RC-009

実行時の動的かつ頻繁なグループ変更に対応可能なMPI環境下でのマルチキャストの実装

An Implementation of the MPI Multicast Function Capable of Coping with Run-time Dynamic and Frequent Update of Multicast Group

長嶺 祐輔 * , 福間 慎治 *, 森 眞一郎 *

Nagamine Yusuke, Shinji Fukuma, Shin-ichiro Mori

1 はじめに

我々は次世代のシミュレーション技術として, 大規模並列環境下での実時間シミュレーション環境を提供する MPI ベース [1] のシミュレーションフレームワークの開発 [2] を行っている. このフレームワークではマスター・ワーカー型の並列処理機構を採用しており, シミュレーションの時間進行と共に, マスターノードが特定のワーカノード群に対して順次シミュレーションタスクを割り当てることでシミュレーションを進めていく. ワーカノード群の初期化はタスク割り当て時に毎回実施するのではなく, シミュレーション開始時に一度だけ実行環境の初期化を行ない, それ以降のタスク割り当て時には過去の実行結果（履歴）からの差分情報のみで計算に必要な初期化処理を行なう.

その実装においては, 図 1 に示すように, マスターノードからある条件が成立するワーカーノード群に対してマルチキャストが必要となる. また, ワーカーノード群を特定する条件は時系列シミュレーションの実行結果や実行時のユーザ介入に応じて実行中に数十から数百ミリ秒の間隔で動的に再定義されるためプログラムの実行開始前にマルチキャストの配信構造を静的に求めておくことは不可能である.

MPI では集団通信手法としてブロードキャスト (一対全通信) のメカニズムが提供されており, あらかじめ指定したグループに属す全てのノードにブロードキャストする機能を利用してマルチキャストを実装することが可能である. しかしながら, 各ワーカーノードは自分がどのグループに所属するかの情報が実行前に既知である必要があり, 実行時にグループ構成が確定しかつ頻繁にその構成が変動する環境下でのマルチキャストへの応用ができない.

本論文では, このような MPI を用いたマスター・ワーカー型の大規模並列処理環境下で, 実行時の動的かつ頻繁なグループ変更に対応可能な動的マルチキャスト機構の実装について報告を行う.

*福井大学大学院工学研究科, University of Fukui

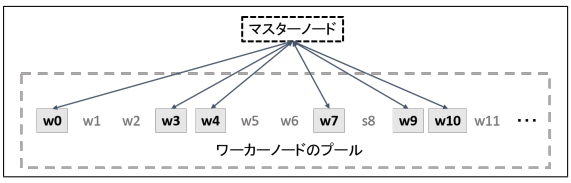

図 1: マスター・ワーカー型シミュレーション概念図

2 研究関連

2.1 MPI_Bcast を用いたマルチキャスト

本研究で使用するメッセージ通信ライブラリ (MPI) では, 原則として通信を行うノード同士が通信を行う前に相互に既知でなければならない.

2.2 MPI_Spawn による動的グループ構築

MPI の動的プロセス生成機能である MPI_Spawn を用いると, 計算に必要な数のプロセス群を動的に生成し, 生成されたプロセス群内での集団通信を可能にできる. しかしながら, 本研究で想定している数十ミリ秒単位で動的なグループ生成が必要な環境では, シミュレーションの実行に必要な初期化処理が必要となるプロセスの動的生成は非効率であり, 実時間性の保証が求められる本研究への適用は不適切である.

2.3 配信アルゴリズム

ここでは集団通信の配信アルゴリズムの内, 本提案手法に関連する木構造ベースの配信アルゴリズムと, その通信コストについて説明する. 相互通信する上で 1 byte のデータ送信に要する時間を M [sec], ノード間の遅延を S [sec] とし, 集団通信に参加するノード数を p, 送信するメッセージ長を n [byte] とする. 木の高さや通信ステップの回数を $stage$ と表現する. また, いかなるノード間でも通信遅延は同じであると仮定する.

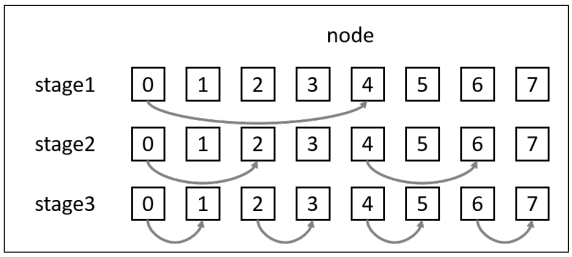

図 2: binomial-tree 構造

2.3.1 逐次的配信

一例として root がすべてのメンバーに逐次配信する場合のコストを表す．1 回の通信コストは $nM+S$ である．これを root は $p-1$ 回逐次的に配信していく．よって通信コストは $(nM+S)(p-1)$ と表される．一方で p 台のノード間で i 番目のノードが $(i+1)$ 番目のノードへ順番に配信する ring 形式の分散配信の場合も同じ通信コストになる．

2.3.2 binomial-tree 配信

binomial-tree アルゴリズム [3, 9] は MPICH のメッセージ長や配信ノード数が小さい Bcast に使用される．また他の条件下での Bcast を行うための基盤にもなっている．図 2 はノード数 8 の場合の binomial-tree 分配構造である．root となるノード 0 を最上階として，下の階層へ 2 分木的に次々に分配されてくアルゴリズムであり，高さは $\lceil \log_2 p \rceil$，1 stage あたりの通信コストは $nM+S$ である．よって集団通信コストは

$$(nM+S)\lceil \log_2 p \rceil \tag{1}$$

となる．

2.3.3 binomial-tree_scatter_ring_allgather 配信

MPICH の Bcast には scatter_allgather によって構築されたアルゴリズムが存在する． scatter 部と allgather 部の 2 部で構成され，このとき allgather を ring アルゴリズムによって行うものが binomial-tree_scatter_ring_allgather である． MPICH の Bcast ではメッセージ長が 512 [Kbyte] 以上，もしくはノード数が 2 のべき乗でないときに利用される．図 3 のように，始めに scatter 部では binomial-tree によって集団通信に参加するノードにメッセージを分配する．このときメッセージの全長を送信するのではなく，$1/p$ に分割したメッセージをそれぞれノードに振り分けて送信する．この分散したメッセージを ring アルゴリズムによる allgather で収集することによってマルチキャストが完了する．図 4 はメッセージの配信の様相を rank 数 8 の場合で表したものである．

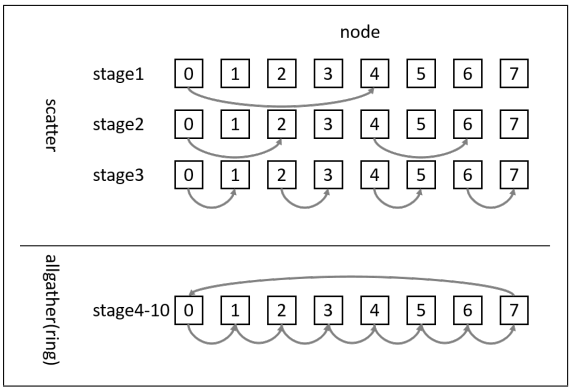

図 3: ring アルゴリズムによる scatter_allgather

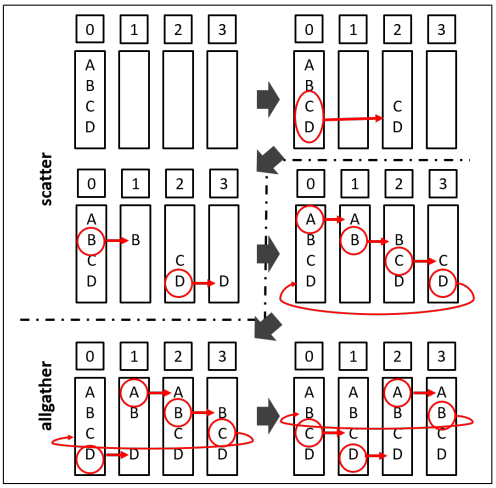

図 4: binomial-tree_scatter_ring_allgather によるメッセージ配信の様子

scatter 部 scatter 部の通信コストについて説明する．binomial-tree のアルゴリズムによる木構造に従って，1 stage では最上階となる root からサイズが $n/2$ のメッセージがひとつ下の階層へと送信される．例として，図 4 では rank4 に 2 分割したメッセージを送信している．i stage ではサイズ $n/2^i$ に分割されたメッセージがひとつ下の階層へとそれぞれ分配されていく．メッセージを受信した中継ノードは受信後は送信ノードとなり，同様に自身の配下となるノードに分配する．この手順をすべてのノードが行うことでサイズ n/p のメッセージが各ノードに分配されたことになる．このアルゴリズムにより，通信コストが最大となるのは root である．root の通信コストは

$$\sum_{i=1}^{\log_2 p} (nM/2^i + S) \tag{2}$$

よって scatter 部の通信コストは

$$(p-1/p)nM + S\log_2 p \tag{3}$$

allgather(ring) 部　ring 形式の allgather の通信コストについて説明する．各ノードにメッセージの一部を分配した後, これを allgather によって収集しメッセージを復元する．ring アルゴリズムによって隣のノードに n/p [byte] のメッセージを送信し, $p-1$ stage 行うことで完了する．つまり allgather 部の通信コストは

$$(nM/p + S)(p-1) \tag{4}$$

である．

以上の scatter 部 (式 3) と allgather 部 (式 4) を合算した通信コストは

$$2(p-1/p)nM + (\lceil \log_2 p \rceil + p - 1)S \tag{5}$$

となる．

3　提案手法

マスターノードが動的かつ頻繁に選択されたワーカーノードに対してマルチキャストを行う際, 既存のライブラリに存在する Bcast を利用したグループ通信では, グループを作成するために選択外のノードも含めて同期処理が必要になるため, 動的かつ頻繁なマルチキャストの実現は難しい．そこで, 選択外のノードは作業に一切関与せずに行うのが本手法のマルチキャストとなる．

3.1　実装方針・必要要件

実装方針　本マルチキャストでは, 一般的な集団通信のアルゴリズムとして利用される binomial-tree と binomial-tree_scatter_ring_allgather アルゴリズムをメッセージ長に応じて使用する．このアルゴリズムを利用して, マスターノードによって選択されるワーカーノードのみでマルチキャストを行う．

作業コストの問題　マスターノードはシミュレーションを行う上で重要な役割を担っている．そのため, マルチキャストに関する作業コストを可能な限り抑える必要がある．この解決手法を章 4.4 で解説する．

動的なマルチキャストの問題　MPI 通信を行う上で, 送信ノードと受信ノードは互いに既知でなければならない．マスターノードによって選択されたグループメンバーが動的であるため, 待機状態であったワーカーノードは自身がマルチキャストに参加するのか, また参加した場合, どのノードに配信するのかをその都度知る必要がある．この解決策として配信先リストを利用した手法を採用する．

3.2　前提条件

今回の実装においては, マルチキャストの受信対象となり得るワーカーノードは, 任意のノードからの MPI メッセージを受信可能な状態であると仮定している．

3.3　本手法の全体の流れ

マスターノードはマルチキャストを行うワーカーノードを選択した後, 選択したワーカーノードを配信先リストに登録する．配信先リストの実装法は 4.2 章にて説明する．マスターは配信先リストをメッセージと共にマルチキャストに参加するワーカーノードに配信する．このとき, マスターノードが配信先とするワーカーノードは 1 ノードのみである．マスターノードからデータを受信したワーカーノードがマルチキャストの root となる．root は受信した後, 予め用意されたアルゴリズムにより配信先リストを読み取り, 分配構造を把握し, この構造に従って順次配信先リストとメッセージを配信する．配信の終了後,ack 応答による同期処理により確認する．

4　実装の詳細

4.1　配信アルゴリズムの選択

前章で述べた scatter_allgather のアルゴリズムでは遅延による影響を受けやすいが, メッセージ長による影響を比較的受けにくいことが分かる．そこで本手法では binomial-tree と binomial-tree_scatter_ring_allgather アルゴリズムをメッセージ長によって切り替えながら利用する．　メッセージサイズが 8 [Kbyte] 以下のとき binomial-tree, それ以上のとき binomial-tree_scatter_ring_allgather を利用する．これは MPICH の Bcast の配信アルゴリズムの切り替えになぞらえたものである．

4.2　配信先リストの実装

ここではマルチキャストの対象となるワーカーのメンバーに応じた配信構造の動的構築のための配信先リストについて説明する．マスターはマルチキャストを行うワーカーをリストアップする際, マルチキャスト内での通し番号 (以下より論理番号) を各ノードに割り当てる．MPI により設定された各ノード (あるいはプロセス) に対応するノード番号とマルチキャスト内の論理番号は異なるため, これらを関連付けするためのリストを作成する．このリストは, マルチキャストの受信対象となるノードのランクに対応した bit に 1 を, それ以外に 0 を記録して表現されるものであり, 以下よりビットマップデータと呼ぶ．ビットマップデータを, 例えば実数ではなくビットで表現することにより, ノード数 P

のときそのデータ量は Pbit となる．図 5 はビットマップデータの一例である．図のビットマップデータが示すマルチキャストに参加するノードは,2,4,7,8,9 となる．

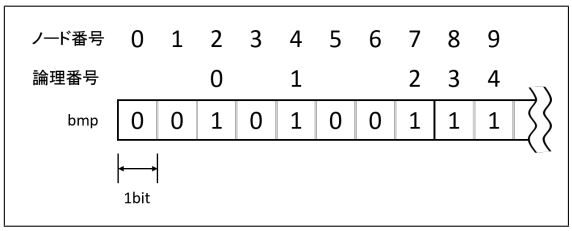

図 5: ビットマップデータ

4.3 分配構造の動的構築

ビットマップデータを受け取った各ワーカーはビットマップデータを読み取って, マルチキャストにおける自身の役割を把握する．送信先ノードの算出に必要な情報は自身の論理番号とマルチキャストメンバーの総数であり, 送信先の論理番号を dest, 自身の論理番号を src, メンバー総数を P, 全ステージ数を $\log_2 P$ とすると, 以下の式から算出できる．

$$dest = src + 2^{s-i} \quad (1 \leq i \leq s) \tag{6}$$

つまり, ビットマップデータから自身の論理番号とメンバーの総数を読み取ることで配信構造を構築し, 分配先の論理番号を抽出することが出来る．その後, 算出した論理番号がどのノード番号かをビットマップデータから読み取り送信先のノード番号を把握できる．

4.4 マスターノード

マスターはメッセージを配信するワーカーを選択した後, ビットマップを作成し, マルチキャストメンバー内の 1 ノードにのみ送信する．具体的にはメンバー内で最もノード番号の小さいノードに配信する．この様子を示したものが図 6 である．この以降, マスター自身はマルチキャストに一切参加しないことで, マスター・ワーカー型環境において多忙であるマスターの作業コストを最小限に抑える．図 6 は 3 つの異なるマルチキャストグループ対してマルチキャストを行う概念図である．マスターノードの作業工程をひとつのマルチキャストに対して 1 ノードとの通信のみとし, 分配作業はグループ内で完結させることで複数のマルチキャストをパイプライン処理することが可能となる．

4.5 ワーカーノード

分配構造が動的変動することから, ワーカーノードでは自らがマルチキャストの対象に含まれるか否か, また, マルチキャストの受信対象となった場合にどのノードから情報を受け取るかは, 実際にビットマップデータを受けとるまで判らない．そこで, マルチキャストの対象に含まれる可能性をもったワーカーは任意のノードからのメッセージを受信可能にする MPI_any_source を送信元として設定して, MPI 受信を可能な状態にしておく．受信したメッセージがマルチキャストに関連するものか否かの識別は MPI メッセージに付随する tag 情報を用いて行う．

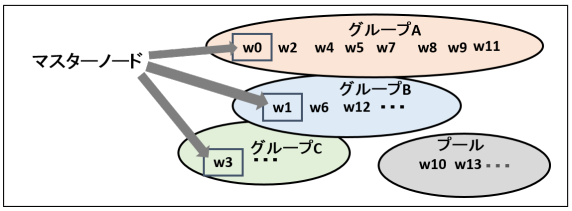

図 6: マルチキャストにおけるマスターノードの作業

4.6 同期処理

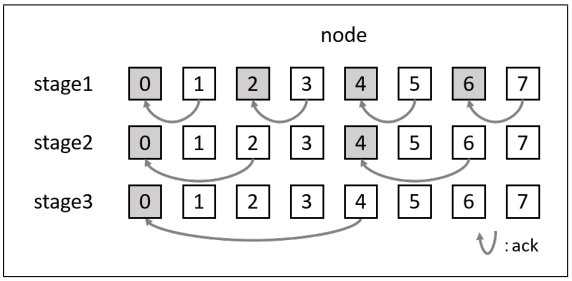

図 7: ack 応答による同期処理

本マルチキャストでは簡単な同期処理を行うために一対一通信による ack 応答を施している．マルチキャストの分配作業を終えた各ノードは ack として終了メッセージを送信する．ack の宛先は root ではなく始めにビットマップデータとメッセージを受信したノードとする．中継ノードは自身がビットマップとメッセージを配信したノードから ack をすべて受信するまで待機した後,ack 応答を行う．この工程を各ノードが行うことにより, 図 7 のような binomial-tree 構造を基盤とした, 葉から root に向かって ack が収集されていく構造が形成される．同期処理は root がすべての ack を受け取り終わることで完了する．

5 性能評価

5.1 目的

前章までに示した動的マルチキャスト機能を実装することで, MPI_Bcast では実現できない動的かつ頻繁なグループ変更に対応可能となった．本章では, 実装し

た動的マルチキャスト機能の実装効率を検証するため, 同じ構成のワーカノード群に予め静的なコミュニケータを設定し MPI_Bcast を行った場合と, 本提案の動的マルチキャストの性能を比較しその検証を行う. 今回の計測では生成するグループを一つのみとし, グループ生成を始めてから配信を終えるまでの時間を実行時間として, その時間を計測する.

また, 計測は全ワーカーノードが指示を待ち受けている状態 (アイドル状態) [1] で行う.

5.2 実験環境

表 1 の諸元を持つ並列計算機 Cray XC30[5, 11] の 256 ノードを用いて評価実験を行なった. 各ノードに 1 ランクを割り当て 256 個のワーカノードを起動し, その中の r ノードにマルチキャストするのに要する実行時間を計測した.r の値としては 8,16,32,64,128 および 256 について検証を行ない, それぞれ $256/r*i(i=0\sim r-1)$ のランク番号をもつノードに配信を行なった. 各 r の値に対して 100 回計測を行い, 上位下位それぞれ 10 %を除いたトリム平均を行なった値を実行時間とした. 評価に用いたメッセージサイズとしては 80 [Kbyte] と 8 [Mbyte] を用いた. これは double 型の 1 次元配列データ 10K 要素ならびに 1M 要素（あるいは 1K × 1K の 2 次元配列）に相当するデータサイズである.

表 1: 実験環境

計算ノード	360node/8640CPU コア
CPU	Intel Xeon E5-2690v3 2.6GHz
メモリ	128GB
総理論演算性能	359.4TFLOPS
ネットワーク	CRAY Dragonfly Topology[10]
コンパイラ	crayc++
MPI	cray-mpich

5.3 実験結果と考察

メッセージサイズが 8 [Mbyte] の場合の実行時間を図 8 に示す. 提案した動的マルチキャスト（図中の Mcast) は MPI_Bcast（図中の Bcast）に対して約 2 倍の速度性能を確認した. 一方で, メッセージサイズが 80 [Kbyte] の場合 (図 9), 提案した動的 Mcast は MPI_Bcast よりも低速となった. これは, 配信ノード数が 2 のべき乗であったために, 中規模データサイズのメッセージに対する MPI_Bcast の第 3 のアルゴリズム [9] である”doubling”(binomial-tree_scatter_doubling_allgather) という通信回数 $O(\log r)$ の手法を用いて MPI_Bcast が最適化を行なったのに対して, Mcast が”ring”アルゴリズムを用いて O(r) の通信回数が必要であったためであると考えられる. しかしながら, 我々の Mcast においても 256 ノードへの配信が約 1.3 [ms] で終了しており, 実用上十分な性能が得られていると考える. また, 我々の Mcast においても”doubling”アルゴリズムを実装するこ とで MPI_Bcast と同等の性能が得られるものと考える.

[1]特定のワーカーノードの作業終了を待機する事による遅延が発生しない.

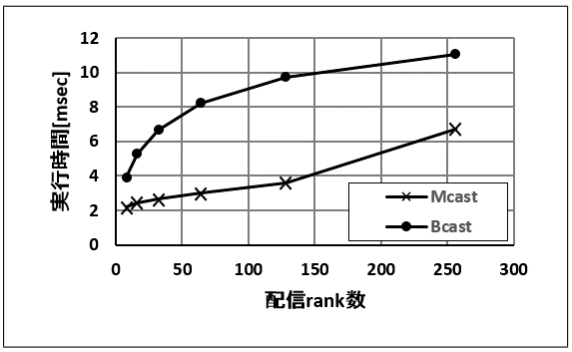

図 8: Cray XC30:メッセージサイズ 8MB での実行時間

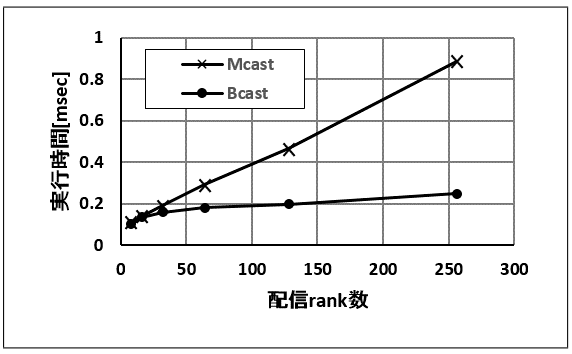

図 9: Cray XC30:メッセージサイズ 80KB での実行時間

6 まとめ

本論文では, 実行時に数十から数百ミリ秒の頻度で頻繁かつ動的にグループメンバが変更される環境下での動的マルチキャストを可能にする手法を検討し, 実装を行なった結果を報告した. 今回の実装においては MPI の Bcast で用いられている 3 つの配信アルゴリズムのうち 2 つに相当するアルゴリズムを用いた動的マルチキャストに応用した. その結果, メッセージサイズが大きい場合には Bcast よりも高速に配信できることを確認するとともに, 中サイズのメッセージに対しては Bcast には性能が劣るものの, シミュレーションシステムへの実装には支障がない性能が得られることを確認した.

今後, 我々の Mcast への”doubling”アルゴリズムの実装や, Mcast の実装に用いたビットマップデータを応

用した各種集団通信ライブラリの実装, さらには, ネットワークトポロジーを考慮した通信の最適化などについて研究を実施することを計画している.

7　今後の課題

我々が扱っているシミュレーションは今のところ大規模並列環境下で実行していないため, 本提案手法を用いることによって大規模並列環境下ではどの程度実時間性が保証されるのか確認がとれていない. 今後, シミュレーションを大規模並列環境下で運用し, 本手法の効果を確認する必要がある. また今回の実装では配信アルゴリズムを 2 種類用意し, メッセージ長に応じてスイッチングすることでメッセージの分配をある程度効率化している. 今後の課題としては, 中規模データサイズのメッセージ配信に最適なアルゴリズム,"doubling" アルゴリズムを実装する. これにより提案した動的マルチキャストが中規模データサイズのメッセージ配信においても, Bcast と同等以上の速度性能になると期待できる. また性能を更に向上させるためにネットワークのトポロジーを考慮する. グリッド環境において, ランク間の距離は異なっていることが多く, ネットワークトポロジーを考慮したブロードキャストの研究が多くされている. 本マルチキャストにおいてもネットワークトポロジーを考慮した配信手法を取り入れることで更なる速度性能向上が期待できる. また, 今回実装したグループ内通信はマルチキャストのみであるが, 作成したビットマップデータは Alltoall,Reduce などの多くの集団通信に対応することが可能である. 本論文で提案したマルチキャストを足がかりとし, 動的かつ頻繁なグループ変更に対応可能な集団通信手法として更なる集団通信の機能追加が見込まれる.

謝辞

本研究の一部は,JSPS 研究費 25280042 の助成を得て実施した. 本研究の一部は, 北陸先端科学技術大学院大学情報社会基盤センターのシステムを利用して実施した. また, 数々の有力な御指導, 御意見を頂いた松山幸雄技術職員をはじめ, 日々様々な面でお世話になった本学森・福間研究室の皆様に深く感謝致します.

参考文献

[1] Message Passing Interface. (http://www.mpi-forum.org 参照:2016-02-04)

[2] 松井 祐太, 岩永 翔太郎, 福間 慎治, 森 眞一郎：操作の連続性を考慮した投機計算 を行うインタラクティブシミュレーションシステム,Vol.2014-HPC-143, No.13, pp.1-7, 2014.

[3] Rajeev Thakur, Rolf Rabenseifner, William Gropp. Optimization of Collective Commjunication Operations in MPICH. International Journal of High Performance Computing Applications, Vol. 19, No. 1, 49-66,2005.

[4] MPICH — High-Performance Portable MPI. (https://www.mpich.org/ 参照:2016-02-04)

[5] Cray XC30. (http://www.cray.com/Assets/PDF/products/xc/CrayXC30Brochure.pdf 参照:2016-02-04)

[6] 蓬来 祐一郎, 西田 晃, 小柳 義夫: 木構造型ネット ワークにおける最適ブロード キャスト スケジューリング, 情報処理学会論文誌:コンピューティングシステム, Vol.45, No.SIG 3(ACS 5)pp.100-108(2004).

[7] 松田　元彦,　石川　裕,　工藤　知宏: グリッド上のコレクティブ通信アルゴリズム, 社団法人　情報処理学会　研究報告書, 2016-HPC-107.pp.257-262(2006).

[8] 千葉 立寛, 遠藤 敏夫, 松岡 聡: グリッド環境におけるマルチレーンを用いた MPI コレクティブ通信アルゴリズム, 情報処理学会論文誌:コンピューティングシステム, Vol.48 No.SIG 8(ACS 18)(2007).

[9] M.Barnett, L.Shuler, S.Gupta, D.G.Payne, R.A.van de Geijn, and J.Watts: Building a high-performance collective communication library. In Supercomputing, pages 107116(1994).

[10] John Kim, William J.Dally, Steve Scott, Dennis Abts: Technology-Driven, Highly-Scalable Dragonfly Topology, ieee computer society, DOI 10.1109/ISCA.2008.19.

[11] JAIST 情報社会基盤センター. (http://isc-w3.jaist.ac.jp/iscenter/ 参照:2016-02-04)