Vol.2016-HPC-153 No.5
2016/3/1

非同期通信を使用した動的な集団通信アルゴリズム選択技術の実装と評価

杉山　裕宣[†1]　南里　豪志[†1,†2]

概要：集団通信の性能向上のためには，適切な集団通信アルゴリズムの選択が重要である．しかし，計算機の大規模化やネットワークの複雑化に伴い，従来の静的な手法によるアルゴリズム選択には限界が来ている．また，通信衝突や負荷バランスの変化等の，実行時の状況が通信性能に影響を与えるため，集団通信実行時の状況に応じて動的にアルゴリズムを選択する必要がある．従来の動的アルゴリズム選択技術はブロッキング集団通信のみを対象としており，MPI3.0 以降に追加された非ブロッキング集団通信インタフェースや遠隔アトミック操作，MPI Forum で現在議論中の永続型集団通信インタフェース等の新しい技術には対応していない．そこで本稿では，永続型非ブロッキング集団通信に対応し，遠隔アトミックを使用した効率の良い動的アルゴリズム選択技術の実装と評価を行う．性能評価の結果，提案手法により，適切なアルゴリズムの選択を低オーバーヘッドで実現可能である事を確認した．

1. 背景

並列計算機の大規模化に伴い，集団通信の性能が重要になっている．集団通信の実装には，複数のアルゴリズムが提案されており，実行される状況によりアルゴリズム間の性能の優劣が変動する．また，最も性能の良いアルゴリズムと性能の劣るアルゴリズムとの間で実行時間に 10 倍以上の差が生じる場合もあるため，適切なアルゴリズムの選択が集団通信の性能向上のために重要である．

従来の MPI ライブラリでは，一般的に，使用するシステムにおけるベンチマーク結果を基にアルゴリズム切り替えの閾値を設定し，静的にアルゴリズムを決定していた．しかし，計算機の大規模化やネットワークトポロジの複雑化に伴い，人力であらゆる状況におけるベンチマークを取る事が現実的でなくなってきている．さらに，複数のプロセスで通信路を共有するネットワークトポロジにおける通信衝突や負荷バランスの変化等，実行時の状況が通信性能に影響を与えるため，集団通信実行時の状況に応じて動的にアルゴリズムを選択する必要がある．

そこで，集団通信実行時の状況に応じて動的に最適なアルゴリズムを選択するための研究が行われている．例えば STAR-MPI[1] では，プログラム中での集団通信呼び出し毎にその集団通信で使用可能な全アルゴリズムを一つずつ実測し，実測結果から最速のアルゴリズムを選択している．一方，MPI では，MPI3.0 以降の改良で，非ブロッキング集団通信インタフェースの採用や片側通信における遠隔アトミック操作の追加，永続型集団通信インタフェースの提案がされている．しかし，従来の動的アルゴリズム選択技術はこれらに対応していない．そこで本稿では，永続型非ブロッキング集団通信に対応し，遠隔アトミック操作を使用した効率の良い動的アルゴリズム選択技術の実装と性能評価を行う．

†1 九州大学
Kyushu Uniersity
†2 独立行政法人科学技術振興機構　戦略的創造研究推進事業
Japan Science and Technology Agency(JST), Core Research for Evolutional Science and Technology(CREST)

2. 既存の集団通信アルゴリズム動的選択技術

Faraj ら [1] は，実測値を基にした動的アルゴリズム選択技術である，STAR-MPI を提案している．STAR-MPI は，最適アルゴリズムを選択する Learning Phase と，選択したアルゴリズムの性能を監視し，変動を検知するとアルゴリズムを選び直す Monitoring Phase から構成される．Learning Phase では，プログラム中で集団通信が呼ばれる度に，その集団通信で使用可能なアルゴリズムの中から一つを実行し，性能を計測する．全アルゴリズムについて一定回数の計測を終了後，STAR-MPI は全プロセスの計測結果の総和を取り，その結果から最速のアルゴリズムを選択する．

このようにして選ばれたアルゴリズムが，その後の状況の変化により，最適でなくなる可能性がある．そこで，Monitoring Phase において，その集団通信が一定回数呼

び出される毎に，Allreduce 通信を用いて全プロセスの所要時間を導出し，Learning Phase 時の計測値と比較する事で，アルゴリズムの性能変動を監視する．その際，予め設定した閾値超える変動を検知すると，STAR-MPI は，使用するアルゴリズムを，Learning Phase で 2 番だったアルゴリズムに変更する．

また，集団通信は同一引数であっても，プログラム中の呼ばれる場所によって負荷バランスの影響等から最適アルゴリズムが替わる場合がある．そのため，STAR-MPI では，call site と呼ばれる引数を追加する事で，同一の集団通信であっても呼び出し場所毎にアルゴリズム選択を行えるようにしている．

STAR-MPI は，Learning Phase で全アルゴリズムを一定回数実行するため，他よりも著しく性能の劣るアルゴリズムがある場合に，そのアルゴリズムを実行する事によるオーバーヘッドが生じる．そこで南里ら [3] は，予め性能予測により遅いアルゴリズムを候補から除外する事で，Learning Phase で実行するアルゴリズムの絞り込みを行っている．これにより，Learning Phase のオーバーヘッドを低減している．

これらはブロッキング集団通信のみを対象とした技術であり，さらに，MPI_Allreduce による全プロセスの同期を定期的に行う事によるオーバーヘッドが問題となる．一方，最新の MPI3.0 では，通信の開始と完了を分離した非ブロッキング集団通信インタフェースや，片側通信における遠隔アトミック操作が追加されている．また，2015 年 6 月の MPI Forum[2] では，集団通信全体の内の準備処理を独立した関数として実行可能にする永続型集団通信インタフェースが提案され，採用に向けた議論が進められている．従来のブロッキング集団通信のみを対象としたアルゴリズム選択技術は，MPI3.0 以降に登場したこれらの仕様には対応していない．そこで本研究では，永続型非ブロッキング集団通信に対応した，遠隔アトミック操作による効率の良い動的アルゴリズム選択技術の実装を行う．

3. MPI3.0 以降の改良によるアルゴリズム選択への影響

3.1 非ブロッキング集団通信インタフェースの採用

MPI では，通信関数呼び出し後，送受信の完了を待つ事なく，その通信の結果に依存しない別の処理に移る事を可能とする非ブロッキング通信が定義されている．MPI3.0 では，新たに非ブロッキング集団通信が追加された．非ブロッキング集団通信では，その集団通信の結果に依存しない計算を通信と並行して行う事で，通信の効率化が期待される．

非ブロッキング集団通信の実装では，計算と並行して集団通信アルゴリズムを進行させる推進機構の実装手段が，性能や利便性に大きく影響する．実装方法の一つとして，通信開始関数と通信完了待ち関数に集団通信アルゴリズムの処理を切り分ける実装方法がある．集団通信アルゴリズムは一般に，一対一通信の組み合わせにより構成されており，例えば図 1 に示す Ring アルゴリズムによる Allgather 通信では，プロセス数を P とすると $(P-1)$ 回のステップに通信を分け，各ステップで前のプロセスからデータを受け取り，そのデータを次のプロセスに渡していく．そのため，各ステップで他のプロセスからのデータの到着を待つ必要がある．このアルゴリズムによる非ブロッキング集団通信では，集団通信開始関数で 1 ステップ目の通信を発行し，通信完了待ち関数で残りの全ステップの通信を発行する事で，1 ステップ目の通信を計算と並行して行う事が可能となる．本稿では，この実装方法を想定し，動的アルゴリズム選択技術の開発を行う．

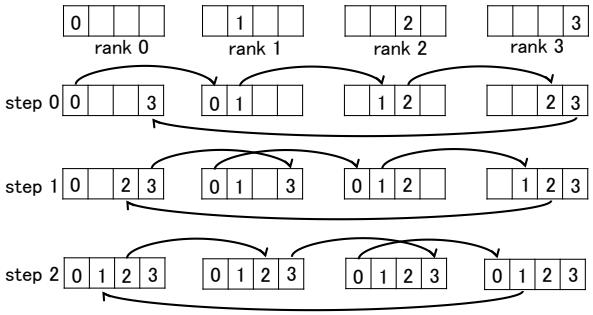

図 1　Ring アルゴリズムによる Allgather 通信

通信開始関数と通信完了待ち関数に集団通信の処理を切り分ける実装では，全通信処理に占めるオーバーラップ可能な通信処理の割合がアルゴリズムによって異なるため，通信と並行して実行させる計算の量によってアルゴリズムの優劣が異なる場合がある．例えば，アルゴリズム A とアルゴリズム B の二つを選択可能な集団通信があるとする．ここで，アルゴリズム A の実行時間は 10μsec であり，全ての通信を通信開始関数で発行可能とし，アルゴリズム B の実行時間は 6μsec であり，その内の 2μsec を占める通信を通信開始関数で発行可能とする．この時，計算時間が 1μsec の場合，図 2 に示す通り，アルゴリズム A とアルゴリズム B の通信開始から完了までの所要時間はそれぞれ 10μsec と 6μsec となり，最適アルゴリズムは B となる．一方で，計算時間が 7μsec の場合，図 3 に示す通り，アルゴリズム A とアルゴリズム B の通信開始から完了までの所要時間はそれぞれ 10μsec と 11μsec で，最適アルゴリズムは A となる．このように，非ブロッキング集団通信では，通信の所要時間だけでなく計算量がアルゴリズムの優劣に影響を与える．2 節で説明した従来のアルゴリズム選択技術では，この特性を考慮したアルゴリズムの絞り込みを行えない．

なお，非ブロッキング集団通信の実装手段としては，他に，計算中に適切な頻度で MPI_Test 関数を呼び出し，そ

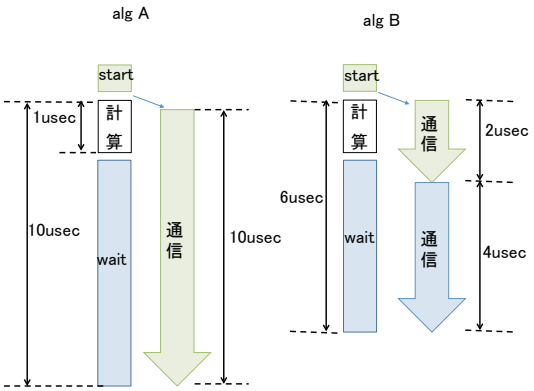

図 2　通信と並行して実行する計算量が少ない場合のオーバーラップの例

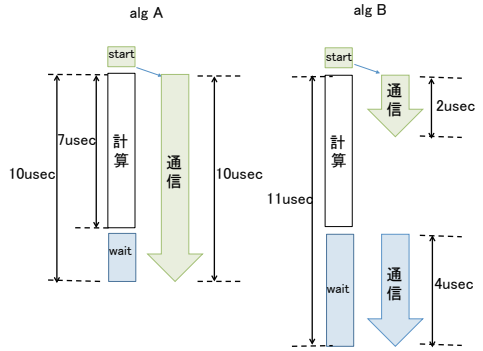

図 3　通信と並行して実行する計算量が多い場合のオーバーラップの例

の中でアルゴリズムを進行させる方法や，プログレススレッドによりアルゴリズムを進行させる方法等が考えられる．これらの実装手段に対応した動的アルゴリズム選択については，今後の検討課題とする．

3.2　片側通信における遠隔アトミック操作の追加

MPI3.0 から，片側通信命令として遠隔アトミック操作が追加された．これにより，遠隔メモリに対して排他的な値の変更が可能となる..

STAR-MPI[1] では，アルゴリズムの性能変動を検知するために，Allreduce 通信を一定回数毎に行っていた．しかし，この方法では，性能変動の有無に関わらず，一定回数毎に全プロセスでの同期を発生させ，一番遅いプロセスに足並みを揃える事になる．

一方，アトミック操作を使用する事で，性能変動の通知のための同期を必要最小限にする事が出来る．集団通信に参加するプロセスの中から任意のプロセスをマスターとし，マスタープロセスが性能変動の一定回数の性能測定終了後，各プロセスはモニタリング結果と Learning Phase での自分の測定結果を比較し，性能の変動を検知すると，遠隔アトミック操作によりマスタープロセスの持つカウンタの値をインクリメントする．マスタープロセスは，カウンタの値を確認する事で性能変動を検知したプロセスの数を知る事ができ，この値によってアルゴリズム切り替えの判断を行う．遠隔アトミック操作は，片側通信であるため，各プロセスとマスタープロセスの間で同期が発生しない．そのため，同期を必要最小限に抑えて性能変動の通知を行う事が可能となる．

同様の処理は，例えば，遠隔アトミック操作の代わりに MPI_Isend 関数を使用し，マスタープロセスはカウンタの値を確認する代わりに，任意の通信の到着を確認する MPI_Iprobe 関数仕様する事で可能である．しかし，この方法は，各プロセスからの通知の度にマスタープロセスで受信処理が必要となる．これに対し提案手法では，カウンタの値を定期的に確認するだけで変動を検知出来る．そのため，本稿では遠隔アトミック操作を使用した性能変動検知を実装する．

3.3　永続型集団通信インタフェースの提案

並列プログラムでは，ループ中で同一の引数を持った通信を繰り返し実行する事が頻繁にある．そこで MPI では，永続型通信が定義されている．永続型通信とは，通信全体の内，通信に必要な準備処理を独立した関数として実行可能とするものである．この準備関数は，準備処理が済んだ通信関数の情報をリクエスト構造体として返す．プログラマは，このリクエスト構造体を繰り返し使用する事が出来る．これにより，同一の通信を繰り返す場合に効率よく実行する事が可能となる．

現在の最新の MPI 規格である MPI3.1 では，一対一の永続型通信のみが定義されている．このインタフェースを集団通信に適用した永続型集団通信インタフェースが，2015 年 6 月の MPI Forum[2] で提案され，採用に向けた議論が進められている．永続型集団通信インタフェースでは，集団通信準備関数と集団通信開始関数が新たに追加され，その完了待ちを既存の MPI_Wait 関数等で行う事が出来る．

2 節で説明した通り，STAR-MPI では，MPI の標準に対して call site と呼ばれる引数を追加する事で，集団通信の呼び出し場所を識別しており，使用出来る状況が限定される．一方，永続型集団通信では，リクエスト構造体を，集団通信の呼び出し場所の識別に利用する事で，呼び出し場所に応じたアルゴリズム選択が可能となる．

4.　永続型非ブロッキング集団通信に対応した動的アルゴリズム選択技術の実装

4.1　提案する動的集団通信アルゴリズム選択技術の概要

本研究で提案する動的アルゴリズム選択では，STAR-MPI[1] と同様に，Learning Phase と Monitoring Phase から構成され，また，Learning Phase では，[3] と同様のアルゴリズム絞り込みを行う．

本提案手法は永続型集団通信を対象とするため，Learning

Phase の一部は集団通信準備関数で行われる．プログラム中で集団通信準備関数が呼ばれると，その集団通信で使用可能なアルゴリズムの絞り込みを行う．さらに，絞り込みで残った候補アルゴリズムの準備と，アルゴリズム選択のための準備を行い，そこで得られた情報をリクエスト構造体に書込む．

一方，集団通信開始関数と集団通信完了待ち関数では，Learning Phase の間，STAR-MPI[1] と同様に，各アルゴリズムでの性能を計測し，その結果を基に最速アルゴリズムを決定する．

その後の Monitoring Phase では，Learning Phase で決定したアルゴリズムについて，遠隔アトミック操作を使用した性能変動の検知を行う．一定回数の性能計測後，性能変動を検知したプロセスは，遠隔アトミック操作によりマスタープロセスのカウンタをインクリメントする．マスタープロセスはカウンタの値を基に Learning Phase に戻るか否かを判断し，それを各プロセスに通知する．

4.2 性能予測を使用した候補アルゴリズムの絞り込み

ブロッキング集団通信を対象とした絞り込みでは，アルゴリズムの通信性能のみを考慮すれば良かった．しかし，本稿で想定している非ブロッキング集団通信の実装では，3.1 節で述べた通り，通信と並行して実行させる計算の量によって最適アルゴリズムが異なる場合があり，アルゴリズムの絞り込みに大きな影響を与える．通信と並行して実行させる計算の量が少なく，集団通信開始関数で発行した分の通信が計算によって完全に隠蔽出来ない場合，アルゴリズム間の優劣は，アルゴリズムの通信性能に左右される．しかし，計算量が多くなると，通信時間が長くてもオーバーラップにより実行時間が短くなる場合がある．この場合は，完了待ち関数で発行される通信時間がアルゴリズム間の優劣に影響を与える．そこで，集団通信開始関数で発行した分の通信が計算によって完全に隠蔽出来た場合を対象としたアルゴリズムの優劣評価が必要となる．

一方で，使用する通信プロトコルも，アルゴリズムの絞り込みに影響を与える．多くの MPI の実装ではメッセージサイズによって通信プロトコルを切り替えており，一般的に，メッセージサイズが小さい範囲では Eager プロトコルが，メッセージサイズが大きい範囲では Rendezvous プロトコルが使用される．Eager プロトコルは受信バッファを介した通信方式で，送信側プロセスは受信側プロセスの受信バッファに空きがある限り，いつでもメッセージを送信する事が出来る．このプロトコルの場合，集団通信開始関数で通信命令を発行後，すぐに送信が開始されるため，通信と計算を並行して行う事が出来る．一方，Rendezvous プロトコルは，送信関数と受信関数が呼び出されるまで待ってメッセージの送信が開始される通信方式である．集団通信開始関数で通信命令を発行した段階では，送信側と受信側がどちらも相手側の命令発行を確認出来ず，集団通信完了待ち関数で相手側の命令発行を確認して初めてメッセージの送受信を開始するため，通信と計算を並行して行う事は出来ない．即ち，Rendezvous プロトコルでは，アルゴリズム間の優劣はアルゴリズムの通信性能によって決定される．

以上の事を考慮し，本研究で実装するアルゴリズム選択では，メッセージサイズに応じて絞り込みの方針を変える．Rendezvous プロトコルが使用されるメッセージサイズの範囲では，アルゴリズムの絞り込みにおいてアルゴリズムの通信時間を予測し，予測時間が最速アルゴリズムのものに対し 2 倍よりも遅いアルゴリズムを実行候補アルゴリズムから除外する．一方で，Eager プロトコルが使用されるメッセージサイズの範囲では計算量によってアルゴリズムの優劣が異なるため，アルゴリズムの通信時間だけでなく，集団通信開始関数で発行した通信を除く全処理の所要時間についても予測を行う．その上で，どちらか一方でも最速アルゴリズムの予測時間の 2 倍以内に収まっていれば実行候補アルゴリズムに残す．これにより，アルゴリズムの予測通信時間だけでは遅いと判断されて除外されるアルゴリズムの中から，オーバーラップにより実行時間が速くなるアルゴリズムを候補に残す事が出来る．

4.3 遠隔アトミック操作を使用した性能変動検知技術

STAR-MPI では，Monitoring Phase において，性能変動を検知するために Allreduce 通信を一定回数毎に実行しており，その度に全プロセスで同期が取られる．そこで本研究では，同期を必要最小限に抑えるために，遠隔アトミック操作を使用した性能変動検知技術を提案する．本研究で提案する性能モニタリングでは，遠隔アトミック操作の準備，性能変動の通知，Learning Phase へ戻る事の通知の三つの実装を必要とする．

この内，遠隔アトミック操作の準備は，集団通信準備関数で行う．MPI の片側通信は，Active target 通信と Passive target 通信に分かれており，使用目的によりどちらかを選択する．Active target 通信では，メッセージの転送元と転送先の両プロセスが明示的に通信に参加するのに対し，Passive target 通信では，片側のプロセスの通信命令のみでメッセージの転送を完了させる事が出来る．今回の実装では，マスタープロセスのもつカウンタに対して，非同期的に全プロセスがアクセスする必要があるため，Passive target 通信を使用する．そこで，準備関数内では，カウンタを用意し MPI_Win_create 関数によりそのカウンタを他のプロセスからアクセス可能にし，さらに MPI_Win_lock_all 関数により Passive target 通信を可能にする．以降，全てのプロセスはマスタープロセスのカウンタに対して自由にアクセス可能となる．

Monitoring Phase では，集団通信開始関数と集団通信完

了待ち関数が一定回数呼び出される度に，Learning Phase 時の計測値との比較を行う．この呼び出し回数を ITER とすると，ITER 回目の集団通信完了後に性能変動を検知したプロセスは，MPI_Fetch_and_op 関数による遠隔アトミック操作でマスタープロセスのカウンタをインクリメントする．MPI_Fetch_and_op 関数実行後は，通信の完了を待つ事なく復帰し，集団通信完了待ち関数を抜ける．一方，マスタープロセスは，ITER+1 回目の集団通信完了時にカウンタ値を確認する．カウンタ値が予め設定した閾値を超えている場合，特別なタグを付けた MPI_Isend 関数で，Learning Phase に戻る事を他の全プロセスに対し通知する．MPI_Isend 関数を実行した場合，ITER+2 回目の集団通信開始前に，全ての MPI_Isend 関数の完了を待つ．マスタープロセス以外のプロセスは，ITER+2 回目の集団通信完了後，MPI_Iprobe 関数を使用し指定したタグに対応するメッセージが届いているかを確認する．メッセージが届いている場合は次回の集団通信から Learning Phase に戻り，メッセージが届いていない場合は次回の集団通信から再び ITER 回の性能計測を行い，上記の操作を繰り返す．

5. 実験

5.1 実験環境

本研究で提案するアルゴリズム選択の性能を評価するため，Allgather 通信を対象として，Bruck, Ring, Linear の 3 つのアルゴリズムを実装した．プロセス数を P とすると，Bruck アルゴリズムは，各ステップで前のステップまでに受信したメッセージをまとめて送信する事で，ステップ数を $\lceil log_2 P \rceil$ に抑える事が出来る．しかし，ステップ毎にメッセージサイズが冪乗で増加していく他，全ステップ終了後にメッセージのソーティングのためのメモリコピーが必要である．Ring アルゴリズムは，プロセスをリング状に並べた際の前のプロセスからメッセージを受信し，次のプロセスに対しそのメッセージを送信して行く事で，$(P-1)$ ステップで通信を完了する．Linear アルゴリズムは，各プロセスがそれぞれ他の全プロセスに対し自分の持つメッセージを送信し，他の全プロセスからメッセージを受信する．

表 1 実験環境

System	FUJITSU PRIMERGY CX400
相互結合網	InfiniBand FDRx1(片方向 6.78GB/s)
総ノード数	1476
メモリバンド幅	102.4GB/s
OS	Red Hat Linux Enterprise
Compiler, MPI	OpenMPI-2.0.0rc1

実験では，通信と計算のオーバーラップにより最適アルゴリズムが変化する事を確認するため，計算量を変えながら，アルゴリズムの実行時間を計測した．また，その際に提案手法が最適なアルゴリズムを選択できているか確認した．実験環境は，九州大学の FUJITSU PRIMERGY CX400(表 1) を使用している．全ての実験は，ノード内プロセスを 1 として行っており，性能の計測には OSU Micro-Benchmarks 5.1[7] を使用している．このベンチマークプログラムでは，ループ中で集団通信開始関数を実行後，任意のサイズの行列積計算を行い，集団通信完了待ち関数を行う．

5.2 オーバーラップさせる計算量による各アルゴリズムの性能とオーバーラップ率との関係

図 4 から図 7 はそれぞれ，16 プロセスと 64 プロセスにおける集団通信開始から完了までの所要時間を，各アルゴリズムと提案手法適用時で，行列サイズを変えて比較したものである．各グラフはそれぞれ，横軸がメッセージサイズ，縦軸が所要時間である．また，グラフ中の alg0 から alg2 は，それぞれ Bruck，Ring，Linear の各アルゴリズムを表し，OPT_AP は提案手法を表している．

図 4 より，16 プロセスで行列サイズが 25×25 の場合，メッセージサイズが 128KB 付近までは Linear が最速で，それ以降は Ring が最速になっている事が分かる．Bruck は，メッセージサイズが 2KB 付近までは最速アルゴリズムに近い性能で，メッセージサイズが大きくなると性能が落ちている．これは，Bruck はステップ毎に転送されるメッセージサイズが 2 倍ずつ増えるためである．一方で，図 5 から分かる通り，16 プロセスにおいて行列サイズを 100×100 にすると，メッセージサイズが小さい範囲で Bruck と Linear の性能差が行列サイズ 25×25 の場合よりも小さくなっている．これは，Bruck と Linear のオーバーラップ率の影響であると考えられる．

同様の傾向が，6 と図 7 に示す 64 プロセスの場合にも見られる．図 6 では，メッセージサイズが 1KB までは Bruck が，1KB から 64KB までは Linear が，64KB 以降は Ring がそれぞれ最速になっている事が分かる．一方，図 7 から分かる通り，行列サイズが 100×100 になると，1KB までの範囲で Bruck と Linear の性能差が小さくなっており，また，Bruck と Ring の優劣が入れ替わるメッセージサイズが 32KB になっている．

Bruck は，ステップ数を $\lceil log_2 P \rceil$ に抑える事が出来る．しかし，ステップ毎にメッセージサイズが増加するため，メッセージサイズが小さい範囲で，プロセス数が大きくなるほど他のアルゴリズムに比べて優位となり，図 6 はその傾向を顕著に表している．その一方で，オーバーラップ率に着目すると，Bruck と Ring は 1 ステップ目の通信のみがオーバーラップ可能なのに対し，Linear は全ての通信をオーバーラップ可能である．また，Bruck は 2 ステップ以降の通信量が大きくなるため，他のアルゴリズムに比べてオーバーラップ出来る通信の量が少ない．そのため，図 6 と図 7 から分かる通り，計算量を増やした際に，Bruck は

他の二つのアルゴリズムに比べて所要時間の増加量が多く，Linear は他の二つのアルゴリズムに比べて所要時間の増加量が少ない．

このように，今回の非ブロッキング集団通信の実装では，アルゴリズムによって通信全体に占めるオーバーラップ可能な通信時間が異なるため，通信と並行して実行させる計算量によって，アルゴリズム間の相対的な優劣関係が変わる．これに対し，各グラフで OPT_AP は，ほぼ全てのメッセージサイズで最適アルゴリズムに近い性能を示している事から，最適なアルゴリズム選択が行えていると確認出来る．

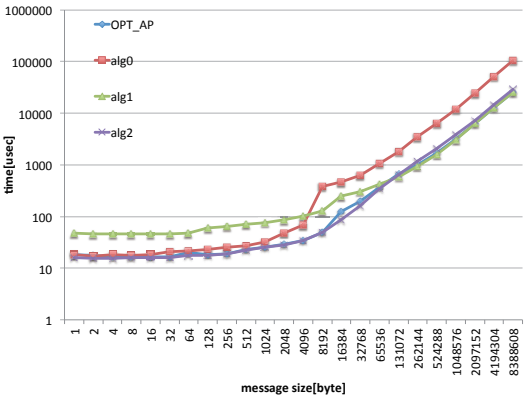

図 4　16 プロセスにおける行列サイズ 25×25 の場合の所要時間の比較

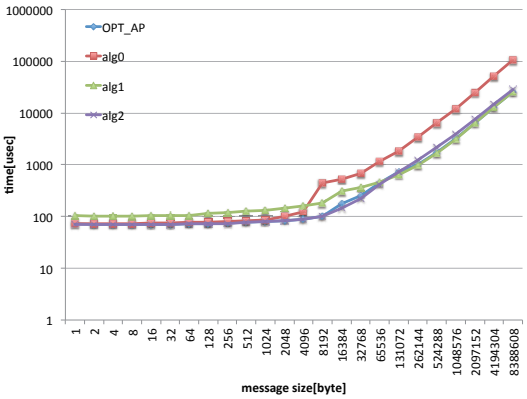

図 5　16 プロセスにおける行列サイズ 100×100 の場合の所要時間の比較

5.3　提案手法によるアルゴリズム選択のオーバーヘッド

図 8 に，64 プロセスにおける行列サイズ 100×100 の場合の提案手法によるオーバーヘッドの割合を示す．これは，各メッセージサイズについて，提案手法によるアルゴリズム選択をした場合の所要時間から，そのメッセージサイズにおける最適アルゴリズムの所要時間を引いた値を，最適アルゴリズムの所要時間で割った値である．グラフの横軸はメッセージサイズである．図 8 より，メッセージサイズが 16KB と 64KB の場合を除いて，オーバーヘッドの割合は最適アルゴリズムの所要時間に対して 10%程度に収まっている事が分かる．他のプロセス数，行列サイズにおいても，同様に，いくつかの例外を除いて 10%以内のオーバーヘッドで動的アルゴリズム選択が行えている．

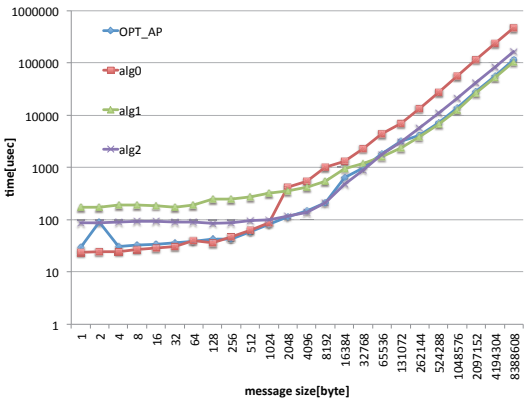

図 6　64 プロセスにおける行列サイズ 25×25 の場合の所要時間の比較

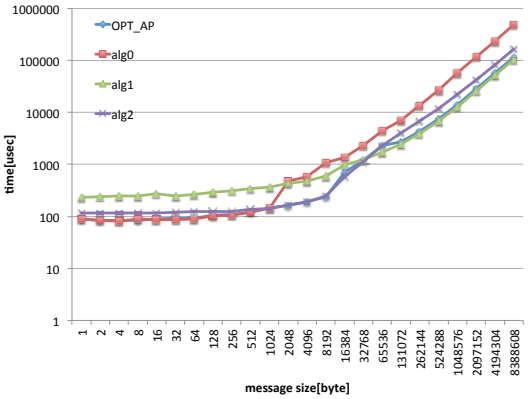

図 7　64 プロセスにおける行列サイズ 100×100 の場合の所要時間の比較

なお，いくつかのメッセージサイズにおいてオーバーヘッドの割合が負の値となっているのは，計測誤差が原因であると考えられる．また，メッセージサイズが 16KB を境にオーバーヘッドが大きくなっており，この原因の特定については今後の課題とする．

6.　まとめ

本稿では，MPI3.0 以降で追加された非ブロッキング集団通信インタフェース，及び片側通信におけるアトミック操作，さらに現在採用に向けて議論が進められている永続型集団通信インタフェースに対応した，動的アルゴリズム選択技術の実装と評価を行った．Allgather 通信を例として，三つの異なる特徴を持ったアルゴリズムを対象に，提案する動的アルゴリズム選択の性能評価を行った．その結果，提案手法により，少ないオーバーヘッドで適切なアルゴリズムの選択が出来ている事を確認出来た．

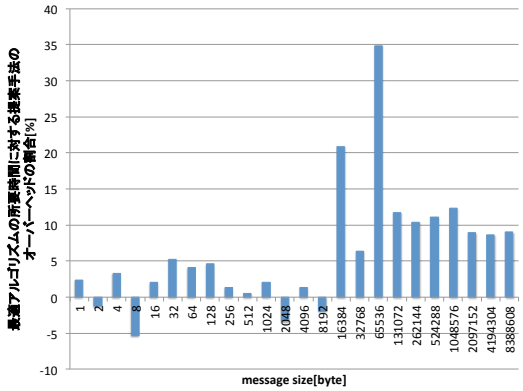

図 8　64 プロセスにおける行列サイズ 100×100 の場合の提案手法によるオーバーヘッド

今後の課題としては，非ブロッキング集団通信の他の実装への対応，提案手法のオーバーヘッドの評価，実行時に状況が変化する場合における提案手法の効果の検証，実アプリケーションに対する提案手法の効果の検証等が挙げられる．

参考文献

[1] A. Faraj, X. Yuan, and D. Lowenthal, “STAR-MPI: self tuned adaptive routines for MPI collective operations.” Proceedings of the 20th annual international conference on Supercomputing. ACM, 2006.
[2] MPI Forum, http://www.mpi-forum.org/
[3] T. Nanri, and M. Kurokawa, “Efficient Runtime Algorithm Selection of Collective Communication with Topology-Based Performance Models.”, Proceedings of the 2012 International Conference on Parallel and Distributed Processing Techniques and Applications, 2012.
[4] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian and T. von Eicken, “LogP: Towards a Realistic Model of Parallel Computation.”, Proc. Symposium on Principles and Practice of Parallel Programming (PPoPP), pp. 1-12, 1993.
[5] A. Alexandrov, M. F. Ionescu, K. E. Schauser and C. Scheiman, “LogGP: Incorporating Long Messages into the LogP Model - One Step Closer Towards a Realistic Model for Parallel Computation.”, Proc. Symposium on Parallel Algorithms and Architectures (SPAA), pp.95-105, 1995.
[6] T. Kielmann, E. B. Henri , and G. Sergei, “Bandwidth-efficient collective communication for clustered wide area systems.” Parallel and Distributed Processing Symposium, IPDPS 2000. Proceedings. 14th International. IEEE, 2000.
[7] http://mvapich.cse.ohio-state.edu/benchmarks/