スーパーコンピュータシステム ITO における並列計算の通信隠蔽技術と効果

南里 豪志 [1)]

1) 九州大学 情報基盤研究開発センター

nanri.takeshi.995@m.kyushu-u.ac.jp

Study on the Methods and Effects of Communication Hiding for Parallel Processing on Supercomputer System ITO

Takeshi Nanri[1)]

1) Research Institute for Information Technology, Kyushu Univ.

概要

本稿では、大規模並列計算でより高い演算性能を達成するために重要となる通信の隠蔽技術について、最新のインターコネクトネットワークを採用した九州大学のスーパーコンピュータシステム ITO における効果を調査した。計測対象のプログラムとしては、隣接プロセス間で一対一通信を繰り返すステンシル計算、集団通信と行列計算を並行して行うオーバラップベンチマークプログラム、および、反復ごとに集団通信を呼び出す偏微分方程式ソルバーフレームワーク PETSc を用いた。通信隠蔽を行った場合と行わなかった場合の所要時間を比較したところ、ステンシル計算については、非ブロッキングの一対一通信関数を用いた場合も、片側通信関数を用いた場合も、適切に MPI ライブラリを選ぶことによって隠蔽効果が得られることがわかった。一方、集団通信を用いた 2 つのプログラムについては、特に Mellanox 社の SHARP と呼ばれるオフロード技術を利用可能な MPI ライブラリ HPCX が、集団通信の処理をスイッチ装置に任せることができるため、高い通信隠蔽効果が期待されたものの、今回の計測では明確な効果が得られなかった。

1 背景

一般に、並列計算に参加する計算ノードを増やすことによって、並列計算の実行時間のうち、計算に要する時間の短縮が期待できる。しかし、計算ノード間や、ノード内のプロセス間で呼び出される通信は、計算ノード数の増加にともなって増加するか、もしくは不変である。その結果、実行時間に占める通信時間の比率が高くなり、並列計算における性能向上の阻害要因となる。これに対して非ブロッキング通信は、あるプロセス間の通信と、それらのプロセス上での計算を並行して進行させることにより、見かけ上、通信に要する時間の一部を計算時間で隠蔽することを可能とする。

プロセス並列プログラムにおける通信の標準規格である Message Passing Interface (MPI) [1] で定義されている、主な非ブロッキング通信関数として、非ブロッキング一対一通信関数の `MPI_Isend`、`MPI_Irecv` や、非ブロッキング集団通信関数の `MPI_Iallreduce`、`MPI_Iallgather`、`MPI_Ialltoall` 等がある。また、他プロセスのメモリ空間に対して非同期にデータを読み書きする片側通信関数 `MPI_Get`、`MPI_Put` 等も、一対一通信の通信時間隠蔽に用いることができる。

これらの通信隠蔽技術による効果は、実行する計算機や使用する MPI ライブラリに大きく依存する。そこで本稿では、執筆時点で最新の高性能インターコネクトネットワークである Mellanox 社の InfiniBand EDR を採用した九州大学のスーパーコンピュータシステム ITO において、MPI ライブラリによる通信隠蔽効果の違いを計測して評価する。計測対象のプログラムとしては、隣接プロセス間で一対一通信を繰り返すステンシル計算プログラム、集団通信と行列計算を並行して行うオーバラップベンチマークプログラム、および、偏微分方程式ソルバーフレームワーク PETSc を用いる。

2 通信隠蔽技術

並列計算における通信を隠蔽するには、通信操作を非ブロッキングで行うことが必要である。非ブロッキングな操作とは、その操作を開始したのち、完了を待たずに別の処理に移ることを許す操作である。MPI では、一対一通信、集団通信それぞれについて、非ブ

ロッキングに操作するための関数が用意されている。

2.1 一対一通信の通信隠蔽技術

2.1.1 非ブロッキング一対一通信

一対一通信とは、送信側と受信側のそれぞれが参加してデータを転送する 2 プロセス間通信である。MPI において、この一対一通信を非ブロッキングに行うもっとも一般的な方法は、非ブロッキング一対一通信関数を用いるものである。これは、送信プロセスにおけるデータの送信開始を `MPI_Isend` 等の非ブロッキング送信関数で、受信プロセスにおける受診開始を `MPI_Irecv` 等の非ブロッキング受信関数でそれぞれ指示する。通常、MPI ライブラリは、これらの関数で通信開始のための手続きを済ませた後、完了を待たずに次の処理に移る。その代わり、これらの関数は、完了を待つためのリクエスト識別子を返す。このリクエスト識別子を引数として `MPI_Wait` 関数等の完了待ち関数を呼び出すと、その通信の完了を待つことが出来る。非ブロッキング通信関数呼び出し後、対応する完了待ち関数の完了までの間は、データの送信元、送信先のどちらの領域も状態が不確定であり、それらの領域へのアクセスを避ける必要がある。

非ブロッキング一対一通信は、送信元でデータが生成されたタイミングで送信開始を、送信先でデータの上書きが可能になったタイミングで受診開始を、それぞれ指示することで、不正にデータが上書きされる心配なく、通信を隠蔽したコードを簡潔に記述できる。しかし、MPI ライブラリによっては、転送データサイズが大きい場合、Rendezvous プロトコルのように送信側と受信側で同期を取って転送を開始するプロトコルが選択される。この場合、`MPI_Wait` 関数が呼び出されるまで同期が取れず、通信をほとんど隠蔽できない可能性がある。

2.1.2 片側通信

MPI における片側通信とは、データの転送を、送信プロセスが相手のプロセスへの書き込みを行う `MPI_Put` 関数や、受信プロセスが相手のプロセスからの読出しを行う `MPI_Get` 関数等、片方のプロセスの操作で表現するものである。

MPI の片側通信には、Active Target と Passive Target の二つの方式が用意されている。このうち Active Target の片側通信は、通信の発行側と相手側の間で同期を取りながらデータを転送する。また、`MPI_Win_fence` 関数のように、全プロセスで同期を取り、それまでに全プロセスで発行された片側通信の完了を保証するものもある。

一方 Passive Target の片側通信は、通信の相手側が何もしなくても、データの転送が開始されるものである。そのため、書き込み先のメモリ領域が上書きしても問題ない状況か否か、や、読み出し元のメモリ領域に適切なデータが既に用意されているか否か、等、通信の整合性の確認をプログラム中で明示的に行う必要がある。しかし、相手側の状態によらずデータ転送を開始できるため、Rendezvous プロトコルによる一対一通信や、Active Target の片側通信よりも高い通信隠蔽効果が期待できる。

2.1.3 ACP ライブラリ

Advanced Communication Primitives (ACP) とは、片側通信を基本インタフェースとする、Partitioned Global Address Space (PGAS) 型の通信ライブラリである [2]。一対一通信と違って片側通信は、送信側と受信側の間で暗黙的な同期が行われないため、待ち合わせのための通信バッファ領域を必要とせず、同期待ちのオーバヘッドも基本的に発生しない。そのため ACP は、計算機が大規模化しても少ないメモリで高速な通信が可能な、スケーラブルな通信ライブラリである。プロセス間で非同期にデータが転送されるため、ACP では、MPI の Passive Target の片側通信と同様に、通信の整合性の確認をプログラム中で明示的に行う必要がある。

2.2 集団通信の通信隠蔽技術

2.2.1 非ブロッキング集団通信関数

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関数は、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 [1]。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様に `MPI_Wait` 関数や `MPI_Waitall` 関数等に指定することで通信の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 [3]、MVAPICH2 [4]、Open MPI [5] 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 [6] や富士通社製 MPI 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。また、Mellanox 社のツールキット HPC-X には、SHARP [8] を用いた集約操作の集団通信実装を選択可能な MPI ライブラリが用意されている [7]。

2.2.2 非ブロッキング集団通信の推進機構

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の実装手段は、通信隠蔽効果への影響が大きい。現在、MPI ライブラリで採用されているアルゴリズム推進手法としては、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
- プログレススレッドによる推進
- インターコネクトネットワークのオフロード機能による推進

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。この推進手法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 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。この推進手法は、使用するインターコネクトネットワーク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の推進を選択可能となっている。ただし、本稿執筆時点の 2018 年 9 月における最新バージョンである Open MPI 3.1.1 では、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った。一方、MPICH および MVAPICH2 でもプログレススレッドにタスク単位でアルゴリズムを推進させる機構を実装しており、InfiniBand 上でも選択可能である。

これらに対してインターコネクトネットワークのオフロード機能による推進は、インターコネクトネットワークの 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内からその機能を呼び出すものである。

2.2.3 SHARP プロトコル

Mellanox 社のオフロードプロトコル SHARP は、任意のプロセスグループ内のデータの集約操作をインターコネクトネットワークにオフロードするためのプロトコルである [8]。このプロトコルを用いることにより、インターコネクトネットワーク上でのスイッチ間のデータの転送と併せて集約演算を適用することが出来るため、計算ホスト上で演算適用する場合に比べ、データ転送回数や転送量を削減できる。また、計算ホストの CPU におけるデータ到着待ちが不要となり、負荷不均衡や OS ジッタによる性能への影響を軽減できる。さらに、集約操作の推進をインターコネクトネットワークに行わせることにより、CPU への負担が軽減できるとともに、高い通信隠蔽効果が期待できる。

このプロトコルは、集約対象のデータと、その結果得られるデータの要素数が同じである集約操作であることが、適用の条件となっている。そのため、MPI の集団通信関数のうち、MPI_Allreduce、MPI_Reduce、MPI_Bcast、MPI_Barrier、およびこれらの非ブロッキング版の実装に適している。また、集約時に適用できる演算としては、総和、最小値、最大値、OR、AND、XOR の他、MPI の MinLoc と MaxLoc がある。

3 計測に用いたプログラム

3.1 並列ステンシル計算

ステンシル計算は、科学技術計算で頻繁に用いられる形の計算の一つで、多次元配列で表された場において、配列の各要素の値を、その要素に隣接する要素群の値から計算される値で更新するものである。通常、この計算をプロセス並列化する場合、対象の配列を矩形のブロックに分割し、それらを各プロセスに割り当てて計算させる。

このように並列化されたステンシル計算では、要素を更新するステップ毎に、隣接するブロックを担当す

るプロセス間でブロックの境界に位置する要素の値を交換する、HALO 通信と呼ばれる隣接通信が必要となる。各プロセスから見た HALO 通信の通信相手は、配列の分割方法、分割されたブロックのプロセスへの割り当て方法、および、ステンシル計算に用いられる隣接要素の相対位置に依存する。例えば、場が 2 次元配列で表され、水平方向の次元でブロックに分割されるステンシル計算で、上下左右の隣接 4 要素で要素の更新を行う場合、HALO 通信で隣のプロセスに送付すべき領域は、ブロックの境界の 1 列分である。

この、隣のプロセスに送付すべき領域の要素群がメモリ上で連続していない場合、基本的には、不連続な要素を一つずつ送付する必要がある。しかしこの方法では、通信ごとに必要な遅延時間の影響で、十分な性能が得られない。そこで通常は、別途確保した送信バッファと呼ばれる連続領域に送信する要素をコピーすることで、複数の要素をまとめて一回の送信関数で転送可能とする。この方法は、パッキングと呼ばれる。また、この方法で送られたデータは、受信側で行うアンパッキングによって、各要素を最終的な目的の場所にコピーする。

本稿では、このパッキング/アンパッキングを行う並列ステンシル計算を、以下の 4 つの方法で実装した。

ISND ：MPI_Isend と MPI_Irecv による方法
ACTV ：Active Target の片側通信を用いる方法
PASV ：Passive Target の片側通信を用いる方法
ACP ：ACP による片側通信を用いる方法

このうち ACTV(図 1) では、MPI_Win_fence 関数を用いて 2 箇所で同期をとっている。最初の呼び出しは、各プロセスがデータを受け取る準備ができたことを確認するための同期であり、2 つ目の呼び出しは、全プロセスのデータ転送が完了したことを確認するための同期である。

一方 PASV(図 2) では、計算途中に誤ってブロックの境界領域が上書きされないように、データ転送とは別に MPI_Put 関数を用いて相互に進行状況を示すフラグを送信している。また、ACP による並列ステンシル計算でも、Passive Target の片側通信によるものと同様の方法で、不正な境界領域の上書きを防いでいる。

3.2 Overlap Test ベンチマーク

Overlap Test は、著者らが非ブロッキング集団通信の隠蔽効果を計測するために作成したベンチマークプログラムである。このプログラムは、計算する行列サイズ、および集団通信のメッセージサイズを実

```
setup arrays
MPI_Win_allocate()

for (steps)
  pack
  MPI_Win_fence
  MPI_Put to left
  MPI_Put to right
  calculate inner elements
  MPI_Win_fence
  unpack
  calculate elements of edges
```

図 1　Active Target の片側通信による並列ステンシル計算

```
setup arrays
MPI_Win_allocate()
*lrdy = 1; *rrdy = 1;
for (steps)
  ctr++
  pack
  MPI_Win_flush_all
  lflg = 0; rflg = 0;
  while ((lflg == 0)||(rflg == 0)
    if ((lflg == 0)&&(*lrdy >= ctr))
      lflg = 1
      MPI_Put data to left
      MPI_Put ctr to rack on left
    if ((rflg == 0)&&(*rrdy >= ctr))
      rflg = 1
      MPI_Put data to right
      MPI_Put ctr to lack on right
  calculate inner elements
  lflg = 0; rflg = 0;
  while ((lflg == 0)||(rflg == 0)
    if ((lflg == 0)&&(*lack >= ctr))
      lflg = 1
    if ((rflg == 0)&&(*rack >= ctr))
      rflg = 1
  unpack
  MPI_Put ctr to rrdy on left
  MPI_Put ctr to lldy on right
  calculate elements of edges
```

図 2　Passive Target の片側通信による並列ステンシル計算

行時パラメータとして取得し、ブロッキング集団通信の後に計算を実行した場合の所要時間、および、非ブロッキング集団通信開始後、計算を実行してから通信完了を確認した場合の所要時間をそれぞれ計測する。今回の計測で使用した集団通信は集約操作である MPI_Allreduce および MPI_Iallreduce で、集約演算としては倍精度実数の総和を用いた。また、通信と並行して行う計算としてはベクトル行列積を用い、2 重ループをプロセス及びスレッドに分割したハイブリッド並列で計算する。

3.3 PETSc

PETSc は、偏微分方程式でモデル化された科学技術アプリケーションための、並列化されたソルバ等の関数群とデータ構造群を提供するツールキットである [9]。この中に含まれている線型方程式の反復解法の中には、ノルム計算等で必要となる集約操作を非ブロッキング化し、ベクトル行列積等の計算と並行して行うことで、通信時間の隠ぺいを図るアルゴリズムを実装したものがある。本稿では、それらのうち、Pipelined Conjugate Gradient (PIPE CG) 法を対象として、SHARP による通信隠蔽の効果を調査する。

PIPE CG 法では、計算の順序を調整し、ノルム計算やベクトルの内積計算をベクトル行列積の前で開始している。その後、それまでに開始したノルム計算やベクトル計算で必要となる集約操作のための通信を、まとめて開始する。この通信の完了は、対応する計算の完了を待つ関数の中で確認される。各計算で集約操作の対象となるデータは倍精度実数 1 個ずつなので、基本的に、1 回の反復で 24 バイトの非ブロッキング集約操作が、ベクトル行列積と並行して行われる。

4 通信隠蔽効果の計測

4.1 実験環境

今回の実験に使用したのは、九州大学情報基盤研究開発センターのスーパーコンピュータシステム ITO のサブシステム A である。計算機の諸元を表 1 に示す。

SHARP を用いる MPI ライブラリとしては、Mellanox HPC-X 2.1.0 に付属するものを用いた。また、比較対象の MPI ライブラリとして、Open MPI 3.1.1、および MVAPICH2 2.2 を用いた。このうち MVAPICH2 では、プログレススレッドを用いた場合と用いなかった場合のそれぞれについて、計測した。

なお、ITO のサブシステム A は 2000 台の計算ノードを Full Bisection Bandwidth で接続しており、使用計算ノードの配置による性能への影響は少ないと考えられるものの、出来るだけ同条件で計測するため、同じプログラムを同じ計算ノード数で実行する場合、使用する MPI ライブラリや計算ノード内プロセス数、メッセージサイズ等によらず、全てを同じジョブの中で実行した。

表 1 ITO のサブシステム A の仕様

CPU	Intel Xeon Gold 6154 (3.0GHz, 18core) x 2 / node
Memory	192GB / node
Interconnect	Mellanox InfiniBand EDR
# of nodes	2000 (うち 128 ノード使用)
OS	Red Hat Linux Enterprise
C Compiler	GCC 4.4.6
ドライバ	Mellanox OFED 4.0-2.0.0.1

4.2 ステンシル計算における一対一通信の隠蔽効果

配列全体の大きさを 512、1024 および 2048 としたときの並列ステンシル計算の HALO 通信に要した時間を、図 3、4、5 に、それぞれ示す。凡例で `mv-` が MVAPICH2、`om-`が Open MPI、`acp-`が ACP の結果である。また、末尾が `novlp` は計算と通信を並行して行わなかった場合の通信時間を、`isnd` は非ブロッキング一対一通信で通信隠蔽した場合、`actv` は Active Target の片側通信で通信隠蔽した場合、`pasv` は Passive Target の片側通信で通信隠蔽した場合、`ovlp` は ACP で通信隠蔽した場合、の隠蔽できなかった分の通信時間を、それぞれ示している。

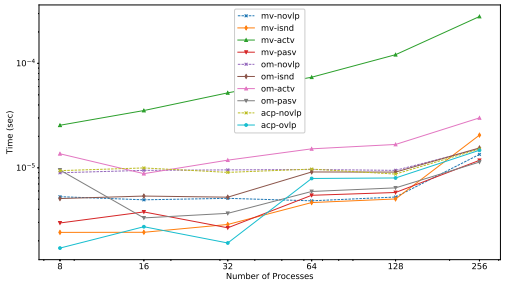

図 3 ステンシル計算の通信時間 (N = 512)

MVAPICH2 の Passive Target の片側通信や、ACP を用いて隠蔽した場合、ほとんどの計測結果で通信隠蔽しなかった場合に比べて見かけの通信時間が減っており、通信隠蔽の効果が得られていることが確認でき

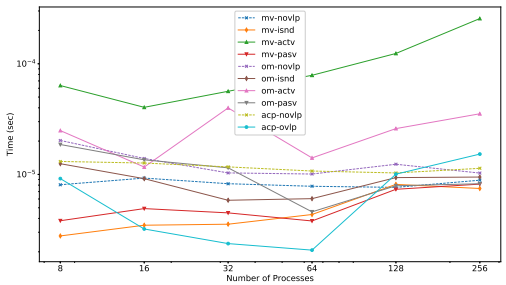

図 4　ステンシル計算の通信時間 (N = 1024)

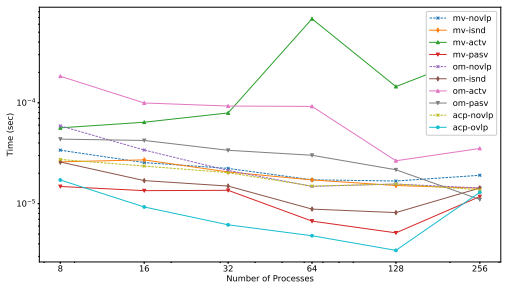

図 5　ステンシル計算の通信時間 (N = 2048)

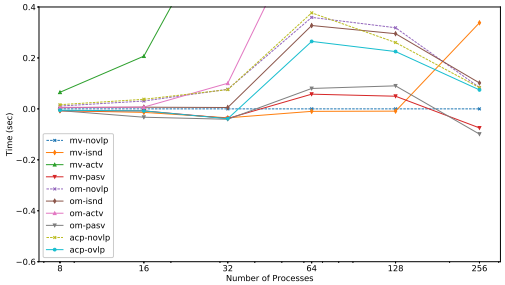

図 6　ステンシル計算全体の実行時間の比率 (N = 512)

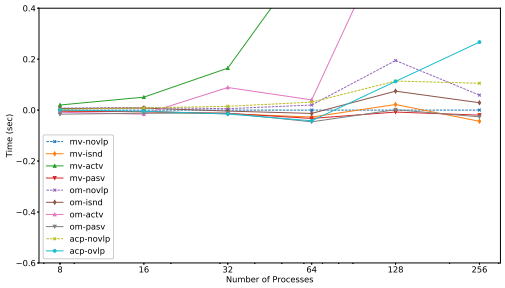

図 7　ステンシル計算全体の実行時間の比率 (N = 1024)

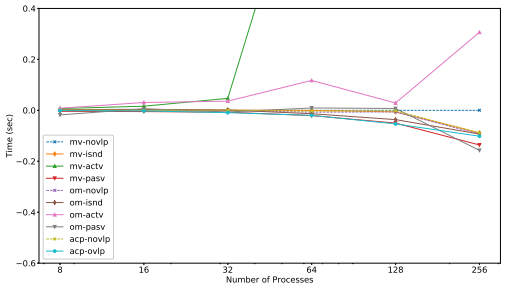

図 8　ステンシル計算全体の実行時間の比率 (N = 2048)

た。特に ACP は、プロセス数が 16、32、64、128 で N が 2048 の場合に高い隠蔽効果が見られた。一方、Open MPI の Passive Target の片側通信では、ほとんどの場合で通信隠蔽効果が得られなかった。

これに対して非ブロッキング一対一通信では、MVAPICH2 ではメッセージサイズが小さい場合、Open MPI ではメッセージサイズが大きい場合に、通信隠蔽効果が見られた。

また、Active Target の片側通信では、通信隠蔽しない場合よりも通信時間が増えた。これは、同期のオーバヘッドによるものと考えられる。

一方、ステンシル計算全体の実行時間について、MVAPICH2 による通信隠蔽なしの場合の実行時間に対する、他の各場合の実行時間の増減の比率を、図 6、7、8 に、それぞれ示す。

N が小さい場合は、全体の実行時間は通信隠蔽を行わない場合と同等か長くなっている。一方、N が 2048 の場合でプロセス数が大きくなると、Passive Target の片側通信や ACP で、実行時間が短縮できているのが分かる。

4.3　オーバラップベンチマークにおける集団通信の隠蔽効果

Overlap Test プログラムについては、計算と集約操作（`MPI_Iallreduce`）を並行して実行した場合の時間を計測した。また、比較のため、計算と集約操作を逐次的に実行した場合の時間も計測した。128 ノードを使用し、ノード内プロセス数やメッセージサイズを変化させて計測した結果を図 9 から図 12 に示す。それぞれの MPI ライブラリで、No Ovlp が、計算と集約操作を逐次的に実行した場合の時間、Ovlp が、並行して実行した場合の時間である。なお、今回の計測では、MVAPICH2 での結果が異常となったた

め、計測対象から除外した。Comm が通信のみの時間、Comm+Comp が通信を隠蔽せずに実行した場合の合計時間、Total が通信を隠蔽した場合の合計時間である。

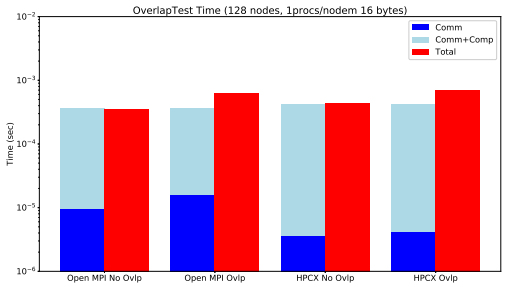

図 9　Overlap Test 実行時間 （1 プロセス/ノード 16 バイト）

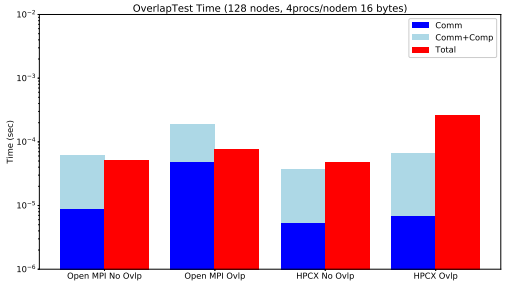

図 10　Overlap Test 実行時間 （4 プロセス/ノード 16 バイト）

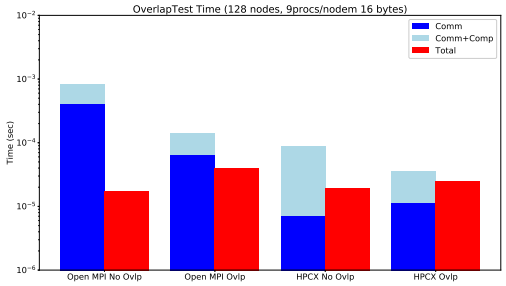

図 11　Overlap Test 実行時間 （9 プロセス/ノード 16 バイト）

いずれの場合も、通信時間は HPCX を用いることで短縮できた。これは、SHARP の効果と考えられる。一方、通信隠蔽を行うと、いずれの場合も合計時間が増加した。これは、非ブロッキング集団通信の処理に CPU が関与したためと考えられる。

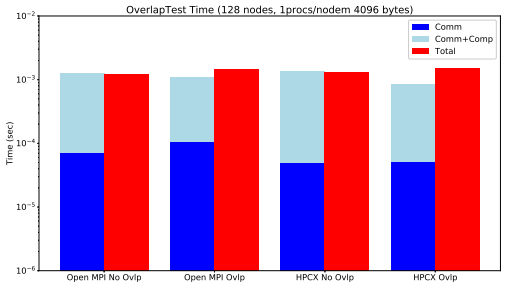

図 12　Overlap Test 実行時間 （1 プロセス/ノード 4096 バイト）

4.4　PETSc における集団通信の隠蔽効果

PETSc のバージョン 3.9.2 に含まれる通信隠蔽型反復解法アルゴリズム PIPE CG について、SHARP による集約操作の隠蔽効果を計測した。前処理には Jacobi を選択した。計測には、PETSc のソースコードに含まれているチュートリアル用プログラムのうち ex3.c を用いた。また、比較のため、通信隠蔽を行わない CG 法による計算時間も計測した。対象の行列サイズは 1536 と 3072 を用いた。計測結果を図 13 から図 16 に示す。

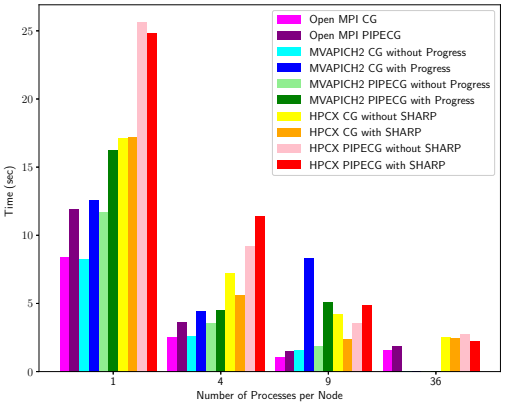

図 13　PETSc ex3 実行時間 （16 ノード 行列サイズ 1536）

今回の計測では、ほとんどの場合で、通信隠蔽を行わない CG 法を用いた方が、PIPE CG 法を用いた場合よりも高い性能が得られた。また、特にノード数が多くなると、Open MPI を用いた場合に、他の MPI ライブラリよりも大幅に高い性能が得られることが分かった。

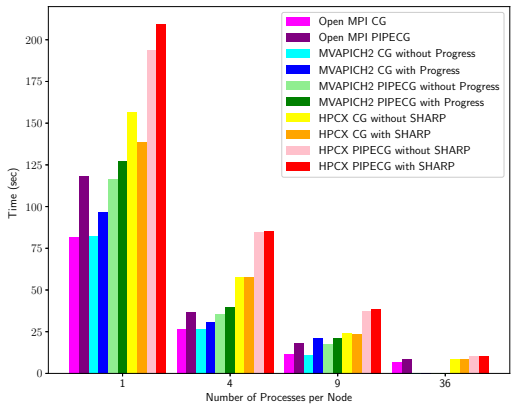

図 14　PETSc ex3 実行時間 （16 ノード 行列サイズ 3072）

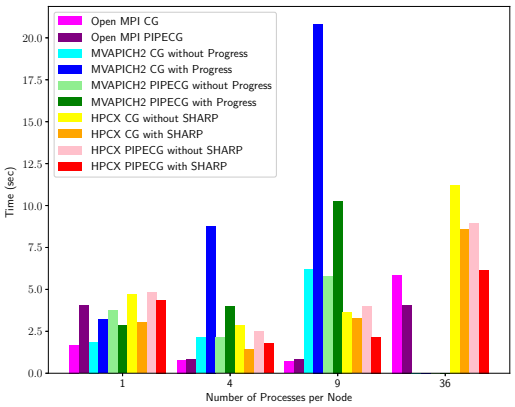

図 15　PETSc ex3 実行時間 （128 ノード 行列サイズ 1536）

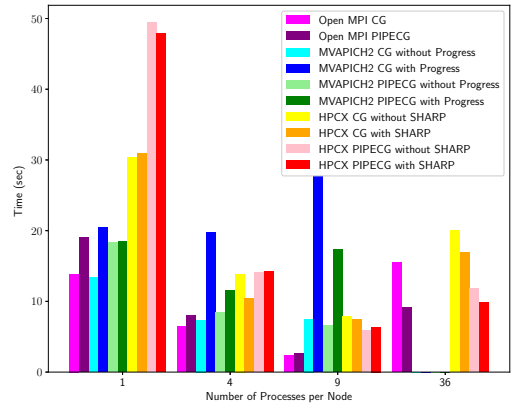

図 16　PETSc ex3 実行時間 （128 ノード 行列サイズ 3072）

5　まとめ

一対一通信、および集団通信を対象に、通信隠蔽の効果を、並列計算プログラムを用いて検証した。一対一通信については、適切な手段を選択することで、通信隠蔽効果が得られることが確認できた。一方、集団通信については、今回の計測では、ほとんどの場合で通信隠蔽効果は見られなかった。ただし、Mellanox 社の SHARP プロトコルによる `MPI_Allreduce` の性能向上は確認できた。集団通信の通信隠蔽効果を得る手段については、今後も調査を継続する予定である。

謝辞

本研究の実験には、九州大学情報基盤研究開発センターのスーパーコンピュータシステム ITO を利用した。

参考文献

[1] MPI-3.0 Draft. `https://www.mpi-forum.org/`

[2] Sumimoto, S., Ajima, Y., Saga, K., Nose, T., Shida, N. and Nanri, T., “ACP: Advanced Communication Primitives for Exa-scale Systems,” 11th International Meeting High Performance Computing for Computational Science, poster, 2014.

[3] MPICH. `http://mvapich.cse.ohio-state.edu/`

[4] MVAPICH2. `https://www.open-mpi.org/`

[5] Open MPI. `https://www.open-mpi.org/`

[6] Intel MPI Library. `https://software.intel.com/en-us/intel-mpi-library`

[7] Mellanox HPC-X `http://www.mellanox.com/page/hpcx_overview`

[8] Richard L. Graham, Devendar Bureddy, Pak Lui, Hal Rosenstock, Gilad Shainer, Gil Bloch, Dror Goldenerg, Mike Dubman, Sasha Kotchubievsky, Vladimir Koushnir, Lion Levi, Alex Margolin, Tamir Ronen, Alexander Shpiner, Oded Wertheim and Eitan Zahavi. Scalable Hierarchical Aggregation Protocol (SHArP): A Hardware Architecture for Efficient Data Reduction. *Proceedings of the First Workshop on Optimization of Communication in HPC*, pp. 1–10, 2016.

[9] PETSc. `https://www.mcs.anl.gov/petsc/`