Vol.2012-HPC-133 No.13
2012/3/26

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

南 里 豪 志[†1,†2] 黒 川 原 佳[†3]

計算機が大規模化, 複雑化する中で, 通信ライブラリの実装手段を適切に選択する技術が重要となっている. 特に, 計算機の通信性能がメッセージサイズやランク数だけでなく, 計算ノードの相対位置によって大きく変動するようになったため, 従来のように, ベンチマークプログラム等を用いて静的に閾値を決定する手法では, 適切な選択が行えない. そこで本研究では, 通信ライブラリの実装手段を動的に選択する技術のひとつとして, 実行時の集団通信のアルゴリズム動的選択技術を提案する. この技術では, ランク配置情報とトポロジの特性から各アルゴリズムの性能を予測し, 選択対象のアルゴリズムを絞り込んだうえで, 実際の集団通信呼び出し時に各アルゴリズムを試して最適なアルゴリズムを選択する. 本稿ではこの技術の実装例として Alltoall 通信を作成し, ベンチマークプログラムを用いて, 提案手法の有効性を確認した.

Topology-Aware Runtime Algorithm Selection of Collective Communications

Takeshi Nanri[†1,†2] and Motoyoshi Kurokawa[†3]

As the size and the complexity of computers increase, selection and tuning the technologies for implementing communication libraries have become important issues. The communication performance of these systems depends not only on the message size and the number of ranks, but also the relative locations of ranks and collisions among messages. Because of the wide variety of the rank allocation and improvability of the collisions, traditional static method cannot choose the appropriate technology for the given situation. As a dynamic technique for choosing implementation technologies, this paper introduces a method that selects a suitable algorithm of collective communications at runtime. At the first invocation of a collective communication, this method predicts the performance of each algorithm from the information of rank allocation and network topology. Then it discards the algorithms that are predicted much slower than others. After that, it examines each algorithm at each invocation of the collective communication to measure its performance. Finally, it chooses the fastest one as the algorithm that is used for the remaining invocations. Results of some preliminary experiments showed the efficiency of the proposed method.

1. 背 景

計算機の大規模化, 複雑化にともない, その上で動作するソフトウェアのチューニングが困難となってきている. これは, 検討すべきパラメータ数が増加するとともに, 個々のパラメータが取りうる値の範囲も広がっているためである. この問題は, 特に通信ライブラリにおいて顕著である. 通信ライブラリの実装では, 提供するそれぞれの機能について, いくつかの実装技術の中から適切なものを選択したり調整したりしている. 例えば, 多くの通信ライブラリでは通信時に使用するプロトコルとして, メッセージサイズが小さい場合は Eager Protocol を, メッセージサイズが大きい場合は Rendezvous Protocol を選択することにより, 通信バッファサイズの制約内で出来るだけ高い性能の実現を図っている. また, ほとんどの集団通信の実装には複数のアルゴリズムが用意されており, その中から最も高速と思われるものを選択する. さらに, 集団通信の内部でパイプライン転送を行う場合, そのセグメントサイズを, 最も高い性能が得られるように調整する. 従来の通信ライブラリでは, このような実装技術の選択や調整を, システム導入時のベンチマーク結果にもとづいて設定したランク数とメッセージサイズに対する閾値を用いて行った.

しかし, 最近の大規模並列計算機では, このように静的に設定した閾値では十分な性能が得られなくなってきている. これは, インターコネクトのトポロジとして, Fat Tree やトーラス, メッシュのように, 計算ノードの相対位置や他の通信との衝突の頻度等によって通信性能が変動するものが採用されることが増えたためである. その結果, 同じランク数とメッセージサイズであっても, 最適な実装技術が同じであるとは限らなくなった. また, 計算ノードの相対位置のように取りうる範囲が多次元であるパラメータや, 通信衝突のようにそもそも予測が困難である事象まで考慮に入れて, 導入時のベンチマークだけで適切な閾値を設定することは困難である. そこで著者らは現在, 将来の大規模並列計算機に向けて著者らが開発中のスケーラブルな通信ライブラリ ACE (Advanced Communication library for Exa) における要素技術の一つとして, 実行時の状況に応じて通信ライブラリの実装技術の選択や

†1 九州大学
Kyushu University
†2 JST CREST
†3 理化学研究所
RIKEN, Japan

調整を行う動的最適化技術の研究を進めている.

本稿では, この通信ライブラリの動的最適化技術の一例として, 集団通信アルゴリズムの動的選択技術を提案し, 実際にその技術に基づいて実装した通信関数で, 提案技術の有用性を確認する. 並列プログラム中で複数のランクが参加してデータの交換や集約, 分散を行う集団通信は, 多くの科学技術計算で多用されるため, 状況に応じた適切なアルゴリズム選択が重要である. 本稿で提案する技術は, 実行開始時に得られるランク配置の情報をもとに候補となるアルゴリズムを絞り込んだ上で, 集団通信呼び出し時に各アルゴリズムを試すことにより最適なものを選択する. これにより, 低オーバヘッドで高精度なアルゴリズム選択の実現を図る. アルゴリズムの絞り込みでは, 予め調査しておいたトポロジの情報と, 実行開始後に取得するランク配置の情報をもとに, アルゴリズムの性能予測モデルを用いて各アルゴリズムの所要時間を見積もり, 他のアルゴリズムよりも大幅に遅いと予測されるアルゴリズムを選択肢から除外する. その後, 残った候補アルゴリズムを, 1 回の呼び出し時に 1 つずつ試し, その際の所要時間を比較することにより, 最速なアルゴリズムを選択する. 本稿では Fat Tree を対象として, この技術による Alltoall 通信を実装する. また, 実験によりこの Alltoall 通信関数の基本性能を検証し, 提案手法の効果を確認する.

2. 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. このうち本稿では, Alltoall と呼ばれる通信について扱う. これは, 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ものである. 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, 特にメッセージが大きい場合に速度が低下する.

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 ランクで通信を行う場合, ランク 0 は最初の送信でランク 1 に対して, ランク 1 宛のデータだけでなく, ランク 3, 5, 7 のデータも送信する. ランク 1 は, 2 回目の送信でランク 3 に対して, 自分のデータ, 及びランク 0 から受信したデータから, ランク 3 宛とランク 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, 各ランク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, しかも状況によって相互の優劣が変わる. 上記のメッセージサイズだけでなく, ランク数や通信性能等によっても, 最適なアルゴリズムは変動する. そのため, 状況に応じて適切なアルゴリズムを選択することが重要である.

3. ランク配置による集団通信アルゴリズムの性能変動

本稿では, 集団通信アルゴリズムの選択において, 候補のアルゴリズムをランク配置とトポロジに基づいて絞り込むことを提案する. そこで予備調査として, 集団通信アルゴリズムの性能へのランク配置の影響を計測した. 実験環境としては, 理化学研究所の RIKEN Integrated Cluster of Clusters (RICC) を用いた. RICC は, Intel Xeon 5570 (2.93GHz, 4 コア) を 2 基と, 12GB の主記憶を搭載した計算ノードで構成され, 計算ノード間は InfiniBand を用いたインターコネクトネットワークで接続されている. RICC のインターコネクトネットワークの概要を図 1 に示す.

予備調査では, RICC のインターコネクトネットワークのトポロジと経路選択ポリシーを考慮して設定したいくつかの配置パターンについてアルゴリズムの性能を比較する. この

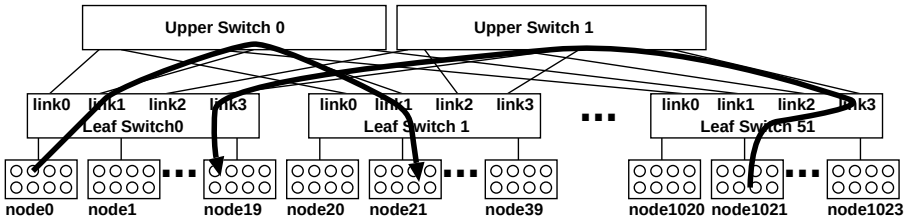

図 1 RICC のネットワーク構成と経路選択ポリシー

ネットワークは 2 階層のスイッチ群による Fat Tree トポロジで構成されており, 下位の 52 台の Leaf Switch が上位の 2 台の Upper Switch と 2 本ずつ, 合計 4 本のリンクで接続されている. この Leaf Switch に 20 台ずつの計算ノードが接続され, 合計で 1024 ノードの PC クラスタを構成している. 各ノードには一意に番号が付けられており, 0 番目の Leaf Switch には 0〜19, 1 番目の Leaf Switch には 20〜39 というように, Leaf Switch 内に連続した番号のノードが配置されている. 一方, 各 Leaf Switch から Upper Switch への 4 本のリンクには, 0〜3 の番号が付けられている. このトポロジにおける Leaf Switch を跨る通信では, Leaf Switch から Upper Switch へのリンクのうち, その通信の宛先ノードの番号を 4 で割った余りの数字と一致する番号のリンクを経由して, 目的のノードにメッセージが送信される.

このようなトポロジでは, ランクが配置されるノードの位置によって各リンクの使用頻度に偏りが生じる. そこで, この偏りによる影響を検証するため, 128 ランクを以下の 5 パターンに従って配置し, 各 Alltoall アルゴリズムの所要時間を計測した.

パターン 0 32 個の Leaf Switch で, それぞれノード番号の小さい方から順に 4 の倍数の番号を持つ 4 ノードにランクを配置する. Leaf Switch 内の 4 ランクが, Upper Switch との間の 1 本のリンクを共有する.

パターン 1 32 個の Leaf Switch で, それぞれノード番号の小さい方から順に 2 の倍数の番号を持つ 4 ノードにランクを配置する. Leaf Switch 内の 2 ランクずつが, Upper Switch との間の 1 本のリンクを共有する.

パターン 2 32 個の Leaf Switch で, それぞれノード番号の小さい 4 ノードにランクを配置する. 各ランクが 1 本ずつ Upper Switch との間のリンクを使用し, 競合は起こらない.

パターン 3 16 個の Leaf Switch で, それぞれノード番号の小さい 8 ノードにランクを配置する. Leaf Switch 内の 2 ランクずつが, Upper Switch との間の 1 本のリンクを共有する.

パターン 4 8 個の Leaf Switch で, それぞれノード番号の小さい 16 ノードにランクを配置する. Leaf Switch 内の 4 ランクずつが, Upper Switch との間の 1 本のリンクを共有する.

メッセージサイズを 16KB と 1MB とした場合の各アルゴリズムの所要時間を計測した結果を, 図 2, 3 に示す. 図で横軸はランク配置パターンの番号, 縦軸は所要時間である. また, それぞれの線が, 各アルゴリズムの所要時間である.

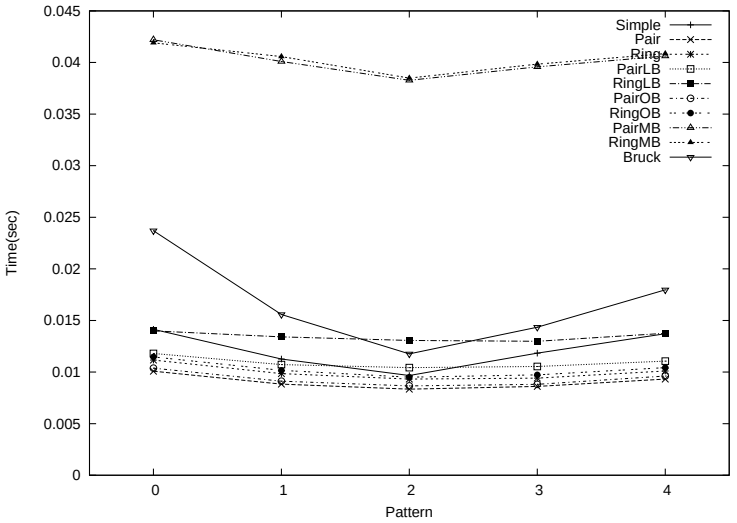

図 2 パターンによる集団通信性能の変動 (16KB))

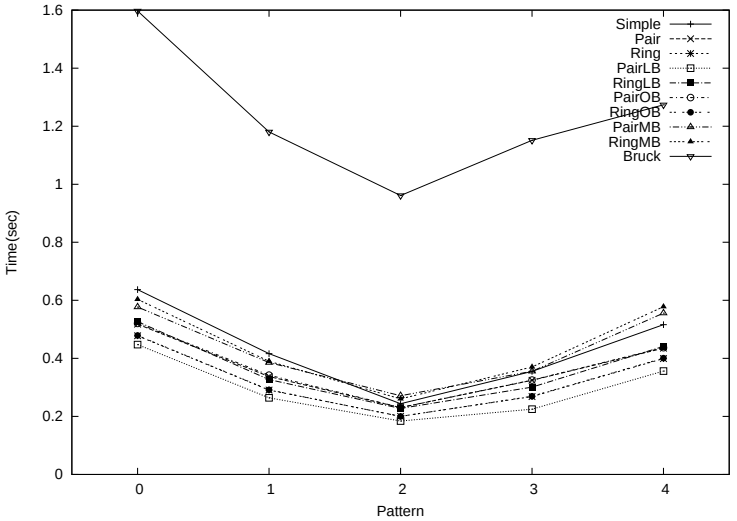

図 3 パターンによる集団通信性能の変動 (1MB))

図 2 より, 特に Bruck でパターン毎の性能の変動が大きいことが分かる. パターン 0 やパターン 4 では最速のアルゴリズムに対して 2 倍以上の時間を要しているのに対し, パターン 2 では最速のアルゴリズムに近い時間となっている. これは, 前述の通りランク数 P に対して, 他のアルゴリズムが 1 回あたり 16KB の一対一通信を P-1 回繰り返して Alltoall 通信を実現しているのに対し Bruck は 1 回あたり 16KB*P/2 の一対一通信を $\log_2 P$ 回繰り返しているため, ランク配置パターンによるバンド幅の変化に影響されたためである. なお, リンクあたりのランク数が同じであるパターン 0 と 4 やパターン 1 と 3 で所要時間が異なるのは, パターン 3 と 4 は, パターン 0 と 1 に比べ, 同じ Leaf Switch 内に配置されるランク数が多く, Leaf Switch をまたぐ通信の数が少ないためである. 一方, 図 3 に示す通り, メッセージサイズが 1MB の場合, 他のアルゴリズムでもランク配置パターンによって性能の変動が見られるようになる.

以上の調査により, ランク配置によってアルゴリズムの性能が大きく変動することが分かった. さらに, ランク配置の性能への影響がアルゴリズムによって異なることを確認した. このように, アルゴリズムの優劣がランク配置に依存するため, ランク配置を考慮したアルゴリズム選択が重要である.

4. ランク配置を考慮した集団通信アルゴリズム動的選択手法

4.1 アルゴリズム選択の概要

本研究で提案する集団通信アルゴリズムの選択手法では, 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, 実行時のランク配置を考慮した選択を可能とする.

(1) プログラム実行前:
計算機のトポロジを調査し, ランク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後:
ランク配置の情報を取得する.

(3) 集団通信の最初の呼び出し時:
ランク配置情報を性能予測モデルに適用してアルゴリズムの所要時間を予測する. 最速と予想されるアルゴリズムに対し, 予測所要時間が予め定めた閾値を超えるアルゴリズムを, 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, RICC 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.

4.2 動的アルゴリズム選択技術を用いた Fat Tree 向け Alltoall 通信関数

提案する動的アルゴリズム選択技術の実用例として, RICC における Fat Tree を対象とした Alltoall 通信関数を実装する. ここで重要となるのは, 各アルゴリズムの性能予測モデルである.

基本的に Alltoall 通信は, 参加する全ランク間で全対全通信を行う. そこで今回作成する Alltoall 通信関数では, 全ランクが一様に通信を行う際の平均帯域幅を計算し, その値を用いて各アルゴリズムの性能を予測する. この平均帯域幅は, あるランクと他の全ランクの間で通信を行う際の個々の通信の帯域幅の平均値とする. ここで個々の通信の帯域幅としては, それぞれの通信時に経由する各リンクの基本帯域幅を, そのリンクを同時に使用する他のランクによる通信の数の期待値で割った値を用いる. Fat Tree では, Leaf Switch と Upper Switch の間のリンクにおいて, 同時に使用するランク数が通信の方向によって異なる. Leaf Switch から Upper Switch への方向は, その Leaf Switch から他の Leaf Switch への通信のうち, 宛先ノード番号を Upper Switch で割った値がそのリンク番号と一致するものが使用する. 一方 Upper Switch から Leaf Switch への方向は, 他の Leaf Switch からこの Leaf Switch への通信のうち, 宛先ノード番号を 4 で割った値がそのリンク番号と一致するものが使用する. このうち今回は, Upper Switch から Leaf Switch への方向の通信を対象に, 平均帯域幅を算出する. この平均帯域幅は Leaf Switch と Upper Switch の間のリンク毎に変動するため, 全てのリンクについて計算した後, その中で最も小さい値を, システム全体の通信を律速する平均帯域幅として用いる.

次に, 各リンクの平均帯域幅の算出方法を示す. ここで, ノード数を N_{node}, ノード内ランク数を P_n, Leaf Switch 数を N_{LS}, i 番目の Leaf Switch を LS_i, LS_i と Upper Switch の

間のリンクのうち j 番目のものを UL_{ij}, プログラムが使用するノードのうち LS_i に配置されたものの数を N_i, そのうちノード番号を 4 で割った値が j であるものを NUL_{ij} とする. また, 今回はノード内, Leaf Switch 内, Leaf Switch 間で基準となる帯域幅は同じ値 B とする.

まず, LS_i のノードの一つに割り当てられたランクが他の全ランクから受信する場合の各通信の帯域幅を見積もる. 他のランクからの受信のうち, ノード内のランクから受信する場合の帯域幅は他の通信に妨げられないと仮定し, B のままとする. 一方, Leaf Switch 内のノード間通信は, Leaf Switch からノードへの 1 本のリンクをノード内の全ランクの受信で共有するため, 平均帯域幅は B/P_n とする. また, Leaf Switch を跨ぐノード間通信の受信では, 同じ Leaf Switch 内の各ランクの通信のうち, 同じリンクを経由して他の Leaf Switch のノードから受信するものと 1 本のリンクを共有する. リンク UL_{ij} を同時に使って受信する LS_i 内のランク数の期待値 RCV_{ij} は, 以下で計算できる.

$RCV_{ij} = 1 + (NUL_{ij} * P_n - 1) * (N_{node} - N_i) * P_n/(N_{node} * P_n - 1)$

これらより, UL_{ij} を経由して受信する LS_i 内のランクについて, 平均帯域幅 BR_{ij} を以下で計算する.

$BS_{ij} = B/((P_n - 1 + (N_i - 1) * P_n * P_n + (N_{node} - N_i) * RCV_{ij})/(N_{node} * P_n - 1))$

これを, $0 \leq i < N_{LS}$, $0 \leq j < 4$, の各 i, j で計算し, その最小値 B_{min} をシステムの平均帯域幅とする. 各アルゴリズムの性能は, 平均帯域幅 B_{min} をそれぞれの性能モデルに適用して予測する. 今回の実装に用いた Alltoall の各アルゴリズムの性能モデルを表 1 に示す.

表 1 Alltoall 通信アルゴリズムの性能モデル

Algorithm	Model
Simple Spread	$(P-1) * L + (P-1) * M/B_{min}$
Ring	$(P-1) * (L + M/B_{min})$
Ring with One Barrier	$(P-1) * (L + M/B_{min}) + (L * log_2 P)$
Ring with MPI Barrier	$(P-1) * (L + M/B_{min}) + (L * (P-1) * log_2 P)$
Pair with Light Barier	$(P-1) * (L + M/B_{min}) + L * (P-1)$
Pair	$(P-1) * (L + M/B_{min})$
Pair with One Barrier	$(P-1) * (L + M/B_{min}) + (L * log_2 P)$
Ring with MPI Barrier	$(P-1) * (L + M/B_{min}) + (L * (P-1) * log_2 P)$
Ring with Light Barier	$(P-1) * (L + M/B_{min}) + L * (P-1)$
Bruck	$L * log_2 P + P * M * log_2 P/(2 * B_{min})2$

これらのモデルは, Hockney モデルによる一対一通信の性能モデルをもとにしている. Hockney モデルでは, 個々の一対一通信の所要時間を, メッセージサイズ M, 遅延時間 L, および帯域幅 B を用いて $L + M/B$ と表す. これを各アルゴリズム内の一対一通信に適用して, 性能モデルを作成した. なお, 一対一通信の性能モデルとしては, 他に LogP[2], LogGP[3], PLogP (Parameterized Log-P)[4] 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ため, 今後適用を検討する.

今回作成した Alltoall 通信関数の実装では, まず事前に基準となる帯域幅と遅延時間を計測した. これは, 2 ランク間の一対一通信を繰り返して計測した結果を用いた. また, トポロジ情報は事前に調査し, その結果を性能予測手法に反映させた. 具体的には, Upper Switch と Leaf Switch の間のリンクの構成, および Leaf Switch にまたがる通信の経路選択ポリシーを調べ, それに基づいて帯域幅を調整する性能予測モデルを作成した.

一方, ランク配置情報の取得は Alltoall 関数の最初の呼び出し時に行う. これを実行開始時に行うこともできるが, 今回は Alltoall 通信関数の試験実装が目的であったため, この関数内でランク配置情報の取得を行っている.

このランク配置情報と, 事前に作成していた性能予測を用いて, 最初の呼び出し時にアルゴリズムを絞り込む. 今回の実装では, 用意されているアルゴリズムのうち, 最も所要時間が短いと予測されたものに対して, 2 倍以上の所要時間がかかると予測されたアルゴリズムを, アルゴリズム選択の候補から除外する. アルゴリズムを除外する閾値は, 今後システムの性能安定性や性能予測モデルの精度を確認しながら調整する予定である.

候補アルゴリズムを絞り込んだ後の最速アルゴリズムの選択には STAR-MPI を用いる. STAR-MPI の処理は, 以下の 2 つのフェーズに分けて行われる.

Learning フェーズ :

集団通信が呼ばれると, 選択候補のアルゴリズムのうちの一つを用いて実行し, 結果を返す. その際所要時間を計測し, 記録する. 全ての候補アルゴリズムについて規定回数の計測が終了するまで, 各集団通信呼び出しに対してこのフェーズを実行する. 全ての候補アルゴリズムの計測が完了すると, それらの所要時間の記録から最も高速なアルゴリズムを選出し, 次の Monitoring フェーズで使用するアルゴリズムとする. なお, アルゴリズムの選出に用いる各アルゴリズムの所要時間としては, 全ランクの所要時間の平均値を用いる.

Monitoring フェーズ :

Learning フェーズで選出されたアルゴリズムを用いて集団通信を行う. このフェーズ

は Learning フェーズが終了してアルゴリズムが選択された後, 各集団呼び出しに対して実行する. 実際の計算環境では実行中に状況が大きく変化することも考えられるため, 定期的に集団通信の所要時間と Learning フェーズで得られた所要時間を比較する. もし, 計測した時間が Learning フェーズ時の所要時間から大きく変動した場合, 他のアルゴリズムの方が高速となった可能性があるので, 再度 Learning フェーズに入り, アルゴリズムを選択する.

5. 実　　験

5.1 実験の概要

前述の RICC において, 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, ランク数とメッセージサイズを変えながら RICC のメタジョブスケジューラに投入した.

RICC において用いられているメタスケジューラは, 出来るだけ空いている計算ノードが少なくなるようにジョブを割り当てて, システムにおけるジョブの充填率を上げることにより, 総合的な処理速度の向上を図っている. そのため, 同じランク数のジョブであっても, 割り当てられるノードの位置に規則性が無い. その結果, 通信遅延の変動や通信衝突の影響により, 通信性能が不安定となる. 従来の MPI ライブラリの集団通信関数で用いられている静的なアルゴリズム選択手法では, このような環境では最適なアルゴリズム選択ができない. 一方, 本研究の提案手法や STAR-MPI では, 実行時に各アルゴリズムを試すため, このような環境でも状況に応じたアルゴリズム選択を行うことができる.

5.2 実 験 結 果

図 4〜図 7 に, メッセージサイズが 16KB のときの, 1 ランク ×32 ノード, 2 ランク ×32 ノード, 4 ランク ×32 ノード, 8 ランク ×32 ノードのそれぞれのランク数での Alltoall 通信の平均所要時間を, 10 回実行したジョブ毎に示す. X 軸はジョブの番号, Y 軸は所要時間である. それぞれの図で示されている性能は, 本稿で提案したアルゴリズムを絞り込んだ動的アルゴリズム選択技術による Alltoall(DYN GROUP) と, アルゴリズムを絞り込まない動的アルゴリズム選択技術による Alltoall(DYN　NOGROUP), さらに, Bruck(Bruck), Pairwise(Pair), Pairwise with Light-Barrier(PairLB), Pairwise with MPI-Barrier(PairMB), Pairwise with One-Barrier(PairOB), Ring(Ring), Ring with Light-Barrier(RingLB), Ring with MPI-Barrier(RingMB), Ring with One-Barrier(RingOB) の各アルゴリズムのものである.

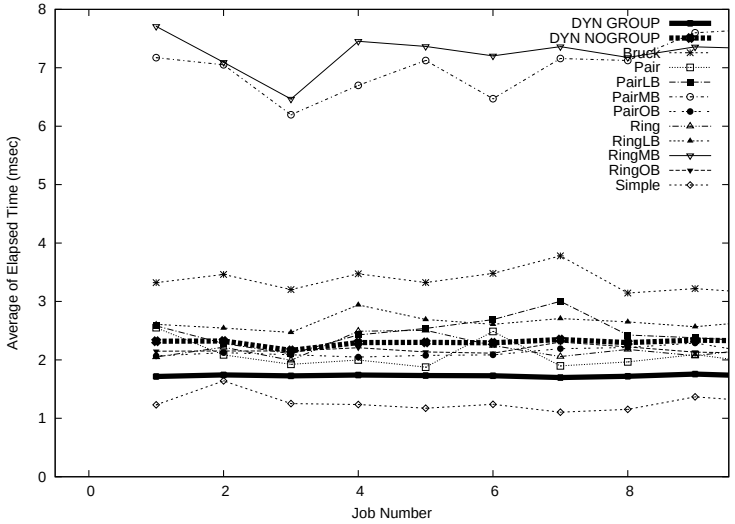

図 4　ジョブ毎の集団通信の所要時間 (16KB, 1 ランク ×32 ノード)

DYN GROUP, DYN NOGROUP とも, 多くの場合で最速に近い性能を達成している. これは, 動的アルゴリズム選択によってほぼ最適に近いアルゴリズムが選択されたことを示している. ここで, 最速のアルゴリズムに対する差分が, 動的最適化に伴うオーバヘッドである. DYN NOGROUP と比較すると, ほとんどの場合で DYN GROUP の方がこの差分が小さいことから, 提案手法が低オーバヘッドで精度の高いアルゴリズム選択を行えることが確認できた.

このオーバヘッドは, 主に Learning フェーズにおいて最速でないアルゴリズムを試すことによる時間である. そこで, アルゴリズムを絞り込むことによる効果を検証するため, Learning フェーズの所要時間の比率を図 8〜図 11 に示す. ほぼすべての場合で, アルゴリズムを絞り込むことによって Learning フェーズの所要時間を短縮できている.

6. 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[5)–9)]. さらに近年, これら

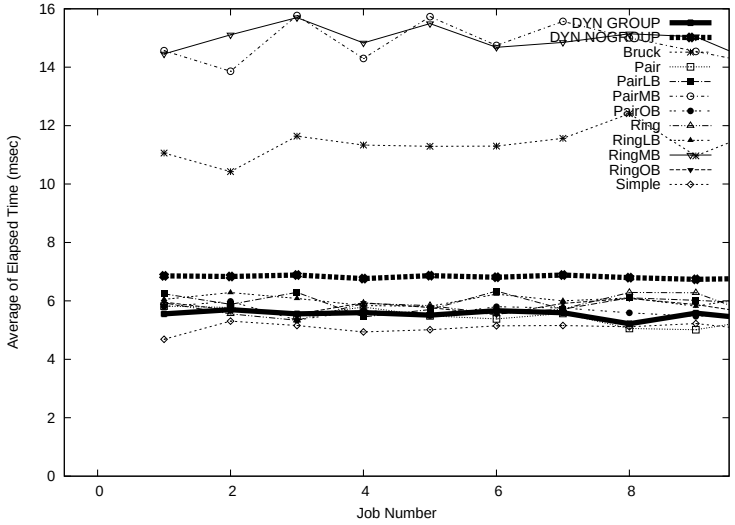

図 5 ジョブ毎の集団通信の所要時間 (16KB, 2 ランク ×32 ノード)

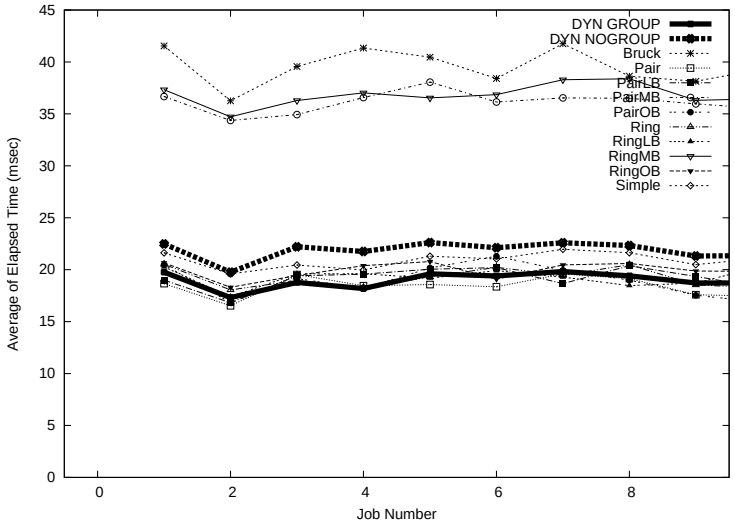

図 6 ジョブ毎の集団通信の所要時間 (16KB, 4 ランク ×32 ノード)

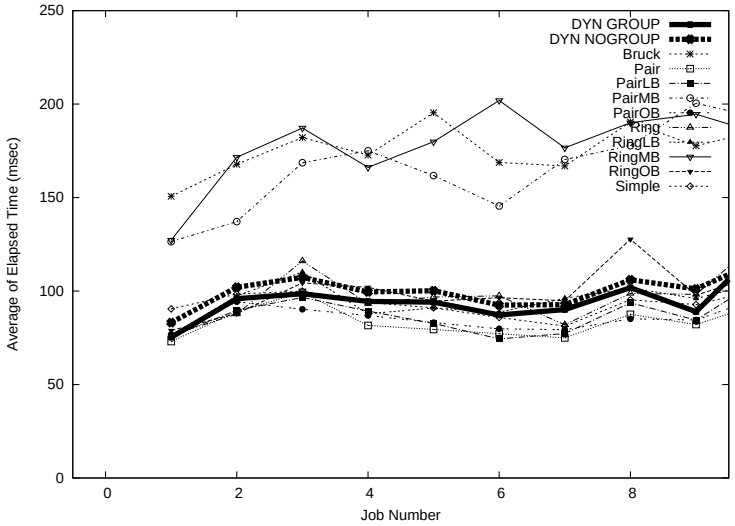

図 7 ジョブ毎の集団通信の所要時間 (16KB, 8 ランク ×32 ノード)

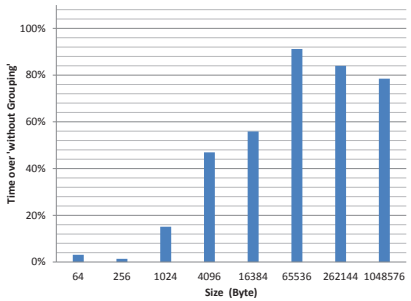

図 8 アルゴリズム絞り込みを行わない場合に対する Learning フェーズの所要時間の比率 (1 ランク ×32 ノード)

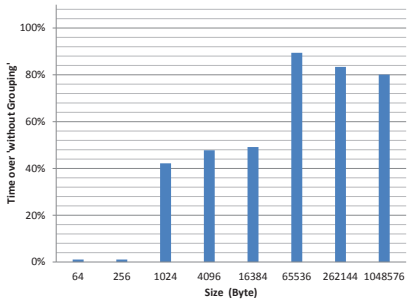

図 9 アルゴリズム絞り込みを行わない場合に対する Learning フェーズの所要時間の比率 (2 ランク ×32 ノード)

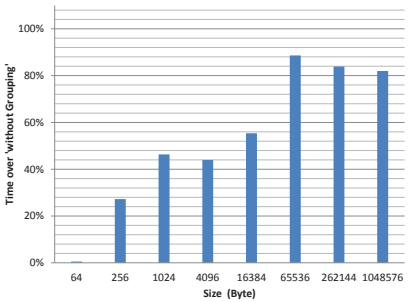

図 10 アルゴリズム絞り込みを行わない場合に対する Learning フェーズの所要時間の比率 (4 ランク ×32 ノード)

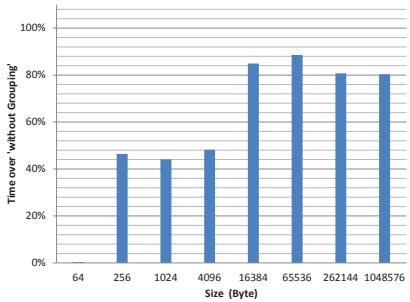

図 11 アルゴリズム絞り込みを行わない場合に対する Learning フェーズの所要時間の比率 (8 ランク ×32 ノード)

のアルゴリズムから最適なものを選択する技術に関する研究が行われている．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[10]．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[11]．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, 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[12]．しかし, この性能予測にはランクの配置による影響が考慮されておらず, 精度に問題がある．また, 各アルゴリズムの計測は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．

なお, 以前より著者らは, 性能予測モデルと実行時の試行を組み合わせた手法を提案していた[13]．以前提案した手法では, 性能予測モデルに用いる遅延時間と帯域幅を実行時に計測して取得していた．しかしこの手法では, 計測のオーバヘッドを抑えるために十分な情報を取得できず, ランク配置による影響を考慮した性能値の算出が困難であった．今回提案する方法は, 基本的な遅延時間と帯域幅の計測は予め時間をかけて行い, 実行時にはランク配置の影響の計算のみを行うことでオーバヘッドの低減とアルゴリズム選択の高精度化の両立を図っている．

7. まとめと今後の課題

通信ライブラリの動的最適化技術の一つとして, 集団通信アルゴリズムの動的選択技術を提案した．提案手法では, トポロジ情報やランク配置情報を用いた各アルゴリズムの性能予測により候補を絞り込んだ上で, 集団通信の呼び出し毎に一つずつアルゴリズムを試すことにより最適なアルゴリズムを選択する．この手法を用いて RICC の Fat Tree トポロジに向けた Alltoall 通信関数を実装し, 実験で有効性を検証した．実験の結果, 提案手法が低オーバヘッドで精度の高いアルゴリズム選択を行えることを確認した．

今後は Alltoall 以外の集団通信への適用や, Fat Tree 以外のトポロジへの適用を進め, 動的最適化技術によるスケーラブルな通信ライブラリの構築を目指す．

8. 謝　　辞

本研究の計算結果は, RIKEN Integrated Cluster of Clusters (RICC) を利用して得られた．

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) 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).
3) 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).
4) 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).
5) Mitra, P., Payne, D., Shuler, L., van de Geijn, R. and Watts, J.: Fast Collec-

tive Communication Libraries, *International Supercomputing Users Group Meeting*, (1995).
6) 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).
7) 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).
8) Rabenseifner. R.: Optimization of Collective Reduction Operations, *International Converence on Computational Science*, LNCS3036, pp.1–9 (2004).
9) 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).
10) 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).
11) 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).
12) Nishtala, R.: Automatically Tuning Collective Communication for One-Sided Programming Models, PhDThesis, UC Berkeley, Berkeley (2009).
13) Nanri, T. and Kurokawa, M.: Effect of Dynamic Algorithm Selection of Alltoall Communication on Environments with Unstable Network Speed, *International Conference on High Performance Computing and Simulation (HPCS), 2011*, pp. 693–698 (2011).