Vol.2013-HPC-138 No.11
2013/2/21

多次元メッシュ/トーラスにおけるプロセス配置に応じた集団通信アルゴリズム選択技術の提案

南里 豪志[1,2,a)]　杉山 裕宣[3,b)]　森江 善之[1,2,c)]

概要：計算環境の大規模化と複雑化に伴い, プロセス配置の多様性や通信衝突の影響が増し, その結果, 通信ライブラリが提供する各機能の実装手法の適切な選択が困難となりつつある. そのため, 実行時の状況に応じて適切な実装手法を自動選択する技術が求められている. 本稿では, そのような通信ライブラリの動的最適化技術の一つとして, 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を考慮した集団通信アルゴリズムの動的選択技術を提案した. この技術は, プロセスが配置されるノード群の形状に応じて有効バイセクションバンド幅を計算し, その結果を基に各アルゴリズムの性能を見積もって, 遅いアルゴリズムを候補から除外する. これにより効率的な動的アルゴリズム選択を実現するものである. Tofu インターコネクト上の実験結果により, 提案技術が低オーバヘッドで動的アルゴリズムの選択を可能にすることを示した.

Proposal of a Method for Selecting Algorithm of Collective Communications on Multi-Dimensional Mesh/Torus

Takeshi Nanri[1,2,a)]　Hironobu Sugiyama[3,b)]　Yoshiyuki Morie[1,2,c)]

Abstract: As the scale of high-performance computing environment increases, selection of appropriate implementation of facilities in communication libraries is becomming difficult task. Because of the wide variety of the process allocations and the improvability of collisions on such system, runtime techniques are required. As one of such techniques, this paper introduces a method for selecting suiltable algorithm of collective communications according to the locations of processes on multi-dimensional mesh/torus topology. The proposed method calculates available bisection bandwidth with consideration of the shape of nodes on which processes are allocated. By applying this bandwidth to the performance models of the candidate algorithms, this method discards the slow algorithms from the list of algorithms that are tested at runtime. Effectiveness of the proposed method is shown by the results of experiments on Tofu interconnect.

1. はじめに

より高い計算能力への要求を満たすため, 今後も計算機のさらなる並列度増大が見込まれている. これに伴い, 高並列化に向けたシステムソフトウェア実装技術の開発が進められている. 特に通信ライブラリは, プログラムのスケーラビリティへの影響が大きいため, 高並列環境での利用に耐える設計と実装が求められる. 高並列環境では, 単に計算機のノード数が増加するだけでなく, メッシュ/トーラスや階層型などの複雑なトポロジが用いられる. これらのトポロジでは, 実行時の状況によって通信性能が変動するため, それに応じて実装手段を適切に選択する技術が重要となる. 通信ライブラリの実装に当たっては, 通常, 多くの機能について複数の実装手段が用意されており, それらの中から適切なものを選んで実行するよう設計されている. 従来, この実装技術の選択ポリシーの決定は, 主にシステム導入時のベンチマーク作業等によって得られる静的な情報を用いて, 人手で行われてきた. しかし, 計算機の大規模化に

[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] 九州大学工学部
Faculty of Engineering, Kyushu University
[a)] nanri@cc.kyushu-u.ac.jp
[b)] 1TE09139N@s.kyushu-u.ac.jp
[c)] morie.yoshiyuki.404@m.kyushu-u.ac.jp

よって最適化の探索空間が爆発的に増大するとともに, トポロジの複雑化によって事前に性能を正確に予測することも困難となっている. さらに今後の計算機では, 従来の通信ライブラリでは考慮されていなかった, メモリ使用量や電力消費等の新しい制約条件が加わる. これらの状況から, 今後の大規模並列計算機においては, 従来の静的な情報のみによる技術では適切な実装手段の選択を行えないと予想される.

そこで我々は, 静的な情報だけでなく, 実行時に得られる動的な情報も活用して, 効率的に最適な実装手段を選択する動的最適化技術の研究開発を進めている. 本稿では, その取り組みの一つとして, 多次元トーラス/メッシュトポロジ向けの集団通信アルゴリズムの選択技術を提案する. この技術では, まずシステムのトポロジ情報と各プロセスが配置されたノードの物理的な座標から, プログラムに割り当てられたノード群の形状を調べ, その形状でのバイセクションバンド幅を算出する. 次に, このバイセクションバンド幅を集団通信アルゴリズムの性能モデルに適用し, 各アルゴリズムの性能を予測する. この予測結果で, 他のアルゴリズムに対して著しく遅いと予想されるアルゴリズムを候補から除外し, 残ったアルゴリズムを実行中に 1 つずつ試して最速のアルゴリズムを選択する. 本稿では, この技術を京コンピュータや Fujitsu PRIMEHPC FX10 で採用されている Tofu インターコネクトの 6 次元メッシュ/トーラストポロジに適用し, 実験により効果を確認する.

2. 通信ライブラリ向け動的最適化技術

通信ライブラリは, 並列プログラムにおけるプロセス間通信のためのインタフェースを提供するソフトウェアである. 既存の通信ライブラリの例としては, MPI(Message Passing Interface) の実装である MPICH, OpenMPI 等の他, ARMCI, GASNet 等が挙げられる. これらの通信ライブラリは, 一対一通信, 集団通信といった基本通信インタフェースを提供する. また, ライブラリによっては, これらに加えて遠隔 Atomic 操作, 複数プロセスにまたがった大域配列の管理等, より高機能なインタフェースも用意している.

これらのインタフェースの実装手段としては, 複数の選択肢が用意されていることが多い. 例えば集団通信には, 複数の実装アルゴリズムが提案されており, しかも, 常に他のどのアルゴリズムよりも高速であるアルゴリズムは存在しないので, 特性の異なるいくつかのアルゴリズムから適宜選択して利用する. また, Send/Receive による一対一通信の一般的なプロトコルとして, 多くの通信ライブラリでは Eager プロトコルと Rendezvous プロトコルを選択的に用いている.

従来, これらの実装手段の切り替えは, 通信ライブラリの導入時に設定された固定の閾値に基づいて行われていた. 例えば集団通信アルゴリズムについては, プロセス数とメッセージサイズについて閾値が設定され, 通信プロトコルについては, メッセージサイズについて閾値が設定されている. このような静的な手法は, 実行時の通信性能を事前に正確に予測できる場合には有効である.

しかし, 計算機の大規模化や複雑化に伴い, このような静的な手法による選択では対応が困難な状況が生じている. 例えば多次元トーラス/メッシュトポロジや, 階層的なトポロジ等, 通信衝突の影響が大きい環境において, 集団通信のアルゴリズムの性能は, 同じプロセス数とメッセージサイズでも, プロセスがトポロジ上にどのように配置されるかによって大きく変動する. そのため, 実行時の状況に応じたアルゴリズム選択が必要となっている. さらに, プロセス数の増加に伴って, 従来のように十分な通信バッファ領域を確保することが困難となっており, 実行時に利用可能なメモリ領域に応じて最適な実装手段を選択する必要がある.

そこで近年, 通信ライブラリの実装手段を動的に選択する技術に関する研究が行われ始めている. 例えば STAR-MPI [1] や ADCL(Abstract Data and Communication Library) [2] は, 集団通信や隣接通信などのパターン通信について, 実行時に各実装を試して最速のものを動的に選択する機能を提供する. また, 京コンピュータの Tofu インターコネクト向けに, 通信相手との通信の頻度に応じて適切な通信プロトコルを選択する技術が開発されている [3].

このような, 通信ライブラリの動的最適化技術は, 今後重要性が増すと予想される. そこで著者らは, 九州大学, 富士通株式会社, 九州先端科学技術研究所による共同研究である, JST CREST の研究領域「ポストペタスケール高性能計算に資するシステムソフトウェアの送出プロジェクト」の研究課題「省メモリ技術と動的最適化技術によるスケーラブル通信ライブラリの開発」 [4] の中で, 動的最適化技術に関する研究を進めている. 本稿で提案する技術は, この取り組みの一つである, 集団通信アルゴリズムの動的選択技術である.

3. 多次元メッシュ/トーラス向け集団通信アルゴリズム動的選択技術

3.1 集団通信アルゴリズム

集団通信とは, 前述の通り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. 例えば Alltoall と呼ばれる集団通信では, 図 1 に示すように, 各ランクが他のランクに対して自分の所有するデータのうち相手のランクに対応する部分を送信する. 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.

集団通信の性能は並列プログラムの処理速度に大きく影響するため, それぞれの集団通信について多数の実装アル

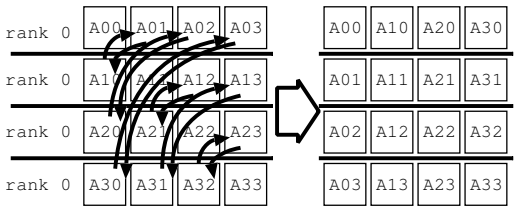

図 1 Alltoall 通信の様子 (4 ランク)

ゴリズムが提案されている. 例えば Alltoall で最も簡単な実装である Simple Spread アルゴリズムは, 各ランクが必要な送信命令を全て発行した後, 必要な受信命令を発行して受信する. このアルゴリズムは, 多数の通信を同時に進行させることが出来るが, 一つのランクに対して複数の送信が同時に発生するため衝突を起こしやすく, 特にプロセス数が多い場合, 性能が低下する.

一方 Ring アルゴリズムは, 通信経路での衝突を回避するため, 同じランクに送信データが集中しないよう同期を取りながら通信を進める. この場合, 衝突の影響は軽微となるが, 同期のコストが問題となるため, 比較的大きなメッセージに適している.

また, Bruck アルゴリズムは, 複数の宛先のデータをまとめて転送することにより, 少ない送受信発行回数で Alltoall を実現するものである. 例えば 8 プロセスで通信を行う場合, プロセス 0 は最初の送信でプロセス 1 に対して, プロセス 1 宛てのデータだけでなく, プロセス 3, 5, 7 宛てのデータも送信する. プロセス 1 は, 2 回目の送信でプロセス 3 に対して, 自分のデータ, 及びプロセス 0 から受信したデータから, プロセス 3 宛とプロセス 7 宛のデータを抽出して送信する. このように複数の通信をまとめることにより, 各プロセスで必要な送受信回数は 3 ($= \log_2 8$) 回となる. しかし, 一回の送受信データ量が 4 倍になるため, Alltoall 全体での総通信量は増加する. そのため, Bruck アルゴリズムは比較的小さいメッセージに適している.

このように様々な集団通信のアルゴリズムが提案されており, メッセージサイズやプロセス数, ネットワークの通信性能等の状況によって相互の優劣が変わる. 特に多次元メッシュ/トーラスでは, 同じプロセス数でもプロセスが割り当てられるノードの形状によって通信衝突の発生頻度が変わり, 通信性能が変動する. そこで著者らは, 多次元メッシュ/トーラスにおける集団通信アルゴリズム選択技術の開発に向けた準備として, 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調査し, 同じプロセス数でも形状によってアルゴリズムの性能の優劣が変わることを確認した [5]. これにより, 状況に応じた適切なアルゴリズムの選択が重要であることを示した.

3.2 提案手法の概要

本研究で提案する集団通信アルゴリズムの選択技術は, 著者らによる Fat Tree 向けの集団通信アルゴリズム選択技術 [6] を多次元メッシュ/トーラスに対応させたものであり, プログラム実行前から集団通信関数呼び出し時までの各段階で, それぞれ以下のように準備や計測, 比較を行うことで, 実行時のプロセス配置を考慮した選択を可能とする.

(1) プログラム実行前:
計算機の基本情報（トポロジ, ルーティング, 通信遅延時間, 帯域幅）を取得する. また, プロセス配置の影響を考慮した各アルゴリズムの性能予測モデルを作成する.

(2) プログラム実行開始時:
各プロセスの, トポロジ上での物理座標を取得し, バイセクションバンド幅を算出する.

(3) 集団通信の最初の呼び出し時:
メッセージサイズと実行開始時に計算したバイセクションバンド幅を性能予測モデルに適用して各アルゴリズムの所要時間を予測する. 予測結果から最速と予想されるアルゴリズムと比較して, 著しく遅いと予想されるアルゴリズムを, 選択の候補から除外する.

(4) 集団通信呼び出し毎 (全候補の実測結果が揃う前):
候補のアルゴリズムの一つを使って集団通信を実行し, 所要時間を実測結果として記録する. 各アルゴリズムについて一定数以上の実測結果が揃った場合, その中から最速だったものを選択する.

(5) 集団通信呼び出し毎 (全候補の実測結果が揃った後):
実測により最速と判断されたアルゴリズムを使って集団通信を実行する.

これらのうち, (1), (2), (3) は, 計算機のトポロジ, 機能, 性能, および各集団通信アルゴリズムの特性に応じて設計し, 実装する必要がある. 本稿では, Tofu インターコネクトに向けた Alltoall 通信についてこれらの実装例を示す. 一方 (4) と (5) は, 計算機やアルゴリズムによる実装の違いはほとんどないため, 今回の実験では Faraj らが提案した STAR-MPI [1] の動的選択機構を用いる.

なお, (2) でアルゴリズムを除外する閾値として, 本稿の実験では, 各アルゴリズムの予想所要時間のうち最小のものの 2 倍の値を用いている. この閾値の妥当性については, 今後アルゴリズムの性能モデルの精度にもとづいた検証を行う予定である.

3.3 集団通信の性能モデル

Alltoall 通信の各アルゴリズムの性能モデルとして, 今回提案する技術では, 性能予測に要するオーバヘッドを低減するため, 非常に簡単なモデルを用いている. まず, 一対一通信の性能モデルとして Hockney モデルを適用する. これは, 個々の一対一通信の所要時間を, メッセージサイズ M に対する線形式 $L + M/B$ で表すものである. このうち遅延時間 L については, ホップ数による影響を無視し, 一定

表 1　Alltoall 通信アルゴリズムの性能モデル

Algorithm	Model
Simple Spread	$(P-1)L+(P-1)M/B$
Ring	$(P-1)(L+M/B)$
Ring with One Barrier	$(P-1)(L+M/B)+(L\log_2 P)$
Ring with MPI Barrier	$(P-1)(L+M/B)+(L(P-1)\log_2 P)$
Ring with Light Barier	$(P-1)(L+M/B)+L(P-1)$
Bruck	$L\log_2 P+PM\log_2 P/(2B)$

時間であると仮定する．一方, 帯域幅 B は, リンク当たりの有効バンド幅 B_0 と, 通信衝突にともなう性能低下率 c の積で算出する．ここで, L と B_0 は, 複数のメッセージサイズについて測定した 2 ノード間の ping-pong 通信の所要時間に対して最小二乗法を適用して導出する．また, c の算出方法は, 次節で説明する．これらの前提を基に作成した Alltoall の各アルゴリズムの性能モデルを表 1 に示す．

なお, 一対一通信の性能モデルとしては, 他に LogP [7], LogGP [8], PLogP (Parameterized Log-P) [9] などが提案されている．特に PLogP は, メッセージサイズの変化に伴う実効帯域幅の変動を表現でき, より高い精度での性能予測が行えると期待できる．さらに, メモリコピー等の通信以外の処理コストや, ホップ数を考慮した遅延時間を加味することで, 精度の向上が見込める．これらの改良については, 予測に要するオーバヘッドとのトレードオフを考慮しながら, 導入を検討する予定である．

3.4　多次元メッシュ/トーラスのプロセス配置に応じた実効バイセクションバンド幅

本稿で提案するアルゴリズム選択技術では, プロセスが配置されたノード群が利用可能な実効バイセクションバンド幅を計算し, アルゴリズムの性能予測モデルに適用する．なお, 今回提案する技術は, 全てのリンクが同じ帯域幅である多次元メッシュ/トーラストポロジを対象とする．また, プロセスが割り当てられるノード群の形状は多次元直方体であり, プロセス間通信はその多次元直方体中のノードのみを経由して行われることを前提とする．

バイセクションバンド幅とは, ネットワークで接続されたシステムにおいて, そのシステムを二等分する切断面で切断されるリンクの合計帯域幅のうち最小のものを指す．多次元メッシュトポロジにおけるバイセクションバンド幅は, ノード数を最長の次元の長さで割ったものに, リンクの帯域幅を乗じることで計算できる．一方, 多次元トーラストポロジの場合, バイセクションバンド幅はメッシュトポロジの場合の 2 倍となる．

そこで提案技術では, まず各プロセスがそれぞれ割り当てられたノードの物理座標を取得し, それを基に多次元直方体の各次元の長さを算出するとともに, それらがトーラスを構成するか否かを判断する．

多次元直方体の各次元の長さは, それぞれの次元における全プロセスの物理座標の最大値と最小値から求める．ここで, ある次元 i について全プロセスの物理座標の最大値が Xi_{max}, 最小値が Xi_{min} であり, その次元がトーラスを構成する場合, システム全体でのその次元の長さを Ni とすると, プロセスが配置されている座標 x_i の範囲としては, この最小値と最大値の内側 ($Xi_{min} \le x_i \le Xi_{max}$) と, この最小値と最大値の外側 ($0 \le x_i \le Xi_{min}, Xi_{max} \le x_i \le Ni$) の二通りが考えられる．そこで, 各プロセスがそれぞれの次元について, 自分の座標が内側にあるか外側にあるかを調べ, その結果を集計することで, プロセス群全体がどちらに配置されているかを判定する．その後, 以下のようにプロセスが配置された多次元直方体の各次元の長さ Li を計算する．

$$Li = \begin{cases} Xi_{max} - Xi_{min} + 1 & \text{内側のとき} \\ Ni - (Xi_{max} - Xi_{min} - 1) & \text{外側のとき} \end{cases} \quad (1)$$

この Li を用いて, メッシュトポロジの場合の D 次元直方体の有効バイセクションバンド幅 B_{bisec} は以下の式で計算できる．

$$B_{bisec} = \frac{\prod_{i=1}^{D} Li}{\max_{i=1}^{D} Li} B_0 \quad (2)$$

なお, $\max_{i=1}^{D} Li$ が奇数である場合の有効バイセクションバンド幅についても, この式で近似することとする．

この有効バイセクションバンド幅を, 最大で全プロセス数の半分の数の通信が共有するとみなし, 衝突を考慮した通信性能の低下率 c を以下で計算する．

$$c = \frac{2}{\max_{i=1}^{D} Li} \quad (3)$$

もし, 長さが最長である次元でトーラスが構成されている場合, c を 2 倍にする．ある次元がトーラスを構成するか否かは, システムの仕様としてその次元がトーラス構造になっているか否かと, プロセスが割り当てられた範囲によって判定する．具体的には, その次元がシステム全体で物理的にトーラス構造になっており, さらにプロセスがその次元全体に配置される場合, プロセスが配置された多次元直方体はその次元についてトーラスを構成する．

3.5　提案技術の Tofu インターコネクトへの適用

本研究で提案するアルゴリズム選択技術を Tofu インターコネクト向けに実装する．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 [10]．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.

Tofu インターコネクトに対応したジョブスケジューラでは, ジョブを投入する際, 利用者は使用するノード数のみを指定するか, もしくはノードの形状を $x \times y \times z$ の 3 次元形状で指定するか, 選択できる. 3 次元形状が指定された場合, x, y, z それぞれに対し, 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 のいずれかを, ノードの空き状況に応じて割り当てる. このように, 同じノード数や同じ 3 次元形状を指定しても, 実際に割り当てられる形状はシステムの混雑状況によって変化する.

これに対して本研究で提案する手法は, ジョブ投入時に利用者が指定するノード数や形状ではなく, ジョブ実行時に割り当てられた 6 次元直方体の形状にもとづいてアルゴリズムの性能を予測することが出来る.

なお, 故障ノードがある場合, そのノードを迂回して割り当てるため, 割り当てられるノード群の形状が 6 次元直方体とならない可能性がある. その場合の実効バイセクションバンド幅の算出方法については, 未対応である. また, 物理的な 6 次元における X, Y, Z 軸について, プロセスが割り当てられた範囲がシステム全体の半分を超える場合, 割り当てられた 6 次元直方体の内部だけでなく外部のノードも経由して通信が行われるため, 実効バイセクションバンド幅が前節に示した式による見積もりとは異なる. 今後, これらの場合についても対応が可能なように算出アルゴリズムの改良が必要である.

4. 実験

4.1 実験環境

今回実装した動的アルゴリズム選択を行う Alltoall 通信関数の性能を検証するため, Alltoall 通信を 200 回呼び出すプログラムを用いて所要時間の計測を行った. このプログラムは STAR-MPI のベンチマークプログラムとして提供されているものである. このプログラムを, プロセス形状とメッセージサイズを変えながらシステムに投入した.

実験に使用したシステムは, 九州大学情報基盤研究開発センターの Fujitsu PRIMEHPC FX10 である. このシステムはノード間インターコネクトとして Tofu を採用しており, 各計算ノードには Fujitsu SPARC64 IXfx (1.848GHz, 16 コア) を 1 基と, 32GB のメモリを搭載している.

なお, 前節で述べたとおり, 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 3 次元で指定することが出来る. 時間の関係で, 今回の実験で試すことが出来たのは $4 \times 3 \times 2$, $8 \times 3 \times 1$, $4 \times 3 \times 4$ の 3 通りである.

4.2 実験結果

各形状での計測結果を, 図 2, 図 3, 図 4 にそれぞれ示す. 横軸はメッセージサイズ, 縦軸は所要時間である. 測定結果としては, まずアルゴリズムの選択技術を用いた場合の, 200 回分の Alltoall 通信の平均時間として, 本稿で提案する手法を用いた場合 (Topo), トポロジを考慮せずに候補を絞り込んだ場合 (Notopo), 候補を絞り込まなかった場合 (Nomodel) を示している. さらに, Alltoall の各アルゴリズムについても, Bruck, Ring, Ring LB(Ring with Light Barrier), Ring MB(Ring with MPI Barrier), Ring OB(Ring with One Barrier), Simple(Simple Spread) のそれぞれについて, Nomodel の計測時に取得した平均所要時間を示している.

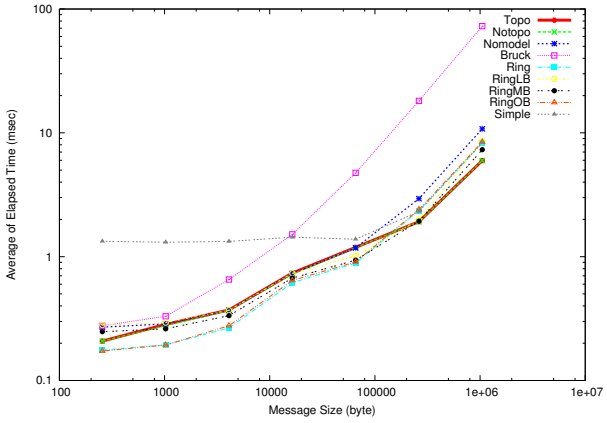

図 2 各アルゴリズムおよび動的選択による Alltoall の所要時間 ($4 \times 3 \times 2$))

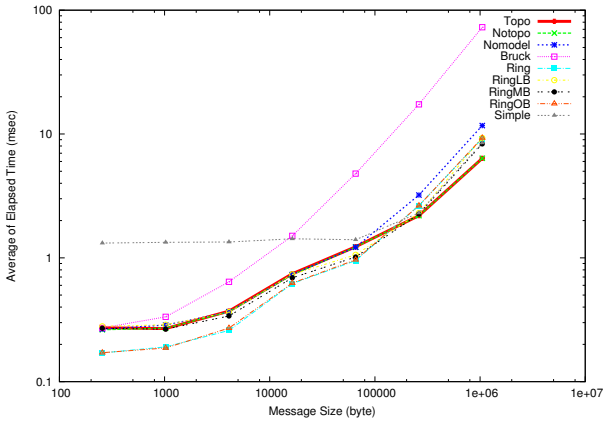

図 3 各アルゴリズムおよび動的選択による Alltoall の所要時間 ($8 \times 3 \times 1$))

今回試した形状の内, 同じプロセス数である $4 \times 3 \times 2$ と $8 \times 3 \times 1$ では, アルゴリズムの優劣はほぼ同じであった. $4 \times 3 \times 4$ は, 若干違いがあるものの, 傾向としてはやはり他

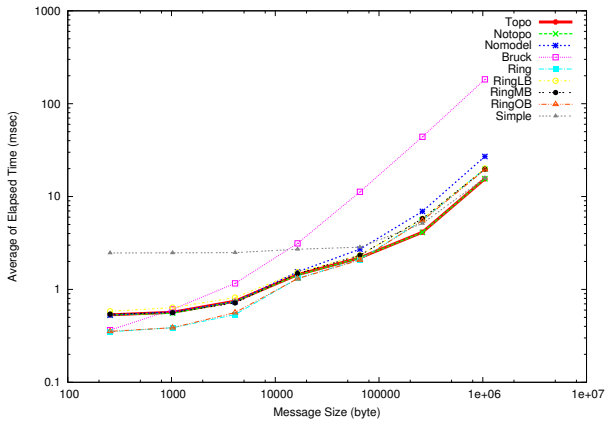

図 4 各アルゴリズムおよび動的選択による Alltoall の所要時間 $(4 \times 3 \times 4)$)

の二つと大きな違いは無かった. これは, ノード数が少なく, 衝突の影響が少なかったためと考えられる. この場合, ノード形状を考慮に入れた場合と考慮に入れない場合での予測性能の差が小さく, 結果的にどちらの場合でも候補から除外されるアルゴリズムはほとんど同じであった. そのため, Topo と Notopo の間に性能の違いは見られない. なお, これは同時に, ノード形状を考慮に入れることによるオーバヘッドが, 無視できるほど小さいことも示している.

一方, メッセージサイズが大きい範囲で, 候補を絞り込まない Nomodel に対して Topo と Notopo の所要時間が短縮されているのが分かる. これは, 特にメッセージサイズが大きいところで, Bruck のように大きいメッセージサイズに向かないアルゴリズムを適切に除外できたためである. その結果, メッセージサイズが 256KB 以上の範囲では, Topo と Notopo は最速のアルゴリズムとほぼ同等性能を示している.

しかし, メッセージサイズが小さい範囲では, アルゴリズムを動的に選択する手法はどれも, 最速のアルゴリズムに対して, 最悪で 1.5 倍程度の時間を要している. これは, Simple Spread のように小メッセージサイズに向いていないアルゴリズムを試すことによるものである. Topo や Notopo では, それらのアルゴリズムの所要時間を実際よりも短く見積もり, 候補から除外することが出来なかった. そのため, 性能モデルの予測精度向上によって, この範囲の性能向上が見込めることが分かる.

5. 関連研究

集団通信高速化に向け, 様々なアルゴリズムが提案されてきた [11], [12], [13], [14], [15]. 特に近年では, 特定のトポロジに特化したアルゴリズムの提案が行われている [16].

また, これらのアルゴリズムから最適なものを選択する技術についても盛んに研究されている. Hamid らは, Ethernet と Myrinet に向けた MPICH における最適なアルゴリズム選択の手法として, システムに応じて閾値を調整する技術を提案している [17]. また, Sivac-Grbovi'c らは, Quadtree を用いることによって, メッセージサイズとランク数に対して効率良く適切なアルゴリズムを探索する手法を提案している [18]. しかしこれらは, 実行前に得られる性能情報を用いた選択手法であり, 実行時の状況によって性能が変動する場合に対応できない.

一方, 動的な手法として Faraj らは, 同じパラメータで何度も呼び出される集団通信について, 最初の数十回を使って用意されている各アルゴリズムを試し, その結果判明した最速のアルゴリズムをそれ以降の呼び出しで利用する STAR-MPI ライブラリを提案した [1]. しかし, これは非常に遅いアルゴリズムを試すことによる性能低下が問題である. また, 同様の手法を, 隣接通信や利用者が定義する任意のパターン通信に適用した ADCL(Abstract Data and Communication Library) も開発されている [2]. これは, パターン通信に対して多数の実装関数を用意し, それらを Star-MPI と同様に実行中に試すものである. さらに ADCL では, 各実装関数に複数の属性を持たせておき, ある属性について最適と思われる値が判明した時点で, 以降, その属性について他の値を持つ実装関数を試行対象から除外することにより, 試す実装関数の数を減らし, オーバヘッドの低減を図っている. しかし, この手法は各実装を特徴付ける属性値を設定可能な場合に限られており, 集団通信のアルゴリズムの相違のように数値化が難しい場合には適用できない. また, Nishtala は集団通信アルゴリズムの性能予測モデルを作成し, それにもとづいて対象アルゴリズムを絞り込んだうえで, 最適なアルゴリズムを探索する技術を, 通信ライブラリ GASNet 上に実装した [19]. しかし, この手法は性能予測にプロセス配置による影響が考慮されておらず精度に問題があるうえ, 各アルゴリズムの計測を最初の集団通信呼び出し時にまとめて行うため, オーバヘッドが大きい.

6. まとめと今後の課題

本稿では, 通信ライブラリの動的最適化技術の重要性を示し, その例として, 多次元メッシュ/トーラストポロジ向けの集団通信アルゴリズムの動的選択技術を提案した. この技術は, プロセスが配置されたノード群の形状から有効バイセクションバンド幅を計算し, それをもとに各アルゴリズムの所要時間を見積もることでアルゴリズムの候補を絞り込み, 効率的なアルゴリズム選択を実現する. 今回の実験では, ノード数が少なかったため, 提案手法でプロセス配置の形状を考慮することによる効果が示せなかった. しかし, 少ないオーバヘッドで提案手法を実現できることを確認した. また, アルゴリズムを絞り込むことにより, 効率的にアルゴリズム選択を行えることも分かった.

今後, より大きなジョブで提案手法の効果を確認するとともに, 他の集団通信にも適用する. また, アルゴリズムの

性能モデルについて, 精度と計算コストのトレードオフを考慮しながら改良し, アルゴリズム動的選択のさらなる効率化を図る.

参考文献

[1] Faraj, A., Yuan, X. and Lowenthal, D.: STAR-MPI: Self Tuned Adaptive Routines for MPI Collective Communications, *Proceedings of International Conference on Supercomputing*, pp. 199–208 (2006).

[2] Gabriel, E. and Huang, S.: Runtime Optimization of Application Level Communication Patterns, *12th International Workshop on High-Level Parallel Programming Models and Supportive Environments*, (2007).

[3] 三浦 健一: スーパーコンピュータ「京」での MPI の実装と評価, サイエンティフィック・システム研究会 科学技術計算分科会 2012 年度会合, `http://www.ssken.gr.jp/MAINSITE/download/newsletter/2012/20121024-sci-2/lecture-04/SSKEN_sci2012-2_miura_summary.pdf` (2012).

[4] ACE プロジェクトホームページ, `http://ace-project.cc.kyushu-u.ac.jp`

[5] 南里 豪志: 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, 第 136 回ハイパフォーマンスコンピューティング研究会 (2012).

[6] 南里 豪志, 黒川 原佳: ランク配置に応じた集団通信アルゴリズム動的選択技術の提案, 第 133 回ハイパフォーマンスコンピューティング研究会 (2012).

[7] Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K. E., Santos, E., Subramonian, R. and von Eiken, T.: LogP : Towards a Realistic Model of Parallel Computation, *Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Pratice of Parallel Programming*, (1993).

[8] Alexandrov, A., Ionescu, M. F., Schauser, K. E. and Scheiman, C.: LogGP : Incorporating Long Messages into the LogP model - One Step Closer Towards a Realistic Model for Parallel Computation, *Proceedings of the 7th annual ACM symposium on Parallel Algorithms and Architectures*, pp. 95–105, (1995).

[9] Kielmann, T., Bal, H. and Verstoep, K.: Fast measurement of LogP parameters for message passing platforms, *International Parallel & Distributed Processing Symposium*, LNCS 1800, pp. 1176–1183, (2000).

[10] 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).

[11] Mitra, P., Payne, D., Shuler, L., van de Geijn, R. and Watts, J.: Fast Collective Communication Libraries, *International Supercomputing User's Group Meeting*, (1995).

[12] Barnet, M., Gupta, S., Payne, D., Shuler, L., van de Geijn, R. and Watts, J.: Interprocessor Collective Library, *Scalable High Performance Computing Conference*, pp. 357–364, (1994).

[13] Bala, V., Bruck, J., Cypher, R., Elustondo, P., Ho, A., Ho, C. T., Kipnis S. and Snir, M.: A portable and Tunable Collective Communication Library for Scalable Computers, *IEEE Transaction on Parallel and Distributed Computing* , Vol. 6, No. 2, pp. 154–164, (1995).

[14] Rabenseifner. R.: Optimization of Collective Reduction Operations, *International Converence on Computational Science*, LNCS 3036, pp. 1–9 (2004).

[15] 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).

[16] 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).

[17] 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).

[18] Sivac-Grbovi'c, J. P., Fagg, G. E., Angskun, T., Bosilca, G. and Dongarra, J.: MPI Collective Algorithm Selection and Quadtree Encoding, *In Proceedings of the 13th European PVM/MPI User's Group Meeting*, pp. 40–48 (2006).

[19] Nishtala, R.: Automatically Tuning Collective Communication for One-Sided Programming Models, PhDThesis, UC Berkeley, Berkeley (2009).