社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－HPC－107 (44)
2006／8／2

グリッド上のコレクティブ通信アルゴリズム

松田元彦† 石川 裕†,†† 工藤知宏†
児玉祐悦† 高野了成†

グリッド環境でのコレクティブ通信アルゴリズムが提案されているが、これまでは広域ネットワークはバンド幅が小さいという仮定が置かれていた。しかし近年、バンド幅は急速に向上し従来の仮定が現状にマッチしなくなっている。そこで、高遅延かつ高バンド幅のネットワーク環境に対応したコレクティブ通信アルゴリズムを提案する。高バイセクション・バンド幅を持つネットワークで効率的な van de Geijn と Rabenseifner の各アルゴリズムを拡張し、クラスタ間ネットワークのバンド幅を有効利用できるようにした。加えて、ネットワークの同時利用ノード数を制限することで広域ネットワークでの輻輳の抑制を行った。アルゴリズムの中から Bcast、Allreduce、Gather、Alltoall について、ネットワーク・エミュレーション環境を用いた評価を行い性能を確認した。

Efficient Collective Algorithms for Grid Environment

MOTOHIKO MATSUDA ,† YUTAKA ISHIKAWA ,†,†† TOMOHIRO KUDOH ,†
YUETSU KODAMA † and RYOUSEI TAKANO†

Several MPI systems have been proposed for Grid environment, in which collective algorithms for wide-area networks are provided, but they are optimized under the assumption of low bandwidth in wide-area networks. However, recently, the bandwidth of wide-area networks has become much wider, and the assumption is now obsolete. Thus, we designed new collective algorithms by modifying ones for clusters proposed by van de Geijn and by Rabenseifner, to effectively utilize the available bandwidth of fast wide-area networks. In addition, the algorithms incorporate a mechanism to restrict the number of nodes to simultaneously communicate over wide-area networks, to reduce network congestion. We confirmed the effectiveness of the algorithms for Bcast, Allreduce, Gather, and Alltoall by experiments using an emulated network environment.

1. はじめに

グリッド環境で複数のクラスタを接続して利用するための MPI システムとして MPICH-G2、MagPIe、PACX-MPI などが知られている[1)~3)]。これらグリッド用 MPI システムでは、高遅延ネットワークに最適化されたコレクティブ通信アルゴリズムが提案されている。ただしその暗黙の仮定として、遅延が大きいことと同時にバンド幅が小さいことが想定されている。

しかし近年、広域ネットワークのバンド幅は急速に向上し、10Gbps から 40Gbps といったバンド幅が提供されるようになってきた。このバンド幅は典型的なクラスタ・ノードの NIC の持つ 1Gbps というバンド幅を越えている。この傾向は、クラスタ・ノードの NIC にかけられるコストを勘案すると、しばらくは続くと考えられる。このようなネットワーク環境は、これまでのグリッド用 MPI における仮定とマッチしていない。

一方、クラスタ内の全ノードが一斉に通信を行うと、ネットワークが輻輳する。一般に、広域ネットワークでは通信に TCP/IP プロトコルが用いられる。TCP では輻輳により致命的に性能が低下することが知られている[4)]。バンド幅を有効利用するとともに、輻輳を避ける必要がある。

そこで我々が開発している GridMPI[5),6)] では、コレクティブ通信のアルゴリズムを高遅延かつ高バンド幅なネットワークに最適化しようとしている。論文 7) では、Bcast と Allreduce の 2 つの通信オペレーションについて高バンド幅ネットワークに最適化したアルゴリズムを提案、評価を行った。

Bcast と Allreduce は MPI のコレクティブ通信で重要なものであり、利用が多いことが知られている[8)]。Bcast と Allreduce の 2 つのオペレーションについては、巧みで高性能なアルゴリズムが提案されている。Bcast のアルゴリズムは van de Geijn らによるものである[9)]。Allreduce のアルゴリズムは Rabenseifner によ

† 産業技術総合研究所グリッド研究センター
National Institute of Advanced Industrial Science and Technology
†† 東京大学 大学院情報理工学系研究科
University of Tokyo

表 1 MPI-1.2 のコレクティブ通信

MPI_Barrier	MPI_Bcast	MPI_Gather	MPI_Gatherv	MPI_Scatter	MPI_Scatterv	MPI_Allgather
MPI_Allgatherv	MPI_Alltoall	MPI_Alltoallv	MPI_Reduce	MPI_Allreduce	MPI_Reduce_scatter	MPI_Scan

るものである[10]。これらはバイセクション・バンド幅の大きなネットワークで効率が良い。

本稿では論文7)で述べた Bcast、Allreduce オペレーションのアルゴリズムの説明に続いて、それ以外のコレクティブ通信について高遅延かつ高バンド幅なネットワークに対応するアルゴリズムを示す。アルゴリズムはメッセージサイズが大きい場合、かつ、ノード数が 2 巾の場合のみを扱うものとする。ノードが複数のネットワーク接続を持つ場合は考慮しない。

我々は、MPI アプリケーションをグリッド環境に移行するのに、遅延時間 10 ミリ秒というのが一つの目安になると考えている。遅延時間と性能の関係は、当然、アプリケーションの性質によって異なる。しかし、ベンチマーク評価から遅延時間 10 ミリ秒までは特に遅延に対処したコーディングを行なわない場合でも、性能劣化がそれ程大きくないという結果が得られている[11]。このため、評価実験は 2 つのクラスタを 10 ミリ秒の遅延で接続した環境で行った。

本稿では以降、2 章で設計方針を述べる。3 章で各アルゴリズムの設計を述べる。4 章で評価実験の結果を示す。5 章でまとめる。

2. 設計指針

既存のコレクティブ通信アルゴリズムを高バンド幅ネットワークに対応させて効率化するポイントは、次の 2 点である。

- 複数のノードがクラスタ間通信に関わる。
- 一斉に通信を行うノード数を制限できる。

Bcast アルゴリズムを例にとると、従来のアルゴリズムではクラスタ間通信をまず行いその後にクラスタ内の Bcast を行う。この場合、送信データを持つルートノードのみがクラスタ間通信を行なうので、バンド幅が有効に利用されない。

Bcast を行う van de Geijn アルゴリズムでは、Scatter に続いて Allgather を行う。Allreduce を行う Rabenseifner アルゴリズムは、Reduce-scatter に続いて Allgather を行う。クラスタ間通信を行うことを考えてこれらのアルゴリズムの動作を見直してみる。

van de Geijn アルゴリズムでは、前半の Scatter の時点でデータの分散が行われている。Scatter までをクラスタ内で行いそこでクラスタ間通信を行うと、複数ノードが通信すべきデータを持っている。これはクラスタ間通信に好都合である。

同様に、Rabenseifner アルゴリズムでは、前半の Reduce-Scatter の時点でデータの分散が行われている。この場合、Reduce-Scatter までをクラスタ内で行い、次いでクラスタ間通信を行う。

クラスタの全ノードがクラスタ間通信を行うデータを持っているので、輻輳を避ける必要がある。そのためには、同時に通信するノード数を制限するのが簡単かつ効果的である。そこで、クラスタ間のネットワーク・バンド幅に合わせて一定数のノードのみが通信を行うように制限を加える。

コレクティブ通信はプリミティブなオペレーションの組み合わせとして実現できる。例えば、van de Geijn アルゴリズムでは、Bcast を Scatter と Allgather の組み合わせとして、Rabenseifner アルゴリズムでは、Allreduce を Reduce-scatter と Allgather の組み合わせとして構成している。さらに、Scatter、Reduce-scatter、Allgather もよりプリミティブなオペレーションの組合せで実現される。この組み合わせを積極的に行い、実験的に最適な組合わせを探す研究もある[12]。本稿でも、プリミティブの組合わせを積極的に利用している。

3. 実　　装

3.1 コレクティブ通信

表 1 に MPI-1.2 で定義される全コレクティブ通信オペレーションを示す。本稿では、Bcast、Allreduce、Gather、Alltoall についてアルゴリズムの提案を行う。この他、Scatter、Allgather、Reduce-scatter、Reduce については Bcast と Allreduce アルゴリズムの部分アルゴリズムが使用できる。

これら以外のオペレーションは以下の理由から対象から外した。Barrier は通信データサイズが 0 であるので本稿では扱わない。ポストフィックス v が付くオペレーション (**MPI_XXXv**) はノード毎に異なるサイズのデータを通信を行う。これらは v の付かないオペレーションと同様である。

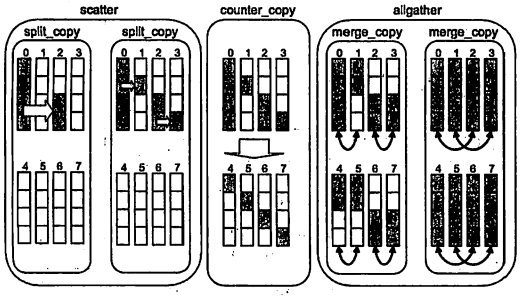
Scan を Rabenseifner アルゴリズムのように処理する場合、Scan に適したノードに結果が集らない。うまくアルゴリズムの変形として処理できないので、Scan は今回の対象から外した。

3.2 簡単な通信コスト・モデル

以下のセクションでは、アルゴリズムの説明にその通信コストを付記する。そのための非常に単純化した通信コスト・モデルを説明する。

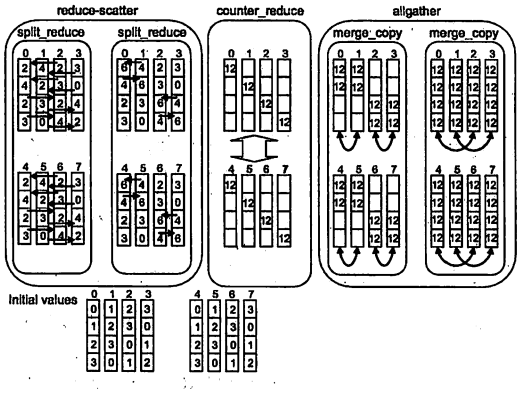
P はノード総数、M はメッセージサイズ、B はノードの持つ NIC のバンド幅、L はクラスタ間の遅延、n はクラスタ間で同時に通信するリンク数である。

クラスタ間の遅延時間は、メッセージ・サイズやクラスタ内の遅延に対して十分大きいと仮定している。特に、クラスタ内の遅延や通信の起動オーバーヘッド




```
void
vandegeijn(void *buf, int siz)
{
  if (rank < (nprocs/2)) {
    for (i = 0; i < (log2(nprocs)-1); i++) {
      split_copy(buf, siz, i, hemisphere);
    }
  }
  counter_copy(buf, siz);
  for (i = 0; i < (log2(nprocs)-1); i++) {
    merge_copy(buf, siz, i, hemisphere);
  }
}
```

図 1 拡張 van de Geijn アルゴリズム (Bcast) とそのデータ移動




```
void
rabenseifner(void *buf, void *tmp, int siz)
{
  for (i = 0; i < (log2(nprocs)-1); i++) {
    split_reduce(buf, tmp, siz, i, hemisphere);
  }
  counter_reduce(buf, tmp, siz);
  for (i = 0; i < (log2(nprocs)-1); i++) {
    merge_copy(buf, siz, i, hemisphere);
  }
}
```

図 2 拡張 Rabenseifner アルゴリズム (Allreduce) とそのデータ移動

等は無視する。

3.3 Bcast(拡張 van de Geijn)

図 1 に拡張 van de Geijn アルゴリズムを示す。これは van de Geijn をベースにクラスタ間の通信ステップを付加したアルゴリズムである。各ステップの処理は以下のようになる。

(1) split_copy のループは Scatter に相当する。i 番目の split_copy のステップで、各ノードはメッセージを半分に分け、その半分を $2^{(nprocs/2-i)}$ 離れたノードへ転送する。

(2) counter_copy は (1) のステップで Scatter されたデータをクラスタ間でコピーする。各ノードはそれぞれクラスタ間で 1 対 1 で対向した転送を行う。

(3) merge_copy のループは Allgather に相当する。i 番目の merge_copy のステップで、各ノードはメッセージを 2^i 離れたノードへ転送し自分の持つメッセージと転送されたメッセージをマージする。

counter_copy がクラスタ間通信のために追加されている。これを取り除くと van de Geijn アルゴリズムそのものである。

counter_copy で全てのノードが一斉にクラスタ間通信を行うと、ボトルネックになっているリンクが輻輳する。輻輳を避けるため、counter_copy では同時に通信するノード数を制限している。指定されたノードのみがクラスタ間通信を行い、それ以外のノードは指定されたノードへメッセージをフォワードする。同時に通信するノード数は、クラスタ間ネットワークに合わせて 1 から $nprocs/2$ の間で自由に選ぶことができる。

簡単化した通信コストを表 2 に示す。コストは通信時間で表す。$L + M/nB$ の項はクラスタ間の通信に相当する。n ノードが同時に通信するのでバンド幅を n 倍している。次に、1 つ目の M/B の項はステップ (1) の Scatter、もう 1 つはステップ (3) の Allgather に相当する。Scatter と Allgather はともにメッセージサイズを順次 1/2 にしながら log(P) ステップ繰り返すので、通信時間の合計は漸近的に M/B となる。詳しく説明しないが他のアルゴリズムも同様である。

3.4 既存の Bcast (farfirst)

他の MPI で提案されているアルゴリズムは、まず遅延時間の大きいクラスタ間通信を行うというものである。本稿では、このアルゴリズムを farfirst アルゴリズムと呼ぶ。

簡単化した通信コストを表 2 に示す。クラスタ内の Bcast には van de Geijn アルゴリズムを使用している。拡張 van de Geijn と farfirst を比較すると、複数リンクを用いることでクラスタ間の通信時間が減少していることが分る。

3.5 Allreduce(拡張 Rabenseifner)

図 2 に拡張 Rabenseifner アルゴリズムを示す。これは Rabenseifner をベースにクラスタ間通信を付加した

表 2 アルゴリズムの通信コスト

拡張 van de Geijn:	$(L + M/nB + M/B + M/B)$
farfirst:	$(L + M/B + (M/B + M/B))$
拡張 Rabenseifner:	$(L + M/nB + M/B + M/B)$
twotier:	$(L + M/B + 2 \cdot (M/B + M/B))$
Scatter:	$(L + M/B)$
Allgather:	$(L + M/nB + M/B)$
Reduce-scatter:	$(L + M/nB + M/B)$
Reduce:	$(L + M/nB + M/B + M/B)$
twotier (Reduce):	$(L + M/B + M/B + M/B)$
Gather:	$(L + M/B)$
Alltoall:	$(L + M/B)$

アルゴリズムである。各ステップの処理は以下のようになる。

(1) split_reduce のループは Reduce-scatter に相当する。i 番目の split_reduce のステップで、各ノードはメッセージを半分に分け、その半分を $2^{(nprocs/2-i)}$ 離れたノードへ転送する。そこでリダクションを行う。

(2) counter_reduce は (1) のステップで Reduce-scatter されたデータをクラスタ間で交換しリダクションを行う。Bcast の counter_copy と同様であるが、双方向である点とリダクションを行う点が異なる。

(3) merge_copy のループは Allgather に相当する。これは Bcast の後半と同一である。

counter_reduce がクラスタ間通信のために追加されている。これを取り除くと Rabenseifner アルゴリズムそのものである。counter_reduce においても counter_copy 同様にクラスタ間の同時通信ノード数の制限を行っている。

3.6 既存の Allreduce (twotier)

他の MPI で提案されているアルゴリズムは、まずクラスタ内でリダクションを行い、結果をクラスタ間で通信するというものである。本稿では、このアルゴリズムを twotier アルゴリズムと呼ぶ。twotier アルゴリズムでは、最初にクラスタ内で Reduce、次にクラスタ間通信、その後クラスタ内で Bcast を行うことになる。

簡単化した通信コストを表 2 に示す。拡張 Rabenseifner では、クラスタ間の通信時間が減る以上に通信時間が小さくなることが分る。

3.7 Scatter

Scatter は、1 つのルートノード上のデータを分散させる通信である。ルートノードの通信能力から、単純にルートノードから各ノードに順次 1 対 1 通信を行う通信が最適である。つまり、複数リンクを使う通信にメリットはない。

3.8 Allgather/Reduce-scatter

Allgather は van de Geijn アルゴリズムの後半に現れる処理である。まずクラスタ間で通信し、続いてクラスタ内で Allgather を行う。

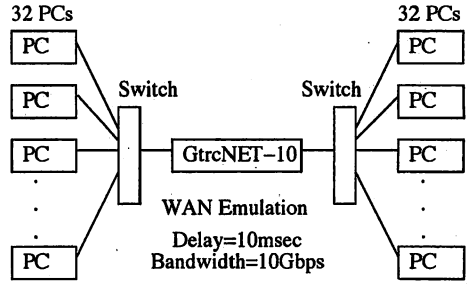


図 3 Experimental Setting

表 3 PC Cluster Specifications

Node PC	
CPU	Opteron (2.0 GHz)
Memory	6GB DDR333
NIC	Broadcom BCM5704 (on-board)
OS	SuSE Enterprise Server 9 (Linux-2.6.5)
Switch	
Huawei-3Com Quidway S5648 + optional 10 Gbps port	

Reduce-scatter は Rabenseifner アルゴリズムの前半に現れる処理である。クラスタ内で Reduce-scatter を行い、続いてクラスタ間で通信する。

3.9 Reduce

Reduce は Reduce-scatter+Gather(クラスタ内) に分解できる。クラスタ内の Gather は Allgather と同じ M/B 時間かかる。つまり、Allreduce と同様である。twotier アルゴリズムでは、Bcast を行わないので $(M/B + M/B)$ の項を省略する。

3.10 Gather/Alltoall

Gather は分散したデータを 1 つのルートノードに集約する通信である。Gather も Scatter と同様、ルートノードの通信能力から、各ノードに対して順次 1 対 1 通信を行う通信が最適である。

ただし、通信がルートノードに集中する通信であるので、輻輳が起る可能性がある。そこで、クラスタ間の通信に counter_copy を使用する。また、クラスタ内でも輻輳は起るため、クラスタ内の通信にはランデブープロトコルを使用する。ただし、クラスタ間とクラスタ内の通信はフェーズ分けされていないため輻輳は完全には抑制されていない。Gather の場合の通信コストは、通信データ総量を M としている。

Alltoall は直接 1 対 1 で通信するのが効率的と考えられる。ただし、クラスタ間通信での輻輳を回避するため、通信するノード数の制限を行っている。ただし、クラスタ間とクラスタ内の通信はフェーズ分けされていないためクラスタ内でもクラスタ間でも輻輳が起こり得るアルゴリズムになっている。

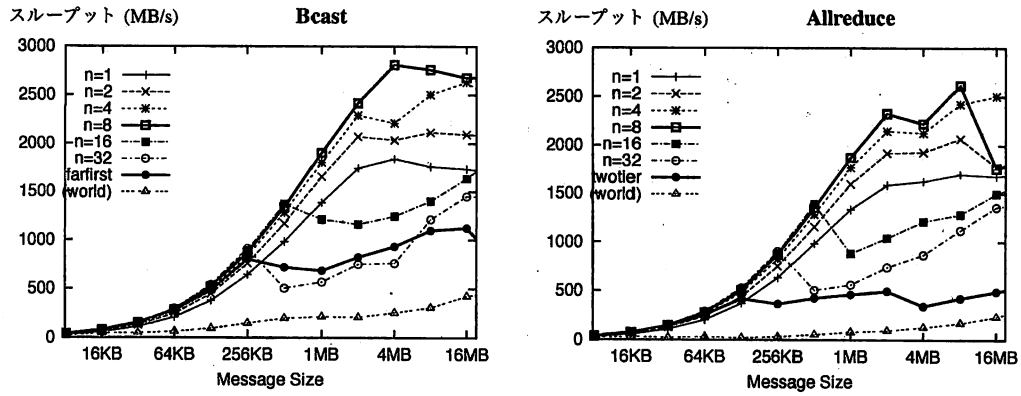


図 4 Bcast と Allreduce のスループット (遅延 10msec)

4. 評　　価

4.1 実験環境

図 3、表 3 に実験環境を示す。32+32 台の PC からなる 2 つのクラスタを遅延を挿入して接続している。PC からは、1Gbps で 10Gbps のアップリンクを持つスイッチに接続している。クラスタ間はバンド幅 10Gbps、遅延 10 ミリ秒で接続している。クラスタ間の遅延の挿入にはネットワークエミュレータ GtrcNET-10[13] を使用した。

以下では、Bcast、Allreduce、Gather、Alltoall について実験結果を示す。Bcast、Allreduce については、広域ネットワークに対応した MPI で提案されているアルゴリズムとの比較を示す。Gather と Alltoall については、比較すべきアルゴリズムの提案がないので、遅延のないクラスタ環境で用いられるアルゴリズムとの比較を示す。

4.2 Bcast

図 4(左) に Bcast の性能を示す。Y 軸の値は、通信する総転送データ量を通信時間で割った値 ($M * nprocs/T$) を表示している。図中「n=1〜32」は提案するアルゴリズム (拡張 van de Geijn) の結果である。図中「farfirst」はクラスタ間で、遅延のある通信をまず行うアルゴリズムの場合である。図中「(world)」は、参考として遅延を考慮せず van de Geijn アルゴリズムを全プロセスで行った場合である。

提案アルゴリズムが他の MPI で提案されている farfirst を上回っている。また、クラスタ間の同時通信数が 8 で最大になり、それ以上になると性能が低下することが分かる。

4.3 Allreduce

図 4(右) に Allreduce の性能を示す。Y 軸の値は、全対全で通信した場合の総転送データ量を通信時間で割った値 ($M * nprocs^2/T$) を表示している。アルゴリズムの実際の通信量と異なるため、Y 軸の数値自体は目安である。図中「n=1〜32」は提案するアルゴリズム (拡張 Rabenseifner) の結果である。図中「twotier」はクラスタ内でリダクションした値をクラスタ間で交換するアルゴリズムの場合である。図中「(world)」は、参考として遅延を考慮せず Rabenseifner アルゴリズムを全プロセスで行った場合である。

提案アルゴリズムが他の MPI で提案されている twotier を上回っている。提案アルゴリズムでは、コストモデルからも分るように、twotier アルゴリズムに比べて大きく性能が改善されている。また、クラスタ間の同時通信数が 8 で最大になり、それ以上になると性能が低下することが分かる。

4.4 Gather

図 5(左) に Gather の性能を示す。Y 軸は各プロセスの転送データ量を通信時間で割った値である。図中「n=1〜32」は提案するアルゴリズムの結果である。図中「direct」はプロセスの全ペアについて直接 Isend と Irecv を使って一斉に通信を起動した場合である。

直接通信を行う場合、小さいメッセージサイズでの性能は良いが、メッセージサイズが大きくなるとすぐに性能が低下する。クラスタ間の同時通信数が 8 で最大になり、それ以上になると性能が低下することが分かる。

メッセージサイズが 2MB の時点で一度性能が低下しているが、原因は解析できていない。考えられる原因として、クラスタ内の転送とクラスタ間の転送が同時に進行しているので、メッセージが衝突する可能性がある。これについては、データを受信するルートノードをクラスタ間通信から外すなど改善の余地がある。

4.5 Alltoall

図 5(右) に Alltoall の性能を示す。Alltoall で各プロセスはデータサイズと全プロセス数の積にあたるデータ量を転送する。データ量が大きいため、他のオペレーションよりデータサイズの範囲を小さくしてい

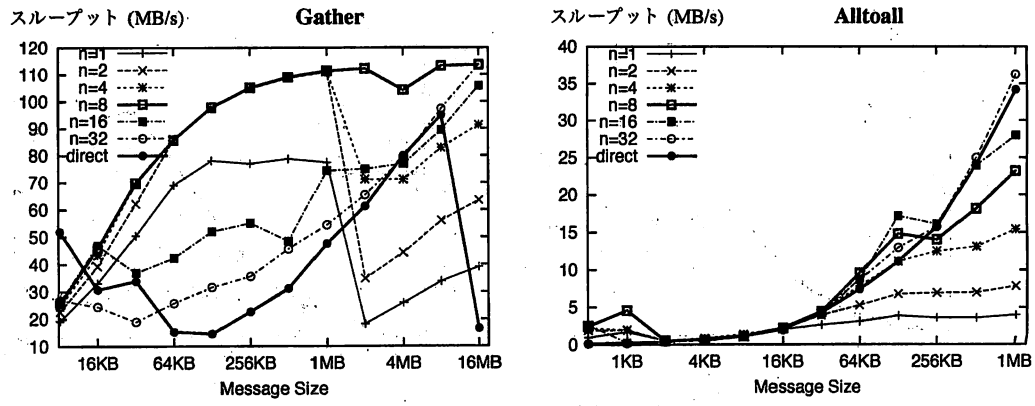


図 5 Gather と Alltoall のスループット (遅延 10msec)

る。Y 軸は各プロセスの転送データ量を時間で割った値である。自分への転送もデータ量に含めている。図中「n=1〜32」は提案するアルゴリズムの結果である。図中「direct」はプロセスの全ペアについて直接 Isend と Irecv を使って一斉に通信を起動した場合である。

クラスタ間通信の制約を行った場合も、同時 8 本で行うより同時 32 本で通信する場合の方が良い結果が得られた。Alltoall ではクラスタ間の転送とクラスタ内の転送が同時に進行している。そのため、クラスタ間転送でも通信の衝突が発生する。逆に、一斉に通信を起動した場合、多数の通信ストリームが同時に動作しているので性能低下がそれほど大きくなかったと想像される。Alltoall の性能改善は今後の課題として残る結果となった。

5. ま と め

高遅延かつ高バンド幅のネットワークに適応したコレクティブ通信アルゴリズムを提案した。ポイントは、クラスタ間で複数コネクションを同時に使用すると同時に、使用コネクション数を制限できる点である。特に、Bcast と Allreduce についてはクラスタ環境で効率の良い van de Geijn、Rabenseifner アルゴリズムをベースにクラスタ間で複数コネクションを同時に使用する拡張を行った。また、Gather と Alltoall についても、クラスタ間で同時通信数を制限するアルゴリズムを提示した。

10Gbps の NIC が安価に利用できる頃には、広域ネットワークも 40Gbps やそれ以上になっていると予想される。ここで提案するような複数コネクションを利用するコレクティブ通信は、今後も必要であると考えている。

謝　辞

本研究の一部は文部科学省「経済活性化のための重点技術開発プロジェクト」の一環として実施している超高速コンピュータ網形成プロジェクト (NAREGI: National Research Grid Initiative) による。

参 考 文 献

1) N.T.Karonis, B.R.deSupinski, I.T.Foster, W.Gropp, E.L.Lusk, and S.Lacour. A Multilevel Approach to Topology-Aware Collective Operations in Computational Grids. Tech.Rep. ANL/MCS-P948-0402, 2002.
2) T.Kielmann, H.E.Bal, and S.Gorlatch. Bandwidth-efficient Collective Communication for Clustered Wide Area Systems. Proc.of the 14th Intl.Symp.on Parallel and Distributed Processing, 1999.
3) E.Gabriel, M.Resch, and R.Rühle. Implementing MPI with Optimized Algorithms for Metacomputing. Proc.of the Third MPI Developer's and User's Conference, 1999.
4) 高野, 工藤, 児玉, 松田, 手塚, 石川. GridMPI のための TCP/IP 輻輳制御実装方式の検討. 情報処理学会 2004-OS-097, SWoPP'04, 2004.
5) 松田, 石川, 鐘尾, 枝元, 岡崎, 鯉江, 高野, 工藤, 児玉. GridMPI Version 1.0 の概要. 情報処理学会 2005-HPC-103, SWoPP'05, 2005.
6) GridMPI Project. http://www.gridmpi.org
7) M.Matsuda, Y.Ishikawa, T.Kudoh, Y.Kodama, and R.Takano. Efficient MPI Collective Operations for Clusters in Long-and-Fast Networks. Proc.of IEEE Cluster2006, 2006 (to appear).
8) R.Rabenseifner. Automatic MPI Counter Profiling of All Users: First Results on a CRAY T3E 900-512. Proc.of the Message Passing Interface Developers and Users Conference 1999 (MPIDC99), 1999.
9) M.Barnett, S.Gupta, D.G.Payne, L.Shuler, R.van de Geijn, and J.Watts. Building a High-Performance Collective Communication Library. Supercomputing94, 1994.
10) R.Rabenseifner. Optimization of Collective Reduction Operations. Intl.Conf.on Computational Science, LNCS, 2004.
11) M.Matsuda, Y.Ishikawa, and T.Kudoh. Evaluation of MPI Implementations on Grid-connected Clusters using an Emulated WAN Environment. CCGrid2003, 2003.
12) 今村, 山田, 町田. 大規模 SMP クラスタにおける固有値ライブラリの通信最適化について. 情報処理学会 2006-HPC-105 HOKKE'06, 2006.
13) GtrcNET-10. http://www.gtrc.aist.go.jp/gnet