様　式　Ｃ－１９、Ｆ－１９、Ｚ－１９　（共通）

科学研究費助成事業　　研究成果報告書

平成 ２７ 年　 ６ 月　 ８ 日現在

機関番号：１７１０２
研究種目：基盤研究(C)
研究期間：2012～2014
課題番号：２４５００６８

研究課題名（和文）並列言語ＣＡＦプログラム向け通信隠蔽技術の研究開発

研究課題名（英文）Study on Communication Overlapping for Parallel CAF Programs

研究代表者
南里　豪志（NANRI, TAKESHI）

九州大学・学内共同利用施設等・准教授

研究者番号：７０２８４５７８

交付決定額（研究期間全体）：（直接経費）　　1,900,000円

研究成果の概要（和文）：並列計算の大規模化に伴い，通信時間の増大が並列計算の性能向上を阻害する要因となる．特に多数のプロセスによる定型パターン通信である集団通信は，多くのアプリケーションで使われているうえに，プロセス数に応じて通信時間が増加するため，性能への影響が大きい．
そこで本研究では，集団通信を計算と並行して実行することで通信時間を隠ぺいするための非ブロッキング集団通信と呼ばれる技術について，いくつかの手法による実装と評価を行った．さらに，並列プログラムの特徴に応じて最適な実装手法を選択するための基本的な調査を行った．

研究成果の概要（英文）：Non-blocking collective communication is a key technique for enabling overlap of collective communication and computation to achieve higher scalability on large scale parallel computers. There can be many types of implementation methods for non-blocking collective communications. To sufficiently obtain effect of overlap,
appropriate method should be chosen. Therefore, it is important to study characteristics of each method. This work examined a characteristic of each method of implementation and evaluated the characteristic by performing the experiment using a simple test program. In addition to that, this work proposed an implementation with lower overhead, and evaluates the performance. From the results of these experiments, there have been some investigations to understand how to choose an appropriate method according to the characteristics of the programs.

研究分野：並列処理

キーワード：並列計算　高性能通信　オーバーラップ

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は、より深刻になると予想される。そこで、通信を計算と並行して実行することによって通信時間を隠蔽する技術が、プログラムの通信効率化手段として提案され、様々なアプリケーションに適用されている。これにより大規模化に伴う性能劣化を防ぎ、計算機の規模に応じた性能向上を期待できるようになる。
通常、この通信隠蔽は、通信されるデータ範囲の解析、計算されるデータ範囲の解析、計算を通信結果に依存しない部分と依存する部分に分離、通信結果に依存しない部分の計算と通信を同時実行する通信コード作成、を必要とする。これらの作業は非常に時間がかかるため、自動的に行う技術の開発が求められている。そこで本研究では、プログラム中で呼び出される集団通信とその前後の計算を対象として、上記の解析とプログラム変形を行う技術を研究開発することにより、プログラマの負担を増やすことなく、大規模計算環境でのプログラムの通信効率化を図る。

２．研究の目的
本研究では、プログラムで呼び出される集団通信のデータ転送範囲と、その前後の計算におけるデータ参照範囲を解析する技術と、これらの解析結果に基づいて、通信結果に依存せず実行可能な計算を分離して、集団通信と同時に実行させるようプログラムを変形する技術を研究開発し、それにより、並列プログラムにおける通信隠蔽の可能性を明らかにする。さらに、様々な並列プログラムに対してこの手法を適用し、実行性能を計測することにより、この自動通信隠蔽技術の効果を検証する。

３．研究の方法
まず、通信隠蔽で最も重要となる非ブロッキング型の集団通信実装に取り組んだ。非ブロッキング型集団通信は、通信の完了を待たずに関数の呼び出し元に戻り、次の処理を始める。そのため、アルゴリズムの進み具合を確認し、進行させるための処理が必要となる。そのための手法として、既存のものと本研究で新たに提案するものをそれぞれ実装し、性能を比較した。その結果、プログラムの特性によって最適な手法が異なることがわかったため、並列プログラムにおいて非ブロッキング集団通信の効果に影響を与える特徴の抽出と、それに基づいた最適なアルゴリズム進行手法の選択方法について検討した。

４．研究成果
まず、既存の非ブロッキング集団通信実装手法として、LibNBC を調査した。LibNBC は、集団通信アルゴリズムを MPI ライブラリの非ブロッキング一対一通信や計算等の命令に分解して並べ、それらの命令を個別に実行することでアルゴリズムを進行させる手法を用いている。
この手法により複数の非ブロッキング集団通信を並行して進めることが可能となる。一方、この手法では、アルゴリズムを構成する命令の単位が小さくなるため、その構成にかかるオーバヘッドが必要となる。
LibNBC の実装は、アルゴリズムを進行させる方法により、さらにプログレス関数による実装とプログレススレッドによる実装に分類される。このうちプログレス関数による実装では、アルゴリズムを進行させるための関数であるプログレス関数を、プログラム中で繰り返し呼び出すことによりアルゴリズムを進行させる。プログレス関数はユーザが明示的に呼び出す必要があり、呼び出す頻度もユーザ自身が決める。これは、ユーザがアルゴリズムの進行を自ら管理することが可能になる、という利点であると同時に、最適な頻度でプログレス関数を呼び出すように調整する負担をユーザに強いる、という欠点でもある。
一方、プログレススレッドによる実装では、メインスレッドとは別に生成されたスレッドへアルゴリズムの進行を任せる。この生成されたスレッドをプログレススレッドと呼ぶ。この実装では、プログレススレッドが自動的にアルゴリズムを進行するので、ユーザがアルゴリズムの進行を負担する必要が無い。さらに、メインスレッドでプログレス関数を呼びだすことによるオーバヘッドが必要でなくなる。ただし、プログレススレッドに専用のコアを１つ割り当てる場合、そのコアは計算を全く行うことが出来ない。一方、専用のコアを割り当てない場合、計算を行うスレッドとプログレススレッドでコアを取り合うこととなるため、コアに対するスレッドのスケジューリングが性能に影響する。
このように、LibNBC の実装では集団通信アルゴリズムの構成におけるオーバヘッドが必要となる。そこで、集団通信を一対一通信に分解せず集団通信ごとに処理することにより低オーバヘッドで非ブロッキング集団通信を実装する手法 TCC(Threaded Collective Communication) を提案した。TCC では、まず集団通信を進行させるためのプログレススレッドを生成する。このプログレススレッドは、メインスレッドにおける非ブロッキング集団通信命令の発行順に、対応する MPI ライブラリのブロッキング集団通信命令を実行する。
TCC では集団通信アルゴリズムの構成において、集団通信を １単位として扱う。そのため、LibNBC のように集団通信を分解した非ブロッキング一対一通信や演算を１単位とする場合に比べ、容易にアルゴリズムを構成することができる。したがって、TCC のアルゴリズム構成方法はオーバヘッドが低い。ただし、必ずプログレススレッドを生成しなければ

ならず、LibNBC のプログレススレッドによる実装と同じく、プログレススレッドに専用のコアを 1 つ割り当てか否かを選択しなければならない。
TCC では、ハンドルおよびハンドルキューというデータ構造を用いて呼び出された非ブロッキング集団通信を管理する。ハンドルは、集団通信の種類、その集団通信に渡された引数、集団通信が終了したかどうかを判定するためのフラグからなる構造体であり、呼び出した非ブロッキング集団通信一つにつき一つのハンドルが対応している。ハンドルキューは、ハンドルを格納しておくためのリングキューである。このハンドルキューに格納されるハンドルを介してメインスレッドとプログレススレッドが連携して集団通信を進めて行く。これらのスレッドは POSIX スレッドで実装し、ハンドルキューは排他制御変数と条件変数で排他的に管理する。
TCC による非ブロッキング集団通信は、プログラミングインタフェースとして、初期化関数、各非ブロッキング集団通信関数、Wait 関数と終了処理関数を提供する。初期化関数では、ハンドルキューの排他制御変数、条件変数の初期化を行う。非ブロッキング集団通信関数では、ハンドルに集団通信の種類と引数を設定し、さらに終了判定フラグの初期化を行い、最後にハンドルをハンドルキューにエンキューする。エンキューの際にハンドルキューが一杯になっている場合は、空きができるまで条件変数によって待つ。Wait 関数では、指定されたハンドルの終了判定フラグが真になるまで待つ。終了処理関数では、まず、発行された全ての非ブロッキング集団通信の完了を待つためにハンドルキューの残り要素数が 0 になるまで待ち、その後、プログレススレッドへ終了を知らせるシグナルを送り、プログレススレッドの終了を待つ。
プログレススレッドは、ハンドルキューからハンドルを 1 つデキューし、そのハンドルに設定された集団通信の種類に対応した MPI のブロッキング集団通信を行う。デキューの際、ハンドルキューが空の場合は、新しいハンドルがエンキューされるまで条件変数によって待つ。また、集団通信が終了した後、ハンドルの終了判定フラグを真にする。
各実装による計算と通信のオーバラップの効果を評価するため、LibNBC および TCC による非ブロッキング集団通信について実験を行った。なお、LibNBC は集団通信アルゴリズムを分解して命令毎に実行するため、独自のアルゴリズムを用いて実装されている。一方 TCC では、MPI の集団通信をプログレススレッドで呼び出すため、MPI ライブラリ内部のアルゴリズムが用いられる。そのため、LibNBC と TCC の実験結果を単純に比較することはできない。MPI ライブラリ内部で用いられているアルゴリズムを分解して LibNBC で用いることにより、LibNBC と TCC の比較が可能となるため、今後の課題である。

実験に用いたプログラムは、非ブロッキング Alltoall 通信を発行した後に行列ベクトル積の計算を行うものである。このプログラムを用いて所要時間の計測を行った。行列ベクトル積の計算では、2 重ループの外側のループに対して MPI によるブロック並列化を行うと同時に、OpenMP によるスレッド並列化を行うことでハイブリッド並列を施した。また、比較のため、LibNBC、TCC それぞれについて、行列ベクトル積の直前で非ブロッキング集団通信を完了させることにより通信を隠蔽させないプログラム、および MPI ライブラリのブロッキング集団通信と行列ベクトル積を順に実行するプログラムについても計測も行った。
実験に使用した計算機は、九州大学が所有する Fujitsu PRIMERGY CX400 である。この計算機の総ノード数は 1476 ノードであり、このうち 32 ノードを使用して実験を行った。各ノードには 16 基のコアと 128GB の主記憶容量が搭載されている。CPU は Intel(R) Xeon(R) CPU E5-2680 2.7GHz である。ノード間インターコネクトとしては、InfiniBand FDR が用いられている。OS は Red Hat Enterprise Linux Server release 6.1 である。コンパイラは Intel Composer XE 2011、MPI ライブラリは Intel(R) MPI Library 4.0 Update 3 for Linux を用いた。また、本実験ではプログレススレッドによる非ブロッキング集団通信の進行を行うため、MPI のスレッドモードとして THREAD_MULTIPLE を用いた。実験では、以下の 5 ケースについて計測を行った。

NBCF-noovlp：
LibNBC のプログレス関数による実装において通信隠蔽を行わない場合
NBCF-ovlp-test：
LibNBC のプログレス関数による実装において通信隠蔽を行い、プログレス関数を行列ベクトル積の外側ループ 1 回につき 1 度呼ぶ場合
NBCF-ovlp-notest：
LibNBC のプログレス関数による実装において通信隠蔽を行い、プログレス関数を一度も呼ばない場合
NBCT-ovlp：
LibNBC のプログレススレッドによる実装において通信隠蔽を行う場合
MPI：
MPI ライブラリの Alltoall 関数を用いて通信隠蔽を行わない場合
TCC-ovlp：
TCC による実装において通信隠蔽を行う場合

計算ノード数が 2 ノードおよび 32 ノードの場合の所要時間を図 1 および図 2 に示す。各ノードで MPI のプロセス数は 1 とし、スレッド並列による行列ベクトル積の計算において生成するプロセスごとのスレッド数は、

プログレススレッドを用いない実装の場合 16 としたのに対し、プログレススレッドを用いる実装の場合は 15 として、プログレススレッドに 1 コアを専有させた。また、メッセージサイズと行列のサイズは、ノード数 2 の場合がメッセージサイズ 5MB、行列サイズ 4,000 x 4,000、ノード数 32 の場合がメッセージサイズ 500KB、行列サイズ 30,000 x 30,000 とした。

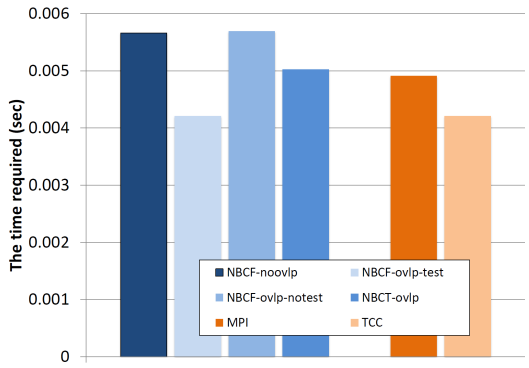

図 1 各ケースの所用時間(ノード数 2、メッセージサイズ 5MB、行列サイズ 4,000 x 4,000)

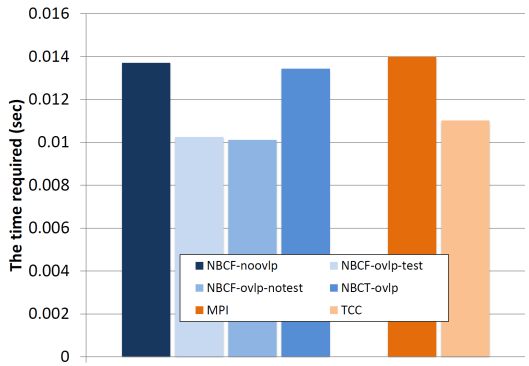

図 2 各ケースの所用時間(ノード数 32、メッセージサイズ 500KB、行列サイズ 30,000 x 30,000)

はじめに、LibNBC による実験結果について考察する。通信隠蔽の効果については、どのケースでも通信隠蔽する場合の方が通信隠蔽しない場合より速いか、もしくは同等の所要時間であったことから、有効性を確認できた。また、プログレス関数による実装とプログレススレッドによる実装を比較すると、プログレス関数による実装の方が速い、という結果となった。この理由として、プログレススレッドによる実装におけるスレッド生成や排他制御等のオーバヘッドの影響が考えられる。一方、プログレス関数の呼び出しの有無による違いについては、32 ノードではほぼ同じ所要時間だったのに対し、2 ノードにでは大きな差が見られた。今回の実験においては、LibNBC の Alltoall 通信は、他プロセスとの非ブロッキング一対一通信を一度に全て発行し、後はその完了を待つというアルゴリズムを用いているため、アルゴリズムを進行させる操作が必要無く、プログレス関数の呼び出しの有無で結果がほぼ変わらないと予測していたため、この結果については今後原因究明のための解析を行う。

次に、TCC による実験結果について考察する。TCC においても、ノード数 2、32 の双方で通信隠蔽の効果を確認することができた。なお、非ブロッキング集団通信の同時呼び出し数が少ない場合、集団通信アルゴリズムを細かい命令に分けて進行させる LibNBC に比べて TCC の方式が有効だと考えられる。実プログラムでも今回の実験と同様の状況は多い。しかし、今回の実験では、前述の通り、LibNBC との単純な比較はできないので、この有効性の実測による確認は今後の課題である。

さらに、今回得られた実験結果に基づいた非ブロッキング集団通信の適応的な自動選択へ向けての指針を述べる。

まず、プログラマがプログレス関数を実行する適切な頻度を探ることができる場合について考える。今回の実験では、プログレス関数を呼び出す頻度は基本的に行列ベクトル積の内側ループ 1 回につき 1 度、もしくは外側ループ 1 回につき 1 度、の 2 通りが考えられる。このように、プログレス関数を実行する頻度の選択肢が少ない場合、最適な頻度を探ることはプログラマにとってあまり負担にはならない。実際に、今回は内側と外側の 2 通りを試し結果の良いほうを選んだが、わずか 2 通りであったため大きな負担とはならなかった。したがって、この場合には、実験により得られた通信隠蔽の効果とプログラマの負担を比較すると、プログレススレッドにアルゴリズムの進行を任せる必要はなく、プログラマがプログレス関数を明示的に呼び出したほうが良いと言える。

次に、プログラマがプログレス関数を実行する適切な頻度を探ることが困難な場合について考える。これは、非ブロッキング集団通信とオーバラップさせる計算が複雑で、プログレス関数を実行する適切な頻度の予想や実測が困難な場合である。そのような例としては、3 重以上の多重ループや、複雑な条件分岐のある計算とオーバラップをする状況が考えられる。

この場合は、プログレス関数による実装では、プログレス関数の適切な実行頻度を探るためのプログラマの負担が大きい。そのため、プログレススレッドを用いた非ブロッキング集団通信実装の利用が好ましい。しかし今回の実験結果によると、LibNBC を用いた場合のプログレススレッドによる実装での通信隠蔽の効果は、ノード数 2 においては、通信隠蔽によるスピードアップは約 1.127 であ

り、プログレス関数による実装の約 1.346 に比べて小さい。また、ノード数 32 においては、プログレススレッドによる実装のスピードアップは約 1.020 であり、プログレス関数の約 1.338 との差が大きくなっている。これは、プログレススレッドのために計算スレッドを一つ減らしていること、および、プログレススレッドと計算スレッドの間の同期コストの影響が考えられる。そのため、LibNBC による実装では、可能な限りプログラマによるプログレス関数の挿入を選択する方が良いと思われる。
なお、今回の実験ではオーバラップさせる集団通信の数が 1 つだけであったのに対し、LibNBC の実装の利点として、複数の非ブロッキング集団通信を並行して進められることがある。そのため、そのような状況では LibNBC による通信隠蔽の効果がより顕著に表れる可能性がある。
一方 TCC では、ノード数 2 におけるスピードアップが約 1。166、ノード数 32 におけるスピードアップが約 1。271 であった。TCC は、同時に一つの非ブロッキング集団通信しか進行させることが出来ないが、構造が単純であるためオーバヘッドが少なく、今回のようにオーバラップさせる集団通信が一つの場合、通信隠蔽の効果が得られやすいと考えられる。
以上のことから、今回の実験により、非ブロッキング集団通信の実装手段を選択するに当たり、プログレス関数の適切な実行頻度を見つけるのが容易であるか否か、および同時に進行させる集団通信の数が、重要な指標であることが分かった。また、適切な実装手段選択の実現に向け、複数の非ブロッキング集団通信を進行させる場合の LibNBC での通信隠蔽の効果の検証、同じアルゴリズムによる LibNBC と TCC の性能比較が必要であることが分かった。
なお、プログレススレッドを用いる場合については、ノード内スレッド数による性能への影響も検証する必要がある。今回の実験では、プログレススレッドを用いる際、計算スレッドを一つ減らして 1 コアをプログレススレッドに占有させた。一方、実行時の選択肢としては、計算スレッド数を減らさず、スレッドがコアを取り合うよう実行させることも可能である。これは、1 コア分の計算能力の減少と、スレッドを取り合ってコンテキストスイッチを繰り返すことによるコスト増のトレードオフとなる。

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）
[1] Tsuyoshi Okuma and Takeshi Nanri, "Performance Study of Non-blocking Collective Communication Implementations Toward Adaptive Selection, Networking," Computing, Systems and Software, 2013.12.
[2] Tsuyoshi Okuma and Takeshi Nanri、"Evaluation of Implementation Methods for Non-Blocking Collective Communications in Overlapping Communication and Computation," International workshop on HPC, Krylov Subspace method and its application、2013.01.

〔図書〕（計　0 件）

〔産業財産権〕
○出願状況（計　0 件）

名称：
発明者：
権利者：
種類：
番号：
出願年月日：
国内外の別：

○取得状況（計　0 件）

名称：
発明者：
権利者：
種類：
番号：
出願年月日：
取得年月日：
国内外の別：

〔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