様　式　Ｃ－１９、Ｆ－１９－１、Ｚ－１９　（共通）

科学研究費助成事業　　研究成果報告書

平成　３０　年　　６　月　１４　日現在

機関番号：　１７１０２
研究種目：　基盤研究(C)（一般）
研究期間：　2015～2017
課題番号：　１５Ｋ００１７２

研究課題名（和文）MPI向け準備型集団通信インタフェースの研究開発

研究課題名（英文）Development of Persistent Collective Communication Interface for MPI

研究代表者
南里　豪志（NANRI, TAKESHI）

九州大学・情報基盤研究開発センター・准教授

研究者番号：７０２８４５７８

交付決定額（研究期間全体）：（直接経費）　　2,500,000円

研究成果の概要（和文）：科学技術シミュレーションやデータサイエンスなど、様々な分野の大規模並列プログラムで広く用いられている集団通信について、事前に通信相手や通信データの場所、大きさなどを登録することにより効率的な実行を可能とする、準備型インタフェースを提案し、プロトタイプを実装した。このインタフェースは、同じパターンで何度も集団通信を実行する際に、通信のためのネゴシエーションなどの処理が不要となる上、計算と通信を並行して進めることで、実質的に通信時間を隠蔽することが出来る。実験により、提案したインタフェースの通信効率化の効果を確認した。

研究成果の概要（英文）：Collective communication is a set of patterns of group communication that are popularly used in large-scale parallel programs of various fields, such as scientific simulations and data sciences. This project proposed an interface that enables persistent style of collective communication. With this interface, by registering target processes, data locations and data sizes of the communication, overheads for the negotiations of communication can be eliminated. Moreover, the interface provides higher possibility of overlapping communication over computation so that it can hide the time for communication. Results of experiments on the prototype of the proposed interface showed its effects on the optimization of communications.

研究分野：　計算機科学

キーワード：　並列計算　高性能計算　通信効率化

１．研究開始当初の背景
今後さらなる大規模化が予想されるスーパーコンピュータにおいて、通信時間がプログラムの性能に与える影響が深刻になる。特に集団通信は、計算機の規模に応じて所要時間が増加するため、特に実装の効率化が求められる。そこで本研究では、準備型集団通信インタフェースを設計し、実装する。従来の集団通信では、準備の不要な低速通信を用いるか、もしくは毎回高性能通信の準備を行う必要があったため、集団通信の所要時間が問題となっていた。一方、提案インタフェースでは、一度準備が完了した集団通信について、高性能通信を用いたデータ転送が可能となり、通信時間を大幅に短縮出来る。

２．研究の目的
本研究の目的は、準備型集団通信インタフェースが、利便性を損なわずに集団通信の性能向上に寄与することを示し、今後の普及につなげることである。そのため、準備型集団通信インタフェースのプロトタイプを実装して有効性を検証する。

３．研究の方法
まず本研究で提案する準備型集団通信インタフェースの仕様を確定する。特に、繰り返し呼び出される集団通信について、パラメータ群を予め登録し、プログラム中で再利用できる準備開始関数インタフェースを定義する。次に、このインタフェース仕様に基づいてプロトタイプを実装する。実装においては、NIC が使用する一時的な記憶領域の確保や、参加するプロセス間の情報交換等を、準備開始関数内で実行するように実装する。最後に、既存の MPI ライブラリを調査し、提案インタフェースの実装手段について考察する。

４．研究成果
４．１　準備型集団通信ライブラリ NBC-SIA の実装
本研究では、実装手段を動的選択する準備型集団通信ライブラリ NBC-SIA を実装した。NBC-SIA の構成を図 1 に示す。

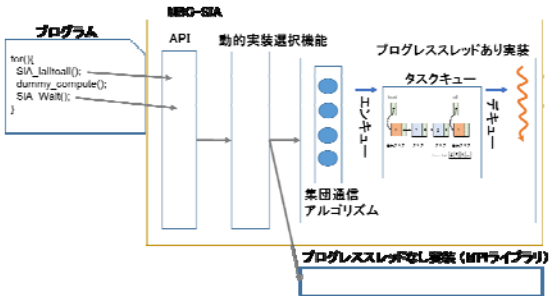

図 1 NBC-SIA の構成

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。

４．１．１ NBC-SIA の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

SIA_Init	NBC-SIA の初期化
SIA_Ialltoall_init	準備型集団通信の初期化
SIA_Start	準備型集団通信の開始
SIA_Wait	準備型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_Finazlize	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。

４．１．１．１ 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、準備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 2 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。

```
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 2 NBC-SIA を用いたプログラム例

４．１．１．２ プログラムのヒント情報提供関数

NBC-SIA は、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。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。

４．１．２ 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。

この推進機構では、計算スレッドが、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。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。

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。

４．１．３ 実装手段の動的選択機構

NBC-SIA の動的選択機構は、STAR-MPI 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。

- 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。

４．２ 性能評価
４．２．１ 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 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.

４．２．２ 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 3 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、低オーバヘッドで実装できていることがわかる。

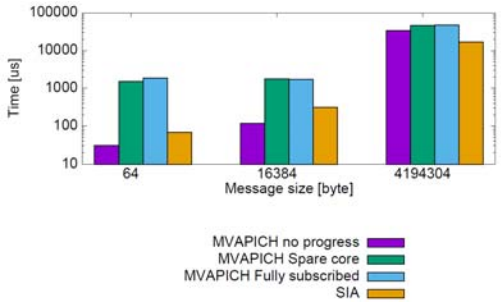

図 3 MVAPICH2 と NBC-SIA における Alltoall 通信の所用時間

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 4 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わかった。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。

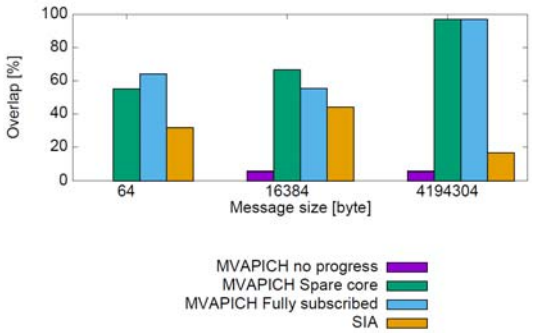

図 4 MVAPICH と NBC-SIA における通信隠蔽率

４．２．３ 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 5 と図 6 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。

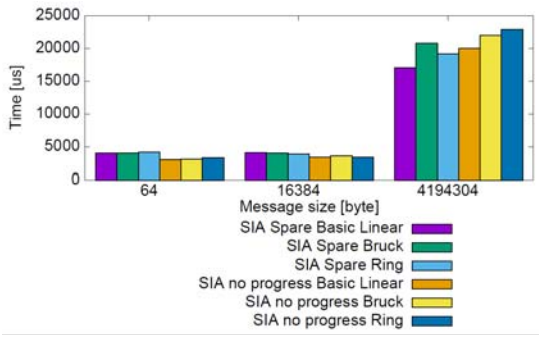

図 6 実装手段による実行時間の相違（ベクトルサイズ = 100,000）

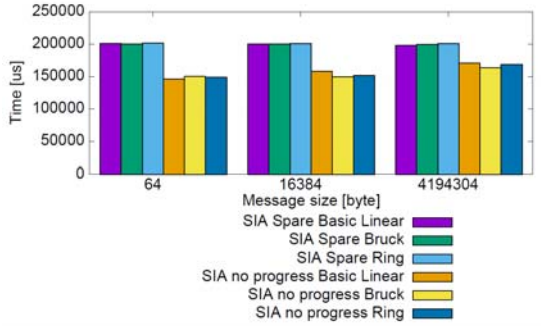

図 5 実装手段による実行時間の相違（ベクトルサイズ = 5,000,000）

５．主な発表論文等

（研究代表者、研究分担者及び連携研究者には下線）

〔雑誌論文〕（計 0 件）

〔学会発表〕（計 2 件）

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, 研究報告ハイパフォーマンスコンピューティング (HPC), 2016-HPC-155, 2016

成林 晃, 南里 豪志, 天野 浩文, 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, 研究報告ハイパフォーマンスコンピューティング（HPC）, 2017-HPC-159, 2017

南里 豪志, 大島 聡史, 小野 謙二, 非ブロッキング集団通信の通信隠蔽効果に関する調査, 研究報告ハイパフォーマンスコンピューティング（HPC）, 2017-HPC-162, 2017

〔図書〕（計 件）

〔産業財産権〕

○出願状況（計 0 件）

○取得状況（計 0 件）

〔その他〕

ホームページ等

６．研究組織

(1)研究代表者

南里 豪志（TAKESHI NANRI）

九州大学・情報基盤研究開発センター・准教授

研究者番号：21799936

研究者番号：