Vol.2017-HPC-159 No.7
2017/4/17

動的最適化機構を持つ非ブロッキング集団通信関数の実装と評価

成林 晃[1]　南里 豪志[1,a)]　天野 浩文[1]

概要：本研究では、実行時の状況に応じて実装手段を選択する動的最適化機構を有する非ブロッキング集団通信ライブラリ NBC-SIA (Non-Blocking Collective that Selects Implementation Automatically) を開発し、初期評価を行った。このライブラリは、利用可能な実装手段を、集団通信の呼び出し毎に一つずつ試行して、最終的に最も高速だったものを選択する。実装手段としては、複数の集団通信アルゴリズムの他、プログレススレッド使用の有無、プログレススレッドへの CPU コアの割当方法も選択可能とし、これらの組み合わせの中から最適なものを選択する。また、プログレススレッドを使用する場合は、本研究で新たに開発した、排他制御のオーバヘッドを低減したアルゴリズム推進機構を用いる。初期評価の結果、NBC-SIA の現在の実装では、必ずしも最速のものが選択できないことが判明した。これは、全プロセスでの計測結果の集計方法における問題であり、今後改良を予定している。一方、NBC-SIA のプログレススレッドが、従来の MPI ライブラリでの実装に対して、十分低オーバヘッドで動作することを示した。また、実行時の状況によって最適な実装手段が変動することも確認し、本研究の必要性を確認した。

Implementation and Evaluation of Non-Blocking Collective Commnication with Runtime Optimization Mechanism

NARIBAYASHI Akira[1]　NANRI Takeshi[1,a)]　AMANO Hirofumi[1]

Abstract: This work has developed a non-blocking collective communication library NBC-SIA (Non-Blocking Collective that Selects Implementation Automatically). This library examines each choises of implementations for non-blocking collectives, one by one at each invocation, and chooses the best one. As for the selection of implementations, it provides choises such as use or not to use progress thread, how to attach CPU core to the progress thread, and algorithms of collectives. In addition to that, as for the progress thread, this work has implemented an efficient mechanism with lower overhead of mutual exclusion for handling task queues. According to the results of preliminal experiments, at this point, it has shown that the selection mechanism of NBC-SIA cannot choose the best one. It is caused by some problem in the way of collecting measured performances among processes. This problem will be fixed in the next version. On the other hand, the measured performance of the progress thread of NBC-SIA is better than the ones provided in other MPI library. Also, necessity of this kind of runtime optimization techniques has been shown by showing that the best implementation changes according to the runtime situation.

1. 背景

計算機の大規模化にともない、並列計算において集団通信に要する時間の削減が、アプリケーション全体のスケーラビリティ向上における重要な課題となっている。一方、通信時間を削減する手段の一つに、非ブロッキング通信関数を用いた通信と計算の重複実行がある。並列計算におけるプロセス間通信インタフェースの事実上の標準規格である Message Passing Interface (MPI) では、従来の一対一通信向け非ブロッキング通信関数に加え、2012 年 9 月に制定された MPI-3.0 規格において、非ブロッキング集団通信関数が採用された。そのため、近年、この関数を活用した集団通信時間の削減技術が注目されている。この非ブロッ

[1] 九州大学
Kyushu University
[a)] nanri.takeshi.995@m.kyushu-u.ac.jp

キング集団通信の実装手段において中核となるのが、集団通信アルゴリズムを内部で進行させる推進機構である。特に、プログレススレッドによる推進機構は、プログラム中に MPI_Test 関数のような、アルゴリズム推進のための関数を挿入しなくても高い通信隠蔽率が期待できる上、特殊なハードウェアを必要としないという特徴がある。しかし、現在の MPI ライブラリにおけるプログレススレッドの実装では、主にスレッド間の排他制御のためのオーバヘッド等により通信に要する時間が増大し、プログラム全体の性能向上への寄与が、限定的である。また、非ブロッキング集団通信の実装手段には、このプログレススレッドの利用の有無や、集団通信アルゴリズムの選択、プログレススレッドへの CPU コアの割り付け方、等の多様な選択肢があり、しかも最適な選択は実行中の状況によって変動する。これに対し、現在の MPI ライブラリの実装では、実行中に実装手段を変更する機能を有していないため、状況に応じた最適な選択が行えない。

そこで本研究では、排他制御を削減した効率的なプログレススレッドによる推進機構と、実行時の状況に応じた実装選択機構を有する非ブロッキング集団通信ライブラリ NBC-SIA (Non-Blocking Collective that Selects Implementation Automatically) を開発する。NBC-SIA には、集団通信アルゴリズムの進行管理に使用するタスクキューを、ロック操作が不要となるように構築することで排他制御のオーバヘッドを削減した、プログレススレッドによるアルゴリズム推進機構を用意する。また、実行時に、選択可能な実装手段を一つずつ試行し、その中から最速のものを選択する動的選択機構により、状況に応じた最適な実装選択を可能とする。NBC-SIA のプログラミングインタフェースには、集団通信の呼び出し位置ごとに独立した実装手段選択が行えるよう、永続型集団関数を用いる。さらに、プログラムの内容に応じてライブラリ内の動作を効率化できるよう、プログラムのヒント情報を受け付けるインタフェースも用意する。

本稿の主な寄与は、以下の通りである。

- 排他制御オーバヘッドを低減したタスクキューを用いたプログレススレッドによる非ブロッキング集団通信の実装と評価
- 実行時の状況に応じて非ブロッキング集団通信の実装手段を選択する動的選択機構の必要性の検討、および効果の検証
- 永続型通信インタフェースとヒント情報インタフェースによるプログラム情報を用いた通信ライブラリ効率化の可能性の検証

2. 既存の非ブロッキング集団通信実装

2.1 非ブロッキング集団通信インタフェース

MPI-3.0 規格において採用された非ブロッキング集団通信インタフェースは、従来の一対一通信における非ブロッキング通信と同様、通信の開始と完了待ちをそれぞれ別の関数とすることで、通信完了を待つ間に、その通信と並行して処理可能な計算や通信の実行を可能とするものである。通信開始関数としては、MPI_Iallreduce 関数や、MPI_Ialltoall 関数等、従来のブロッキング集団通信関数名に I を追加したものが用意されている。これらの関数は、完了を待つための情報を、MPI_Request 型のリクエスト変数に格納する。この変数は、一対一の非ブロッキング通信と同じ MPI_Wait 関数や MPI_Waitall 関数等に指定することで、完了を待つことが出来る。これらの関数を用いて通信の開始と完了待ちを指示し、さらにその通信と関係のない処理を開始と完了待ちの間に挿入することで、その通信の時間の一部を他の処理で隠蔽することが期待できる。

現在、MPICH [2]、MVAPICH2 [3]、Open MPI [4] といった主要なオープンソースの MPI ライブラリの他、Intel MPI Library [5] 等の非オープンソースの MPI ライブラリの多くで、この非ブロッキング集団通信インタフェースが提供されている。

2.2 非ブロッキング集団通信のアルゴリズム推進機構

非ブロッキング集団通信による通信隠蔽の効果は、MPI ライブラリでの実装手段に大きく影響を受ける。特に、集団通信アルゴリズムを他の処理と並行して進行させるためのアルゴリズム推進機構の実装手段は、通信隠蔽効果への影響が大きい。この、アルゴリズム推進機構は、集団通信アルゴリズムを構成する通信関数やメモリコピー、計算等の処理を、集団通信アルゴリズムで定義された依存関係に従って順に発行する。

現在、MPI ライブラリで採用されているアルゴリズム推進機構としては、主に以下の三通りが挙げられる。

- MPI 関数呼び出し毎の推進
- インターコネクトのオフロード機能による推進
- プログレススレッドによる推進

このうち、MPI 関数呼び出し毎の推進とは、全ての MPI 関数内で、非ブロッキング集団通信を推進させるための推進ルーチンを実行するものである。この推進ルーチンは、その時点で進行中の全ての非ブロッキング集団通信について、アルゴリズムの依存関係に基づき、実行可能な命令を発行する。この推進機構を利用する場合、プログラマは、非ブロッキング集団通信の開始関数と完了待ち関数の間に、例えば MPI_Test 関数のように、プログラムの意味を変えない MPI 関数をプログラム中に挿入することで、完了待ちの前にアルゴリズムを進行させておくことが出来る。この推進機構による性能向上の効果は、通信隠蔽による通信時間の削減と、推進ルーチンのためだけに呼び出す MPI 関数のオーバヘッドのトレードオフとなり、十分な効果が

得られるか否かは、プログラムの構造やプログラマの能力に依存する。

一方、インターコネクトのオフロード機能による推進は、インターコネクトの Network Interface Card (NIC) やスイッチ等に集団通信のアルゴリズムを進行させるオフロード機能が用意されている場合に、非ブロッキング集団通信関数の内部でその機能を呼び出すものである。例えば Mellanox 社のインターコネクト技術である InfiniBand は、集団通信アルゴリズムを NIC で処理する CORE-Direct 機能や、スイッチで処理する SHArP 機能を備えている。また、これらの機能を非ブロッキング集団通信の内部で呼び出すよう実装されている MPI ライブラリとしては、MVAPICH2 や Open MPI、Mellanox 社の HPC-X [6]、等がある。基本的に、オフロード機能を有するインターコネクトが利用できる場合、この推進機構を用いることで、効果的に通信を隠蔽することが出来る。

これらに対して プログレススレッドによる推進は、MPI プログラムを進行させるスレッドとは別に、非ブロッキング集団通信アルゴリズムを進行させるためだけのスレッドを生成する。この推進機構は、使用するインターコネクトがオフロード機能を有しない場合や、プログラム中に適切に `MPI_Test` 関数等を挿入することが困難な場合でも、集団通信と他の処理を同時に進行させることが出来るため、容易に通信を隠蔽する手段として注目されている。そのため、代表的なオープンソースの MPI ライブラリである Open MPI、MPICH、MVAPICH2 のいずれも、プログレススレッドによる非ブロッキング集団通信推進機構が選択可能となっている。このうち Open MPI では、Hoefler らが開発した非ブロッキング集団通信ライブラリ LibNBC [7] をコンポーネントとして追加することにより、この推進機構を実装している。この推進機構は、図 1 に示す通り、メインスレッドが、ハンドルキューと呼ぶ待ち行列を介して、集団通信アルゴリズムを構成する処理をタスク単位でプログレススレッドに渡すことにより、アルゴリズムを推進させている。一方、MPICH および MVAPICH2 は、同様の待ち行列を持つ推進機構を独自に開発し、実装している。

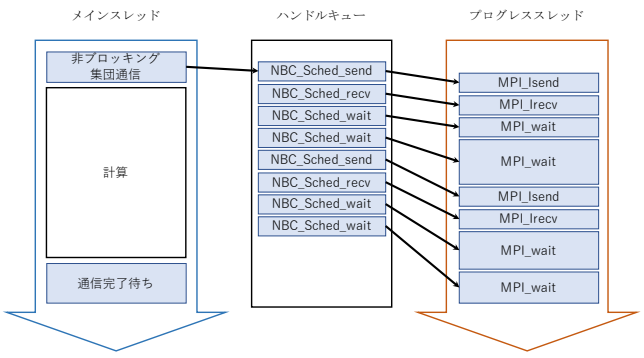

図 1　LibNBC におけるプログレススレッドによる推進機構

2.3　MVAPICH2 および Open MPI における非ブロッキング通信の実装

著者らは、過去の研究において、MVAPICH2 における非ブロッキング集団通信の通信性能と通信隠蔽効果を計測した [8]。その結果、プログレススレッドを使用したアルゴリズム推進機構を選択した場合、通信時間のうち通信を隠蔽できた時間の比率である通信隠蔽率は、ほぼ 100%となることを確認した。しかし、プログレススレッドを使用した場合、プログレススレッドを使用しない場合に比べて通信性能が著しく低下し、結果的に、全体的な性能向上が得られないことが判明した。そこで、この通信性能低下の原因を調査したところ、MVAPICH2、および MPICH において、プログレススレッドの使用を選択した場合、MPI の通信関数を排他的に実行するよう実装されており、この排他制御のための POSIX Thread の Mutex Lock 関数での待ち時間が、性能を大幅に低下させていることがわかった。これは、本稿執筆時点の 2017 年 3 月における最新バージョンである MVAPICH2 2.2 においても、同様であることを確認している。

一方、Open MPI の最新バージョンである Open MPI 2.0.2 について調査したところ、プログレススレッドが利用可能となるのは Ethernet を用いる場合のみであり、しかも、用意されているアルゴリズムが、アルゴリズム内の依存関係がほとんど無い Basic Linear アルゴリズムのみであることから、現時点では、プログレススレッドを使用することによる通信隠蔽の効果がほとんど期待できないことがわかった。

2.4　非ブロッキング集団通信実装手段の選択

非ブロッキング集団通信の使用にあたって利用者が選択すべき事項のうち、性能に与える影響が大きいものとしては、前節までで紹介したアルゴリズム推進機構の他に、使用する集団通信アルゴリズム、およびプログレススレッドへの CPU コアの割り当て方が挙げられる。

集団通信アルゴリズムは、プロセス間の一対一通信やプロセス内のメモリコピー、演算を組み合わせ、その集団通信で定義された通りにデータのコピーや集約を行うものであり、通常、一つの集団通信に対して複数のアルゴリズムが用意されている。アルゴリズム間の優劣は、プロセス数やメッセージサイズ、計算機の性能、プログラムの負荷バランス等、様々な要因に影響される。さらに非ブロッキング集団通信の場合、アルゴリズム毎に通信隠蔽の効果の得られやすさが異なる上、通信と並行して実行する処理の量によっても優劣が変動する可能性がある。このように、アルゴリズムの優劣に影響する要因は多様である上、実行してみなければわからないものもある。一方、現在利用可能な MPI ライブラリでは、アルゴリズムの選択に用いられる指標はメッセージサイズとプロセス数だけである場合が

多く、予め設定したそれらの指標の閾値に基づいて選択するため、実行時の状況に応じた選択は行えない。

また、プログレススレッドを使用する場合、プログレススレッドへの CPU コアの割り当て方として、[9] での Hoefler らの指摘にあるように、Spare Core と Fully Subscribed の二通りがある。Spare Core とは、プログレススレッドに一つの CPU コアを占有させるもので、高い通信隠蔽効果を期待できる。しかし、計算に割り当てる CPU コアが 1 プロセス当たり 1 コアずつ削減されるため、特に MPI のみによる並列プログラムや、ハイブリッド並列でプロセスあたりのスレッド数が少ない場合、計算性能の低下による影響が無視できなくなると予想される。一方、Fully Subscribed は、計算用のスレッドに全ての CPU コアを割り当てておき、プログレススレッドには、どれかの計算用スレッドと CPU コアを共有させるものである。これは、計算スレッドの空き時間に集団通信アルゴリズムを進行させることを期待するもので、Spare Core と比べると、計算性能の低下による影響は少ないと期待できる。しかし、スレッド切り替えのオーバヘッドや、進行状況を確認する頻度の低下により、通信隠蔽効果は低下すると予想される。この、プログレススレッドへの CPU コアの割り当て方についても、計算量やスレッド並列の並列化効果等の実行時の状況によって、優劣が変動する可能性がある。

3. 実装手段を動的選択する非ブロッキング集団通信ライブラリ NBC-SIA

本研究では、実装手段を動的選択する非ブロッキング集団通信ライブラリ NBC-SIA を開発する。NBC-SIA の構成を図 2 に示す。

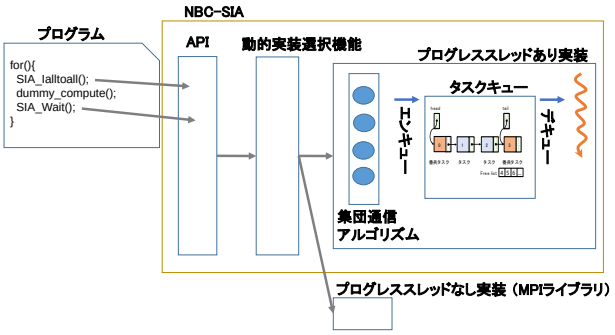

図 2　NBC-SIA の構成

プログラムから、NBC-SIA のプログラミングインタフェースを介して非ブロッキング集団通信が呼び出されると、内部の動的選択機構により、実行時の状況に応じて、使用するアルゴリズムや、プログレススレッドの有無等の実装手段が選択される。また、プログレススレッドを利用する実装が選択された場合、NBC-SIA 内で用意されたアルゴリズム推進機構を利用して、アルゴリズムを進行する。一方、プログレススレッドを利用しない実装が選択された場合、MPI ライブラリで提供されている非ブロッキング集団通信関数を呼び出す。

3.1 プログラミングインタフェース

NBC-SIA で提供するプログラミングインタフェースを表 1 に示す。このうち `SIA_Init` 関数は、NBC-SIA で用いるタスクキュー等のデータ構造の初期化、およびプログレススレッドの生成等を行う。一方、`SIA_Finalize` 関数は、これらのデータ構造の破棄と、プログレススレッドの join を行う。他の全ての NBC-SIA の関数は、プログラム中のこれらに挟まれた範囲で呼び出す必要がある。

表 1　NBC-SIA のプログラミングインタフェース

関数名	機能
`SIA_Init`	NBC-SIA の初期化
`SIA_Ialltoall_init` 等	非ブロッキング集団通信の初期化
`SIA_Start`	非ブロッキング集団通信の開始
`SIA_Wait`	非ブロッキング集団通信の完了待ち
`SIA_Hint`	プログラムのヒント情報提供
`SIA_Finalize`	NBC-SIA の終了

集団通信の関数としては、初期化、開始、完了待ちのそれぞれを用意する。これは、永続型通信インタフェースと呼ばれる形式である。また、NBC-SIA ライブラリの動作の効率化に有用なプログラム情報をプログラマが提供する手段として、ヒント関数を用意する。本節では、これらのインタフェースについて説明する。

3.1.1 永続型集団通信インタフェース

NBC-SIA では、非ブロッキング集団通信のインタフェースとして、永続型集団通信インタフェースを採用する。これは、プログラム中の集団通信関数を呼び出す位置毎に、独立して実装手段の選択を行うためである。

従来の非ブロッキング集団通信によるプログラム例を、図 3 に示す。MPI では、`MPI_Wait` 関数等で非ブロッキング集団通信の完了が分かった時点で、リクエスト変数の値を、初期状態を示す `MPI_REQUEST_NULL` とする。そのため、MPI ライブラリの内部では、ループ内の 1 回目の `MPI_Ialltoall` 関数呼び出しと 2 回目の `MPI_Ialltoall` 関数呼び出しを区別することが出来ない。その結果、このインタフェースで実装手段の動的選択を行う場合、全ての `MPI_Ialltoall` 関数呼び出しに対して、同じ選択しかできない。

一方、NBC-SIA で提供する永続型集団通信インタフェースによるプログラム例を、図 4 に示す。このインタフェースでは、非ブロッキング通信の開始関数の内部処理のうち、初期化部分を分離し、独立した関数として呼び出す。この

```
int main (...) {
  ...
  for(...) {
    MPI_Ialltoall(..., &requestA);
    computeA();
    MPI_Wait(&requestA, &stat);
    MPI_Ialltoall(..., &requestB);
    computeB();
    MPI_Wait(&requestB, &stat);
  }
  ...
}
```

図 3　MPI の非ブロッキング集団通信を用いたプログラム

初期化関数は、その集団通信呼び出しに関する情報を、集団通信完了後もリクエスト変数に保持し続ける。そのため、NBC-SIA の動的選択機構で同じ選択をしたい集団通信呼び出し毎に一つずつ初期化関数を呼び出すことで、同じ集団通信であってもリクエスト変数毎に別の選択を行うことが出来る。

このインタフェースは、MPI の次期規格を検討する団体である MPI Forum において、次の規格での採用に向けて議論が進んでいる永続型集団通信インタフェースの案に基づいている。そのため、次期 MPI 規格でこのインタフェースが採用されれば、NBC-SIA を MPI ライブラリ内のコンポーネントとして利用することが出来る。なお、現在用意している初期化関数は `SIA_Ialltoall_init` のみであり、今後、全ての集団通信について初期化関数の整備を進める予定である。

```
int main (...) {
  ...
  SIA_Ialltoall_init(..., &requestA);
  SIA_Ialltoall_init(..., &requestB);
  for(...) {
    SIA_Start(&requestA);
    computeA();
    SIA_Wait(&requestA, &stat);
    SIA_Start(&requestB);
    computeA();
    SIA_Wait(&requestB, &stat);
  }
  ...
}
```

図 4　NBC-SIA の永続型集団通信インタフェースを用いた非ブロッキング集団通信プログラム

3.1.2　プログラムのヒント情報提供関数

NBC-SIA は、後述する通り、繰り返し呼び出される集団通信について、呼び出し毎に一つずつ、選択可能な実装手段を試すことで、各実装手段での性能を計測、収集し、最終的に最も高速だったものを選択する。そのため、呼び出し回数が不十分な場合、オーバヘッドの影響により、動的選択を行わない方が高い性能が得られる可能性がある。また、明らかに遅い実装がある場合、その選択肢を除外することで、より早い段階で最速の実装手段を選択できるようになる。

これらの情報は、集団通信に必須のものではないため、通信関数の引数として渡すべきものではない。しかし、プログラムのヒント情報として NBC-SIM に提供できれば、動的最適化の効率化を図ることが出来る。そこで、プログラムのヒント情報を提供するための関数として、`SIA_Hint` を用意している。この関数は、キーと値を引数として受け取る。現在、`SIA_Hint` 関数に渡すキーとしては、集団通信の呼び出し回数、プログレススレッドの有無、使用する集団通信アルゴリズムの番号、等を用意している。

なお、このインタフェースの設計は、MPI-3.0 規格で採用された MPI Tool のインタフェースに基づいている。そのため、将来、NBC-SIA を MPI ライブラリ内に実装する際、その MPI ライブラリの MPI Tool インタフェースに追加する形で、容易に移植できる。

3.2　排他制御オーバヘッドを低減したアルゴリズム推進機構

前述の通り、従来の MPI ライブラリでは、プログレススレッドによるアルゴリズム推進機構を使用する場合、通信関数毎の排他制御を行うため、通信性能が大幅に低下する。そこで NBC-SIA では、排他制御が不要な番兵付きのタスクキューを用いることにより、低オーバヘッドでプログレススレッドにアルゴリズムを進行させる推進機構を構築した。

NBC-SIA のタスクキューを図 5 に示す。計算スレッドは、非ブロッキング集団通信開始時に、集団通信アルゴリズムを構成するそれぞれの処理を、タスクとしてタスクキューの末尾にエンキューする。この操作で計算スレッドが書き換えるのは、タスクキューの tail ポインタと、tail 位置の番兵タスクの内容である。一方、プログレススレッドは、タスクキューの先頭から、実行可能となったタスクを探索して取り出し、そのタスクに該当する MPI 関数呼び出しや、コピー、計算等の処理を行う。なお、NBC-SIA では、複数の非ブロッキング集団通信を並行して進行させることも許可している。この場合、タスクキューの中に、複数の集団通信のタスクが混在するため、タスクキューから取り出されるタスクは、必ずしも先頭位置であるとは限らない。しかし、その場合でも、これらの操作でプログレススレッドが書き換えるのは、タスクキューの tail 位置の番兵タスクより前のタスクまでである。このように、プログレススレッドと計算スレッドで書き換える箇所が分離さ

れているため、これらの操作に排他制御は不要である。

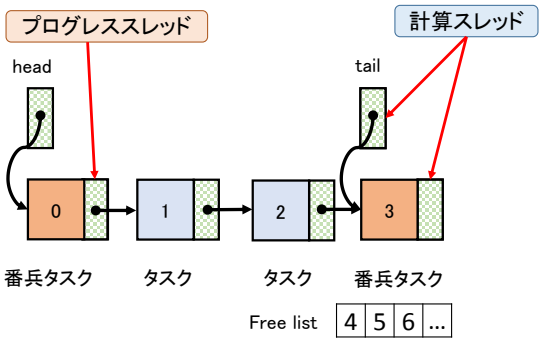

図 5　NBC-SIA のタスクキュー

なお、この推進機構では、プログレススレッドへの CPU コアの割り当てが Fully Subscribed である場合、計算スレッドにおける `SIA_Wait` での待ち時間、およびプログレススレッドにおけるタスクキューが空の間の待ち時間のそれぞれについて、そのスレッドに、sleep させるか、もしくは busy wait させるかを、`SIA_Hint` 関数を用いて指示できる。ここで、sleep させる場合、そのスレッドの再開には、スレッドへのシグナルを送付するため、排他制御が必要となる。ただし、この排他制御は、使用頻度が少なく、全体的な性能への影響は限定的であると予想される。

3.3　実装手段の動的選択機構

NBC-SIA の動的選択機構は、STAR-MPI [11] と同様に、選択可能な実装手段を、集団通信の呼び出し毎に一つずつ試行して、それぞれの計測結果から、最適なものを選択する。例えば、8 通りの実装手段がある集団通信の場合、プログラム中で、その集団通信の呼び出しの 1 回目から 10 回目までは、1 番目の実装手段を用い、11 回目から 20 回目までは 2 番目の実装手段を用いる、という流れで、各実装手段での、通信開始から完了までの所要時間を計測する。80 回の呼び出しを使って、全ての実装手段を試行すると、各プロセスで最速だった実装手段を選び、その中で、多数決により、81 回目以降の呼び出しで使用する実装を決定する。

現在の NBC-SIA の実装では、以下の実装手段の組み合わせを選択可能としている。なお、実験的なものであるため、集団通信としては Alltoall のみを提供している。

- プログレススレッドの利用 (無し / Spare Core / Fully Subscribed)
- 集団通信アルゴリズム (Bruck / Ring / Basic Linear)
- Fully Subscribed の場合の計算スレッドの待ち方 (Sleep / Busy Wait)
- Fully Subscribed の場合のプログレススレッドの待ち方 (Sleep / Busy Wait)

これらの選択項目のうち、プログレススレッドの使用の有無については、初期化時にプログレススレッドを生成しておき、使用しない間は sleep させて、使用する際にシグナルで有効にすることで、切り替える。

一方、プログレススレッドを使用する際の、CPU コアの割り当て方の変更は、計算スレッドのスレッド数を OpenMP のスレッド数設定関数 `omp_set_num_threads` で変更することで行う。なお、現在の NBC-SIA では、アプリケーション内が OpenMP でスレッド並列化されており、かつ、計算スレッド数が実行中に変動しても問題ない場合を想定しているため、この変更を常に適用可能としている。実際には、この想定が成り立たないアプリケーションもあるため、NBC-SIA の次期バージョンでは、この変更の可否についてヒント提供関数で設定できるよう改良する。

また、集団通信アルゴリズムやスレッドの待ち方は、計測結果の集計時に使用する方法の情報を broadcast することで切り替える。

4.　性能評価

4.1　実験方法

今回の実験では、NBC-SIA における通信時間隠蔽の効果の検証、実装手段の動的選択の必要性の確認、および NBC-SIA の動的選択機構の精度の評価を目的とし、OSU Micro-Benchmarks をもとにしたベンチマークを用いて、MVAPICH と NBC-SIA 上で、性能を計測した。実験に使用した計算機は、FUJITSU PRIMERGY RX200 S7 を 16 台、InfiniBand FDR で接続した PC クラスタである。ノード当たり Intel Xeon CPU を 1 基搭載し、クロック周波数は 2.4GHz、CPU コア数は 4、ノード当たりの主記憶容量は 8GB である。また、カーネルのバージョンは 2.6.32-220.el6.x86_64 であり、MPI ライブラリには MVAPICH2 2.2 を用いた。NBC-SIA も、この MPI ライブラリ上に構築した。

MVAPICH2 構築時の configure コマンドのオプションとしては、`--enable-threads=multiple --enable-mpit-pvars=all` を指定した。さらに、実行時の環境変数は、`MV2_ENABLE_AFFINITY=0`, `MV2_SMP_USE_CMA=0`, `OMP_NUM_THREADS=4`, `OMP_SCHEDULE='static'` を指定した。なお、MVAPICH2 では、環境変数に，`MV2_ASYNC_PROGRESS=1` を指定することでプログレススレッドを利用する事が可能となるため，MVAPICH2 でプログレススレッドを利用した実験を行う際には、この環境変数を設定した．

ベンチマークプログラムは OSU Micro-Benchmarks 5.3.2 [10] をベースに、通信と並行して実行するベクトル積計算の時間を、通信時間によらず一定となるように変更したプ

ログラムを作成して、使用した。また、ベクトル積計算は OpenMP を用いて並列化した。今回の実験では，ベクトル積に用いるベクトルの長さを 100,000 および 5,000,000 とした場合について測定を行った．

4.2 プログレススレッドによるアルゴリズム推進機構の性能評価

まず、プログレススレッドによるアルゴリズム推進機構の性能を比較するため、計算を含まない、非ブロッキング Alltoall 通信のみの性能を計測した。図 6 に、通信開始から完了までの所要時間を示す。横軸はメッセージサイズ、縦軸は所要時間である。MVAPICH no progress は、MVAPICH でプログレススレッドを使わなかった場合の所要時間、MVAPICH Spare core および MVAPICH Furry subscribed は、MVAPICH でプログレススレッドを使った場合の、それぞれの CPU コア割り当て方法での所要時間、SIA は、NBC-SIA を用いた場合の所要時間である。いずれのメッセージサイズでも、MVAPICH のプログレススレッドを使用した場合に比べ、NBC-SIA で所要時間を大幅に短縮できていることから、NBC-SIA のプログレススレッドによるアルゴリズム推進機構が、

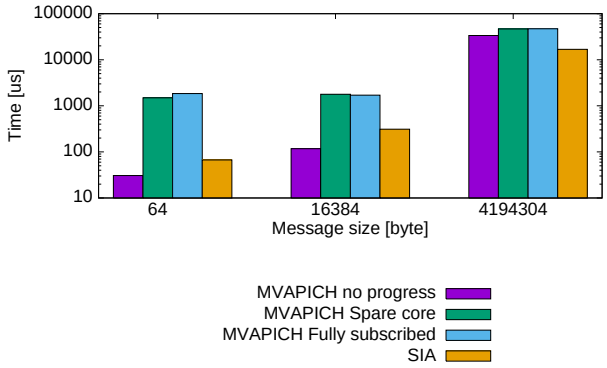

図 6 MVAPICH2 と NBC-SIA における非ブロッキング Alltoall 通信の所用時間

また、通信と計算を並行に実行した場合の通信隠蔽率を、図 7 に示す。通信隠蔽率とは、通信時間全体のうち、計算によって隠蔽できた時間の比率である。NBC-SIA の通信隠蔽率は、MVAPICH2 でプログレススレッドを使わなかった場合に比べ十分高く、効果的に隠蔽できていることが分かった。なお、MVAPICH2 でプログレススレッドを使った場合、通信隠蔽率は高いものの、前述の通り、通信に要する時間が大幅に増加するため、プログラム全体の性能を向上させることは困難である。

4.3 実装手段の動的選択の必要性の確認

ベクトルサイズが 100,000 の場合と 5,000,000 の場合の、非ブロッキング集団通信の実装手段の違いによるプログラム実行時間の相違を、図 8 と図 9 に、それぞれ示す。SIA Spare Basic Linear、SIA Spare Bruck、SIA Spare Ring は、Spare Core で CPU コアをプログレススレッドに割り当てた場合の、各アルゴリズムでの性能を示す。一方、SIA no progress Basic Linear、SIA no progress Bruck、SIA no progress Ring は、プログレススレッドを用いなかった間合いの、各アルゴリズムでの性能を示す。なお、今回の実験環境では、Fully Subscribed による性能が、他の実装手段に比べて大幅に低かったため、グラフから除外している。

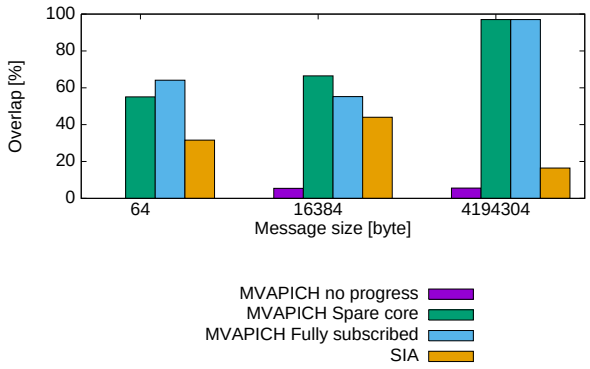

図 7 MVAPICH と NBC-SIA における通信隠蔽率

図より、ベクトルサイズが小さい場合、メッセージサイズが大きくなるとプログレススレッドを用いた場合が高速になるのに対し、ベクトルサイズが大きい場合、どのメッセージサイズでもプログレススレッドを用いない場合が高速になることがわかった。通信ライブラリの内部では、メッセージサイズやプロセス数は、関数の引数により取得できるものの、通信と並行して実行される計算の量に関する情報は取得できない。そのため、適切な実装手段の選択には、実行時の状況に応じた動的な選択機構が必要であることがわかった。

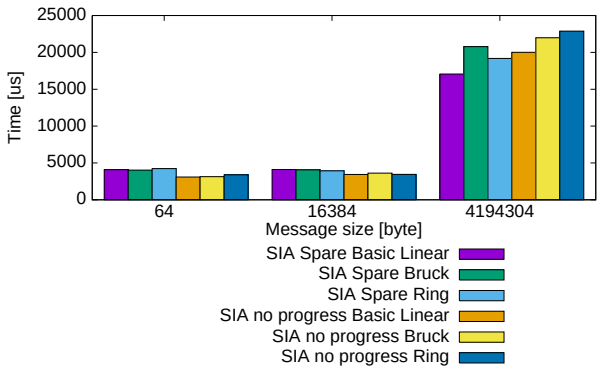

図 8 実装手段による実行時間の相違 (ベクトルサイズ = 100,000)

4.4 動的選択機構の精度

NBC-SIA における動的選択機構が、正しく実装手段を選択できているか否かを検証するため、実装手段を固定し

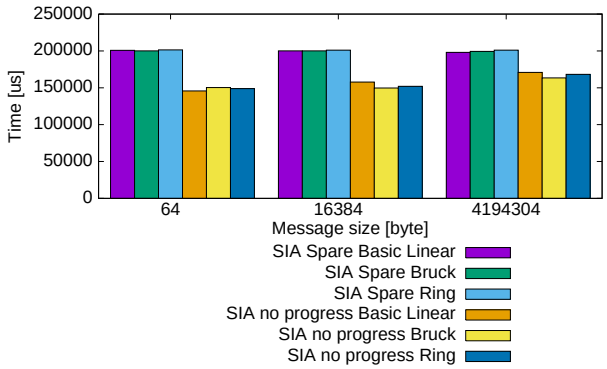

図 9 実装手段による実行時間の相違 (ベクトルサイズ = 5,000,000)

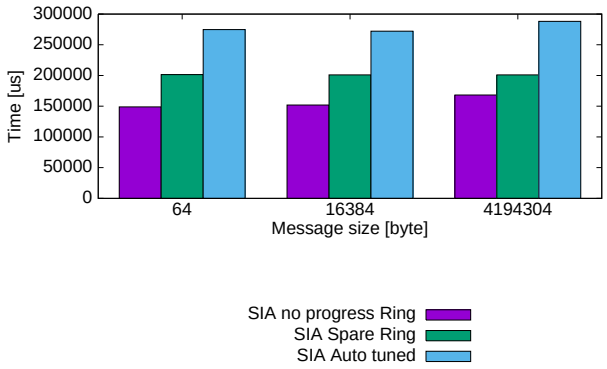

図 11 自動選択の有無による実行時間の相違 (ベクトルサイズ = 5,000,000)

た場合と、動的に自動選択した場合で比較した。ベクトルサイズが 100,000 の場合と 5,000,000 の場合の、プログラム全体の所要時間の相違を、図 10 と図 11 に示す。なお、前節の結果から、今回の実験環境とプログラムでは、アルゴリズム間の性能差が大きくないことがわかったため、実装手段を固定した場合の性能としては、Ring アルゴリズムのもののみを示している。SIA Auto tuned が、自動選択した場合の性能である。

いずれのベクトルサイズの場合も、実装手段を固定した場合に対して、実装手段を自動選択した場合の性能が大きく悪化していることがわかった。また、原因究明のため実行時の経過を解析した結果、自動選択時に、Fully Subscribed が、多くのプロセスで高速に動作したため、多数決で最適だと判定されたことが分かった。しかし、いくつかのプロセスでは Fully Subscribed の所要時間が極端に長かったうえ、他のプロセスがこのプロセスの完了を待って次の処理に移るため、プログラム全体の所要時間が、他の実装手段より大幅に長くなっていた。この問題を解決する手段として、NBC-SIA の次期バージョンでは、各実装手段について、最も所要時間の長かったプロセスでの計測値で比較し、最速のものを選択するよう、実装を変更する予定である。

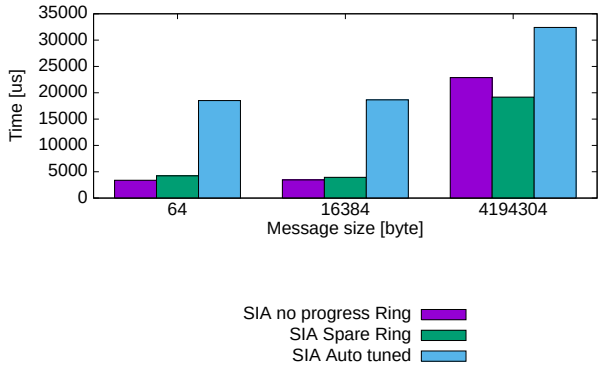

図 10 自動選択の有無による実行時間の相違 (ベクトルサイズ = 100,000)

5. 関連研究

集団通信の動的最適化技術として、Faraj らは、ブロッキング集団通信における動的アルゴリズム選択を可能とした集団通信ライブラリ Self Tuned Adaptive Routines for MPI collective operations (STAR-MPI) を提案した [11]。これは、NBC-SIA と同様に、実行時に集団通信アルゴリズムを 1 つずつ試し、最速だったものを選択する技術である。しかし STAR-MPI は、非ブロッキング集団通信に対応していない。また、プログラム中の集団通信の呼び出し位置毎にアルゴリズム選択を行えるよう、MPI 規格の集団通信関数では定義されていない、call site と呼ぶ引数を、各集団通信に追加している。

一方、非ブロッキング集団通信を対象とした動的最適化機構を持つ通信ライブラリとして、Barigou, Gabriel らは Abstract Data and Communication Library(ADCL) がある [12][13]。しかし、ADCL で動的選択の対象となっているのは集団通信アルゴリズムのみであり、プログレススレッドの使用の有無の選択は行えない。

6. むすび

本研究では、実行時の状況に応じて実装手段を選択する動的選択機構を持つ非ブロッキング集団通信ライブラリ NBC-SIA を開発した。また、NBC-SIA における実装手段の一つとして、排他制御オーバヘッドを低減したプログレススレッドによるアルゴリズム推進機構を開発した。実験の結果、現時点では、NBC-SIA における実装手段の性能を集計する手法に問題があり、正しい選択が行えないものの、実行時に判明する状況によって最適な実装手段が変動することがわかり、動的選択機構の必要性を確認できた。また、NBC-SIA におけるプログレススレッドによるアルゴリズム推進機構が、既存の MVAPICH2 における実装に対して、より効率よく動作することを示した。今後の予定としては、実装手段選択手法を改良するとともに、Alltoall 以外の集団通信の整備を進める。

参考文献

[1] MPI-3.0 Draft. https://www.mpi-forum.org/.
[2] MPICH. http://mvapich.cse.ohio-state.edu/.
[3] MVAPICH2. https://www.open-mpi.org/.
[4] Open MPI. https://www.open-mpi.org/.
[5] Intel MPI Library. https://software.intel.com/en-us/intel-mpi-library.
[6] Mellanox HPC-X http://www.mellanox.com/page/hpcx_overview
[7] Torsten Hoefler, Andrew Lumsdaine, and Wolfgang Rehm. Implementation and Performance Analysis of Non-Blocking Collective Operations for MPI. *Proceedings of the 2007 ACM/IEEE conference on Supercomputing - SC '07*, p. 1, 2007.
[8] 成林晃, 南里豪志, 天野浩文. progress thread を用いた非ブロッキング集団通信の性能調査. 第 155 回ハイパフォーマンスコンピューティング研究会, 2016.
[9] Torsten Hoefler and Andrew Lumsdaine. Message progression in parallel computing - To thread or not to thread? In *Proceedings - IEEE International Conference on Cluster Computing, ICCC*, Vol. Proceeding, pp. 213–222, 2008.
[10] OSU Micro-Benchmarks. http://mvapich.cse.ohio-state.edu/benchmarks/.
[11] 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*, pp. 199—-208, 2006.
[12] Youcef Barigou, Vishwanath Venkatesan, and Edgar Gabriel. Auto-tuning Non-blocking Collective Communication Operations. pp. 1204–1213, 2015.
[13] Edgar Gabriel, Saber Feki, Katharina Benkert, and Michael M Resch. Towards Performance and Portability through Runtime Adaption for High Performance Computing Applications. *Proceedings of the International Supercomputing Conference June 1720 2008 Dresden Germany*, Vol. 22, No. 16, p. accepted for publication, 2010.