

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 example of message transfer 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

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。どのサーバも優先度に差がないとすると、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)。

更に、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性能低下を考慮した実行時間 (算出値) は、実測値と概ね一致していることが分かる。従って、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の HoL ブロッキング発生である可能性が非常に高いと言える。

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

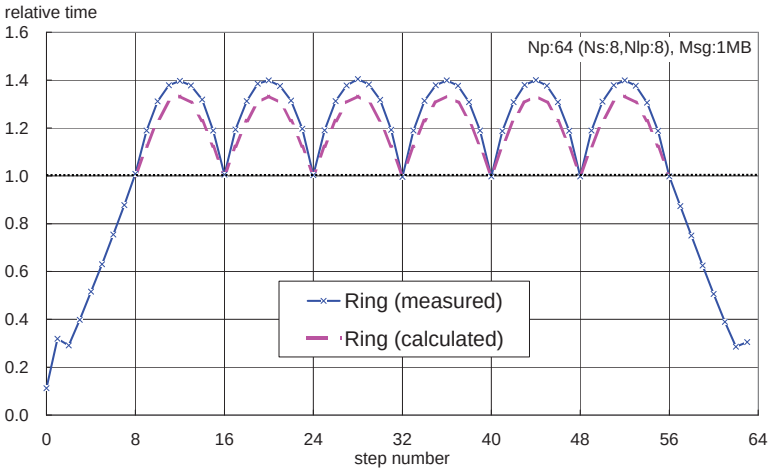

図 6 Ring アルゴリズムの各通信ステップの経過時間予測
Fig. 6 Calculated elapsed time of each step in Ring algorithm

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]
サーバ数を Ns とすると、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]
ローカルプロセス数を Nlp とすると、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]
通信ステップ毎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。

2-Level Ring アルゴリズムの特徴は、サーバ単位で通信相手を考えると、ある通信ステッ

Vol.2009-ARC-186 No.4
Vol.2009-HPC-123 No.4
2009/11/30

```
------------------------------------------------------------------------
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;
        Communication( "send msg to Ip_to and receive msg 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

プにおける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ことである。つまり、2-Level Ring アルゴリズムでは、あるサーバから同時に複数のサーバにメッセージを送信 (もしくは受信) 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。

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。

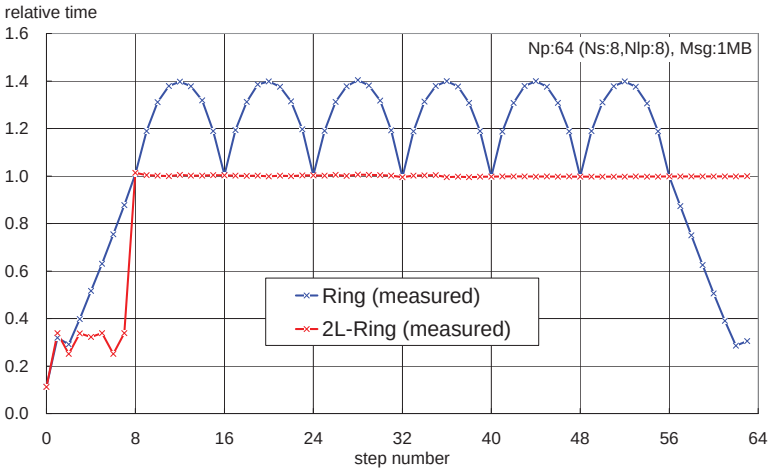

図 8 2-Level Ring アルゴリズムの各通信ステップの経過時間測定結果
Fig. 8 Measured elapsed time of each step in 2-Level Ring algorithm

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 8 に示す[⋆1]。図 8 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 1MB 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。期待通りの結果である。

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)· ローカルプロセス数 (1〜8) にて 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 9 に示す。図 9 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 1MB 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

Vol.2009-ARC-186 No.4
Vol.2009-HPC-123 No.4
2009/11/30

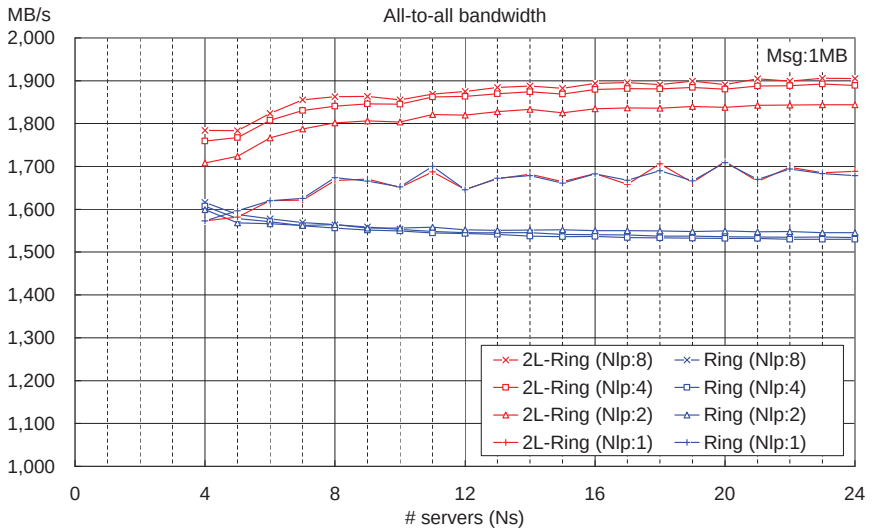

図 9 各種サーバ数での All-to-all バンド幅測定結果
Fig. 9 Measurement results of all-to-all bandwidth at various number of servers

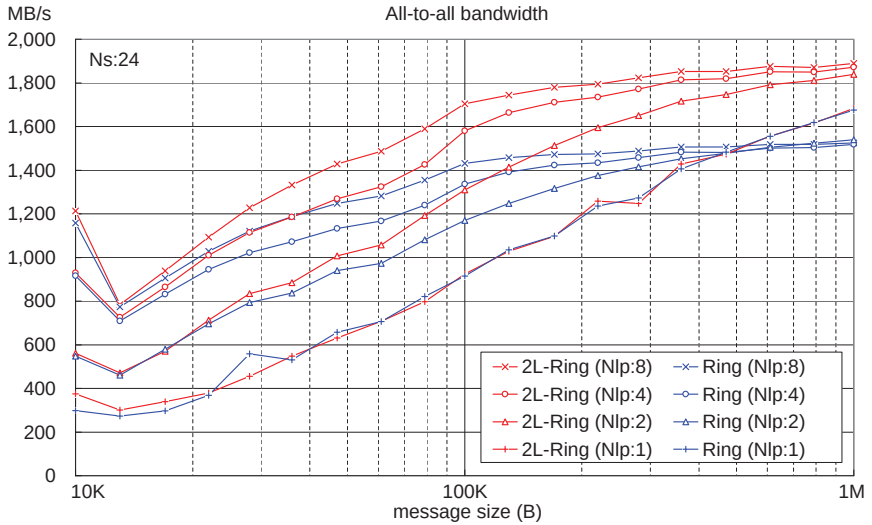

図 10 各種メッセージサイズでの All-to-all バンド幅測定結果
Fig. 10 Measurement result of all-to-all bandwidth at various message size

図 9 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 バンド幅が増加する結果となった。文献 10) の結果と同様の傾向を示しており、妥当な結果と言える。

最も性能向上率が高かったのは、総プロセス数:192(サーバ数:24 × ローカルプロセス数:8) の場合であり、Ring アルゴリズムでは 1,534MB/s だった All-to-all バンド幅が、2-Level Ring アルゴリズムでは 1,904MB/s に向上した。性能向上率は 24.1%である。ネットワークバンド幅の理論上限は 2,000MB/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 (10KB〜1MB) 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 より、メッセージサイズが数十 KB を越えると、ローカルプロセス数 2 以上の場合の All-to-all バンド幅が向上している。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場合に性能が向上しているのは妥当な結果と言える。

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。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はないので、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 アルゴリズムを使用することができる。

6. 関連研究

文献 8) では、All-to-all 通信時の HoL ブロッキングによる性能低下が言及されているが、多段ネットワークの下位 · 上位スイッチ間リンクでの競合を解消するルーティング方式が主

Vol.2009-ARC-186 No.4
Vol.2009-HPC-123 No.4
2009/11/30

テーマであり、ネットワークスイッチ内の競合に着目した本稿とはターゲットが異なる。

文献 9) では、マルチコア向け All-to-all 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 10) でも、マルチコア向け All-to-all アルゴリズムが提案されているが、サーバ間通信の前後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方法の一種であり、メッセージサイズが小さい場合には効果があるが、本稿がターゲットとしたメッセージサイズが大きい場合にはメモリコピー量が多くなり性能が低下する可能性が高い。実際、メッセージサイズが大きい場合の性能は評価されていない。

我々の知る限り、マルチコア PC クラスタシステムにて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、その発生頻度を削減してネットワーク利用効率を高める All-to-all 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 クラスタシステムにて従来アルゴリズム (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。

更に、HoL ブロッキングが発生しにくいマルチコア 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、24 台の 2way Nehalem サーバを DDR InfiniBand ネットワークで接続したマルチコア PC クラスタシステム上で評価した。評価の結果、Ring アルゴリズムと比較して最大で 24%の性能向上を確認、最高性能時の All-to-all バンド幅は 1,904 MB/s を記録、ネットワーク利用効率は 95%を超えており、2-Level Ring アルゴリズムの有効性を明らかにした。

2-Level Ring アルゴリズムはネットワーク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効果を期待できない。しかし、Ring アルゴリズムと比べて性能が低下することはなく、更に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、サーバ数は最大 24 台、ネットワークは InfiniBand の構成で 2-Level Ring アルゴリズムの評価を行ったが、今後はより大規模構成での評価、InfiniBand 以外のネットワークでの評価を行うとともに、All-to-all 通信を行うアプリケーションレベルの性能評価も行う予定である。

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) M. Karol, M. Hluchyj, S. Morgan: “Input Versus Output Queueing on a Space-Division Packet Switch”, IEEE Transactions on Communications, Volume 35, Issue 12, pp. 1347-1356 (1987)
6) A. Faraj, X. Yuan: “Automatic generation and tuning of MPI collective communication routines”, The 19th Annual international Conference on Supercomputing (2005).
7) 南里 豪志, Hyacinthe Nzigou Mamadou, Feng Long Gu, 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, 2008-HPC-118, pp. 73-78 (2008).
8) P. Geoffray, T. Hoefler: “Adaptive Routing Strategies for Modern High Performance Networks”, 16th IEEE Symposium on High Performance Interconnects (2008).
9) A.R. Mamidala, R. Kumar, D. De, D.K. Panda: “MPI Collectives on Modern Multicore Clusters: Performance Optimizations and Communication Characteristics”, The 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (2008).
10) R. Kumar, A. Mamidala and D.K. Panda: “Scaling alltoall collective on multi-core systems”, IEEE International Symposium on Parallel and Distributed Processing (2008).