Vol.2018-HPC-165 No.12
2018/7/30

Mellanox社のスイッチ装置への集団通信オフロード機能による集団通信隠蔽効果の調査

南里 豪志[1,a)]　大島 聡史[1,b)]　小野 謙二[1,c)]

概要：プロセス並列プログラムにおいて、複数のプロセスのデータを集約する集団通信は、様々な並列プログラムで多用されているため、計算機の大規模化に向けた性能向上が求められている。本稿では、この集約操作の集団通信をインターコネクトネットワーク上で行える集団通信オフロード機能を備えたMellanox社のスイッチ装置を対象に、この集団通信を計算と並行して行うことによる通信時間隠蔽の効果を調査した。調査には、既存のベンチマークプログラムOSU Micro Benchmarksだけでなく、本研究グループが作成したOverlap Testプログラム、および、反復解法ライブラリPETScにおける通信隠蔽アルゴリズムを用いた。調査の結果、メッセージサイズやノード数、ノード内プロセス数等の条件によって、集団通信オフロード機能により通信時間や全体の実行時間が短縮できる場合があることが確認できた。ただし、この集団通信オフロード機能は、集団通信自体の高速化にも寄与することから、この効果が通信隠蔽によるものか否かについては、さらなる調査が必要であることも分かった。

Study on the Effect of Overlapping with Collective Communication Offloading Facility of Mellanox Switch

NANRI Takeshi[1,a)]　OHSHIMA Satoshi[1,b)]　ONO Kenji[1,c)]

1. 背景

プロセス並列プログラムにおいて集団通信は、プロセスグループ内でデータの分配や集約、交換等の操作を簡潔なインターフェースで実現出来る重要な機能である。一方、集団通信に要する時間はプロセス数に応じて増加するため、スケーラブルな性能向上を阻害する。そこで、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) では、2012年に制定したMPI-3.0規格において、集団通信の時間を計算時間で隠蔽するための手段として非ブロッキング集団通信関数を定義した [1]。しかし、従来の非ブロッキング集団通信の実装では、隠蔽効果を得るために集団通信アルゴリズムを推進するスレッドを別途起動してCPUコアを割り当てる必要があるため、性能面での課題があり、実用性は限定的であった **?**。

一方、Mellanox社は、Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) と呼ばれる、インターコネクトネットワークに集約操作をオフロードするためのプロトコルを制定し、このプロトコルを実装したスイッチを開発した [2]。このプロトコルは、MPI_Allreduce等、並列プログラムで特に頻繁に利用されている集約操作の集団通信について、抽象的な木構造上でデータを転送させながら集約操作を適用するものである。これにより、このプロトコルを実装したインターコネクトでは、従来の計算ホスト上のCPUを用いた実装に比べ、集約操作の集団通信におけるデータ転送量を削減できるうえ、CPUでの待ち時間が不要となるため、通信時間を短縮できることが、報告されている [2], [3]。また、このプロトコルでは、集団通信アルゴリズムの推進にCPUコアがほとんど不要となるため、非ブロッキング集団通信時の通信隠蔽効果も期待できる。しかし、その効果については、まだ調査や報告がなさ

[1] 九州大学
Research Institute for Information Technology, Kyushu University
[a)] nanri.takeshi.995@m.kyushu-u.ac.jp
[b)] ohshima@cc.kyushu-u.ac.jp
[c)] keno@cc.kyushu-u.ac.jp

れていない。

そこで本稿では、この Mellanox 社の集団通信オフロードプロトコル上に構築された MPI ライブラリを用いて、非ブロッキング集団通信による通信隠蔽効果を調査する。調査には、通信と計算を並行して実行するプログラムが必要である。本稿では、まず、MPI ライブラリの標準的なベンチマークプログラム群である OSU Micro Benchmark の MPI_Iallgather 関数用ベンチマークプログラムを MPI_Iallreduce 関数用に書き換えたものを用いる。このベンチマークプログラムは実行時に計測した通信速度に合わせて計算量を自動調整するため、オフロードプロトコル使用の有無による違い等、他の実装との比較が困難である。そこで、文献?で筆者らが作成した、一定量の通信と計算を並行して実行するベンチマークプログラム Overlap Test を用いて、他の MPI ライブラリを用いた場合との性能を比較する。さらに、実アプリケーションでの効果として、数値計算ライブラリ PETSc 中の通信隠蔽型反復ソルバ PIPE CG を用いて、オフロードプロトコル使用の有無による実行時間への影響を調査する。

本稿の主な寄与は、以下の通りである。

- Mellanox 社のスイッチへのオフロードプロトコルによる非ブロッキング集団通信の素性能計測
- 通信と計算の並行実行ベンチマークプログラム Overlap Test における、オフロードプロトコル使用の有無による性能への影響の調査
- 通信隠蔽型反復ソルバにおける通信隠蔽効果の検証

2. 非ブロッキング集団通信の実装手段

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 ?、MVAPICH2 ?、Open MPI ? 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 ?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、Open MPI をベースに、集約型の集団通信の実装としてオフロードプロトコルを用いたものを選択できる MPI ライブラリが用意されている [4]。

2.2 非ブロッキング集団通信のアルゴリズム推進手法

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。この、アルゴリズム推進手法は、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。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 ?

をコンポーネントとして追加することにより、この推進手法を実装している。この推進手法は、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。ただし、本稿執筆時点の 2018 年 7 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 は、LibNBC とは異なる推進手法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を、InfiniBand 上でも選択可能である。

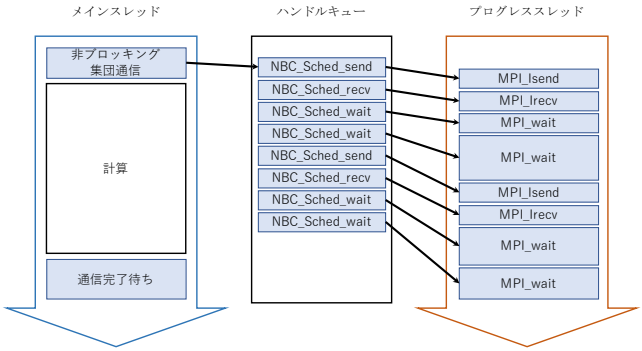

図 1 LibNBC におけるプログレススレッドによる推進機構

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、前述の SHARP プロトコルの他に、集団通信アルゴリズムを NIC で処理する CORE-Direct 機能も提供している。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 [4]、等がある。

3. 集約操作のオフロードプロトコル SHARP

3.1 SHARP の概要

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。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。

このプロトコルが適用できる条件は、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こととなっており、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。

3.2 SHARP における集約操作

SHARP では、スイッチ上、もしくは計算ホスト上に配置した Aggregation Node と呼ばれる抽象的なノード群を木構造に接続した Aggregation Tree と呼ばれる構造で集約操作を抽象化する [2]。集約操作に参加する各プロセスは、この木構造の葉の Aggregation Node のいずれか一つに接続しており、そのノードに対して集約操作を適用するデータを送信することで操作を開始する。各ノードが、プロセスもしくは子ノードからのデータに対する集約演算の適用と、結果の親ノードの転送を、ルート位置のノードまで繰り返すことにより、全データに対する集約操作の結果が得られる。その後、各ノードが木構造を逆にたどって結果を伝搬させ、参加した全プロセスに結果が到達すると、集約操作が完了する。

SHARP における Aggregation Node の実体は、計算ホスト上のプロセス、スイッチ上のプロセス、もしくはスイッチ上のハードウェアで実現することが出来る。Mellanox 社の InfiniBand スイッチ製品のうち Switch-IB 2 には、SHARP 用ハードウェアが実装されており、Aggregation Node を動作させることが出来る。Aggregation Node の配置や相互の接続は、システム上のプロセスの配置や物理的な接続に応じて自動的に決定される。

一方、SHARP における Aggregation Tree は、システム上に複数作成することが出来る。また、各 Aggregation Node は複数の Aggregation Tree に参加出来る。さらに、複数の Aggregation Tree で集約操作を同時に動作させることが出来る。Aggregation Tree は、最大 256 バイトのペイロードを持つメッセージを用いて、集約操作を実行する。256 バイトを超えるサイズの集約操作が必要な場合、複数のメッセージに分けて個別に実行する。これらのメッセージは、同じ Aggregation Tree を使ってパイプライン処理させるか、もしくは複数の Aggregation Tree で同時に進行させることが出来る。

なお、各 Aggregation Node では、全ての子ノードもしくはプロセスからのデータが揃うまで待ってから、あらかじめ決められた順に集約演算を適用する。これにより、同じ Aggregation Tree で実行する集約操作において、計算

表 1　SHARP API の主な関数

関数名	機能
sharp_coll_init, _finalize	SHARP 用データ構造の初期化、破棄
sharp_coll_do_allreduce	ブロッキング集約操作
sharp_coll_do_allreduce_nb	非ブロッキング集約操作
sharp_coll_comm_init, _destroy	集約操作に参加するプロセスグループ作成、破棄
sharp_coll_wait	非ブロッキング集約操作の完了待ち

結果の再現性を保証する。

3.3 SHARP API

SHARP で集約操作を行うプログラムを作成するためのインタフェースとして、SHARP API が用意されている。この API では、SHARP で行う集約操作に関するデータ構造と、いくつかの関数が定義されている。SHARP API の主な関数を表 1 に示す。

3.4 SHARP の利用を選択可能な MPI ライブラリ

本稿執筆時点で利用可能な MPI ライブラリのうち、SHARP を用いた MPI_Iallreduce の実装を選択可能なものは、Mellanox 社のツールキット HPC-X に含まれる MPI ライブラリ [4], [6] と、Ohio State University で開発されている MVAPICH2 のバージョン 2.3 release candidate 2 である。MVAPICH2 は、ソースが公開されているものの、release candidate であるため、本稿では HPC-X の MPI ライブラリを用いて非ブロッキング集団通信の通信隠蔽効果を調査する。

4. 非ブロッキング集約操作の通信隠蔽効果計測プログラム

本稿では、SHARP による通信隠蔽効果について、主に以下の視点で調査する。

- 通信時間のうち隠蔽できた時間の比率
- 他の実装との隠蔽効果の比較
- 実アプリケーション性能への影響

一つのプログラムで、これらを全て計測し評価することは困難である。そこで本稿では、本節で説明する 3 本のプログラムを用いた計測により、SHARP の通信隠蔽効果を調査する。

4.1 OSU Micro Benchmarks

OSU Micro Benchmarks [7] は、MPI や OpenSHMEM 等の並列プログラミングモデルの基本的な機能の性能を測定するベンチマークプログラム群であり、オハイオ州立大学で開発されている。このうち非ブロッキング集団通信用のベンチマークプログラムでは、通信と計算を同時に実行して、隠蔽できた通信時間の比率を計測する。ただし、本稿執筆時点の OSU Micro Benchmarks の最新バージョンである 5.4.2 には、MPI_Iallreduce のベンチマークプログラムが含まれていない。そこで本稿では、MPI_Iallgather のベンチマークプログラムを、MPI_Iallreduce を呼び出すよう書き換えたものを用いる。

図 2 に、OSU Micro Benchmarks の非ブロッキング集団通信ベンチマークプログラムにおける通信隠蔽率計測の概要を示す。このプログラムでは、最初に非ブロッキング集団通信の開始から完了までの時間（秒）を計測し、変数 Tcomm に格納する。その後、MPI_Iallreduce 関数で改めて非ブロッキング集団通信を開始し、Tcomm 秒が経過するまで、dummy_comp() という関数を繰り返し呼び出し、計算する。このようにして、十分に計算時間を費やした後に、MPI_Wait 関数で非ブロッキング集団通信の完了を待つ。この手順で得られた計測結果から、非ブロッキング通信で隠蔽できた通信時間と Tcomm の比率 overlap を、通信隠蔽率として出力する。

```
ts = MPI_Wtime();
MPI_Iallreduce();
MPI_Wait();
Tcomm = MPI_Wtime() - ts;

ts = MPI_Wtime();
MPI_Iallreduce();
tcs = MPI_Wtime();
while (MPI_Wtime() - tcs < Tcomm)
    dummy_comp();
Tcomp = MPI_Wtime() - tcs;
MPI_Wait();
Tall = MPI_Wtime() - ts;

overlap = 100 - ((Tall - Tcomp) / Tcomm) * 100;
```

図 2　OSU Micro Benchmarks の非ブロッキング集団通信ベンチマークプログラムにおける通信隠蔽率計測の概要

このプログラムは、自動的に計算量を調整するため容易に通信隠蔽率を計測できる。しかし、計算量が実行のたびに変動するため、様々な非ブロッキング集団通信の実装について優劣を比較する手段としては適当ではない。また、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では、計算スレッドとプログレススレッドの間での CPU コアの競合が性能に影響するのに対し、このベンチマークプログラムの計算プログラムはスレッド並列化されていないため、その影響を加味した評価が出来ない。

4.2 Overlap Test

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。図 3 にプログラムの概要を示す。このプログラムは、計算する行列サイズ N、および集団通信の通信量 M を実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 Tnoovlp、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 Tovlp をそれぞれ計測する。

```
get_param(&M, &N);
MPI_Comm_size(&procs);

ts = MPI_Wtime();
MPI_Allreduce(M);
do_comp(N, procs);
Tnoovlp = MPI_Wtime() - ts;

ts = MPI_Wtime();
MPI_Iallreduce(M);
do_comp(N, procs);
MPI_Wait();
Tovlp = MPI_Wtime() - ts;
```

図 3 Overlap Test プログラム)

図 3 から呼ばれる計算関数 do_comp_のプログラムを図 4 に示す。計算は、最外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。

```
void do_comp(int N, int procs)
{
  int nn = N / procs;
#pragma omp parallel for private(j,k) schedule(static)
  for (i = 0; i < nn; i++)
    for (k = 0; k < N; k++)
      for (j = 0; j < N; j++)
        c[i*N+j] += a[i*N+k] * b[k*N+j];
}
```

図 4 do_comp 関数

このプログラムは、通信量と計算量を利用者が指示する必要があるため、OSU Micro Benchmarks よりも不便である。しかし、同じ条件で複数の MPI ライブラリの実装を比較し、優劣を議論することが可能である。なお、行列サイズ N がプロセス数とスレッド数の積よりも小さい場合、並列度に応じた並列化効果が得られない。今回の計測でも、通信量に合わせて N を決めたため、プロセス数やスレッド数が大きくなると、計算の並列化効果は得られなくなった。しかし、今回の目的である複数の MPI ライブラリでの通信隠蔽効果の比較において、計算の並列化効果は重要ではないので、そのまま計測した。

4.3 PETSc

```
do {                              b = beta/betaold;
ierr = VecAYPX(P,b,Z);              /* p <- z + b* p */
dpiold = dpi;  ierr = KSP_MatMult(ksp,Amat,P,W); /* w <- Ap
ierr = VecXDot(P,W,&dpi);           /* dpi <- p'w    */
KSPCheckDot(ksp,dpi);             betaold = beta;
a = beta/dpi;                       /* a = beta/p'w  */
ierr = VecAXPY(X,a,P);              /* x <- x + ap   */
ierr = VecAXPY(R,-a,W);             /* r <- r - aw   */
ierr = KSP_PCApply(ksp,R,Z);        /* z <- Br       */
ierr = VecXDot(Z,R,&beta);          /* beta <- r'*z  */
KSPCheckDot(ksp,beta);  dp = PetscSqrtReal(PetscAbsScalar(b
ksp->rnorm = dp;  ierr = (*ksp->converged)(ksp,i+1,dp,&ksp-
if (ksp->reason) break;
ierr = VecXDot(Z,R,&beta);          /* beta <- z'*r  */
KSPCheckDot(ksp,beta);                        i++;
} while (i<ksp->max_it);
```

図 5 PETSc の CG 法コード

5. 性能評価

5.1 実験環境

5.1.1 ベンチマークプログラム

図 3 に示すベンチマークプログラムを利用し、非ブロッキング集団通信による並列プログラムの性能への影響を評価する。ベンチマークプログラムは C 言語で記述しており、MPI および OpenMP で並列化している。なお、プログラム中では、ウォームアップ処理として、プログラム開始直後に計測対象の通信関数を数回ずつ実行する。また、Tcomm、Tnbcomm、Tcomp、Tblock、Tovlp は、それぞれ 20 回ずつ連続して実行した平均値を測定結果とする。

また、計測対象の集団通信関数として図 3 に示した Alltoall 関数を用いるものの他に、Allreduce 関数を用いるものも用意し、計測する。いずれも、M 要素の倍精度実数をメッセージサイズとして指定する。また、Allreduce 関数では、集約操作として MPI_SUM を指定する。

謝辞

参考文献

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

```
do {                    VecNormBegin(R,NORM_2,&dp);
VecDotBegin(R,U,&gamma);
VecDotBegin(W,U,&delta);
PetscCommSplitReductionBegin(R);
KSP_PCApply(ksp,W,M);              /* m <- Bw */
KSP_MatMult(ksp,Amat,M,N);         /* n <- Am */
VecNormEnd(R,NORM_2,&dp);
VecDotEnd(R,U,&gamma);    VecDotEnd(W,U,&delta);
dp = PetscSqrtReal(PetscAbsScalar(gamma));
ksp->rnorm = dp;   (*ksp->converged)(ksp,i,dp,&ksp->rea
if (ksp->reason) break;
beta  = gamma / gammaold;
alpha = gamma / (delta - beta / alpha * gamma);
VecAYPX(Z,beta,N);    /* z <- n + beta * z */
VecAYPX(Q,beta,M);    /* q <- m + beta * q */
VecAYPX(P,beta,U);    /* p <- u + beta * p */
VecAYPX(S,beta,W);    /* s <- w + beta * s */
VecAXPY(X, alpha,P); /* x <- x + alpha * p */
VecAXPY(U,-alpha,Q); /* u <- u - alpha * q */
VecAXPY(W,-alpha,Z); /* w <- w - alpha * z */
VecAXPY(R,-alpha,S); /* r <- r - alpha * s */
gammaold = gamma;                              i++;
} while (i<ksp->max_it);
```

図 6　PETSc の CG 法コード

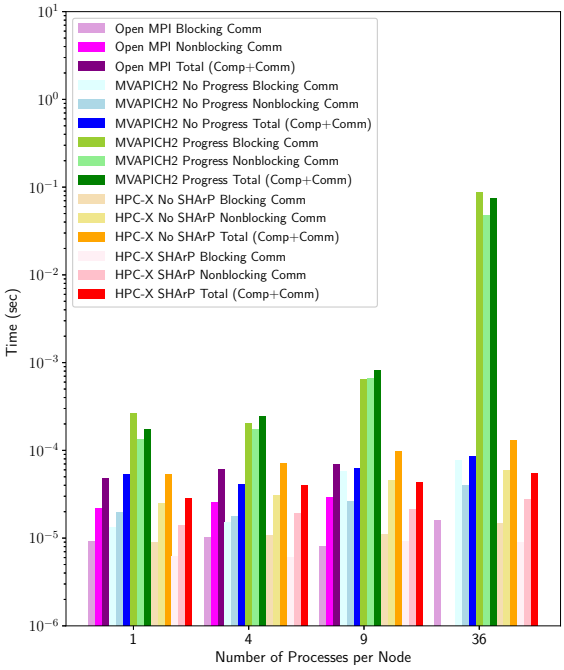

図 8　OSU Micro Benchmarks Allreduce 所要時間（32 ノード 512 バイト）

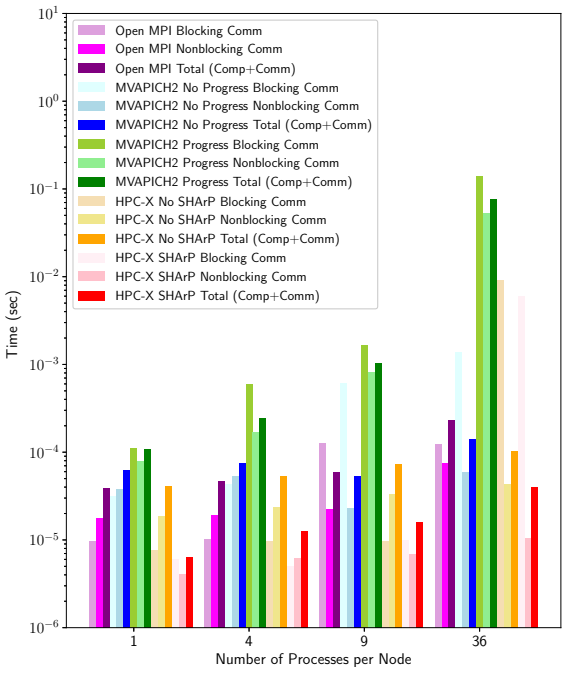

図 7　OSU Micro Benchmarks Allreduce 所要時間（32 ノード 16 バイト）

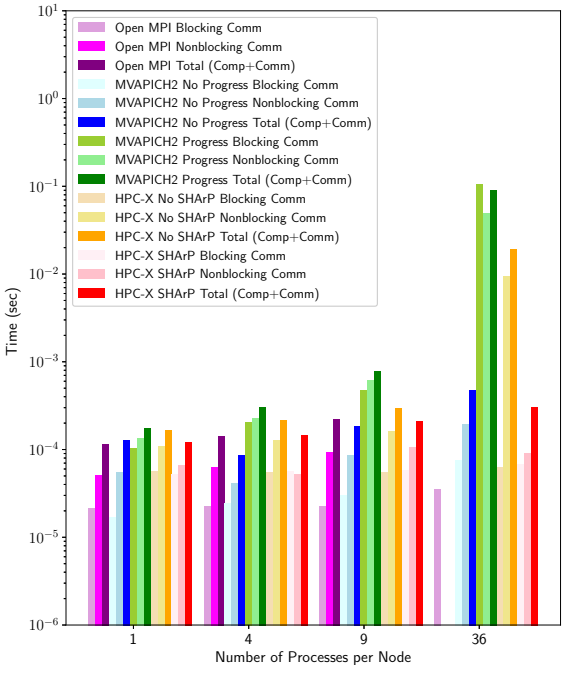

図 9　OSU Micro Benchmarks Allreduce 所要時間（32 ノード 8192 バイト）

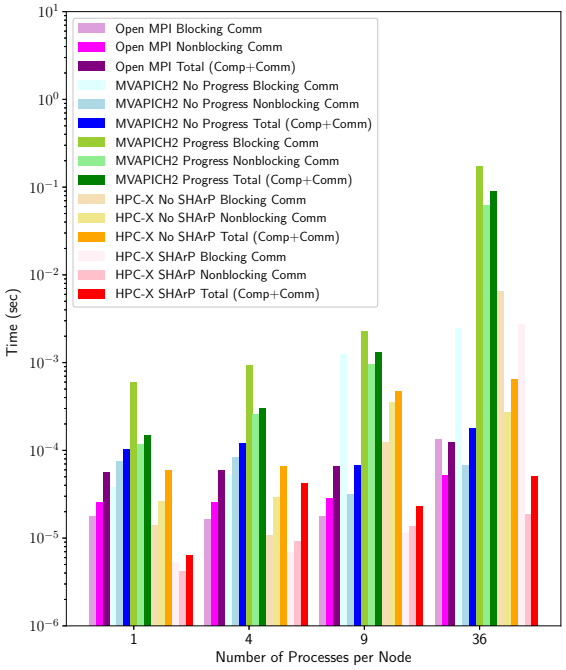

図 10 OSU Micro Benchmarks Allreduce 所要時間（128 ノード 16 バイト）

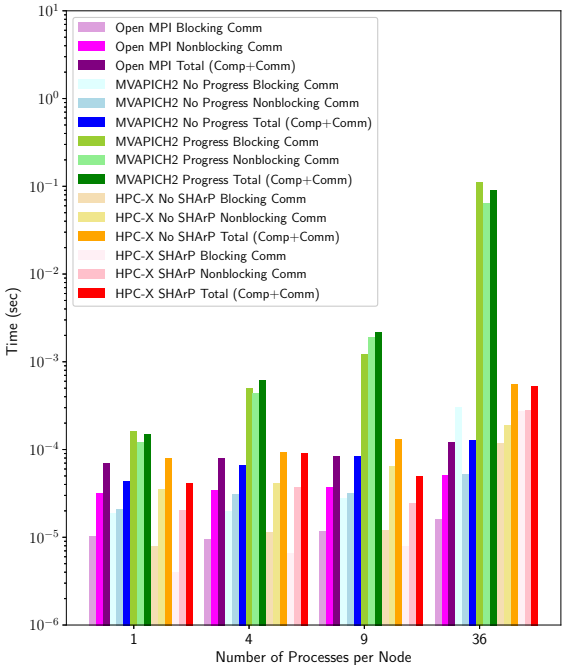

図 11 OSU Micro Benchmarks Allreduce 所要時間（128 ノード 8192 バイト）

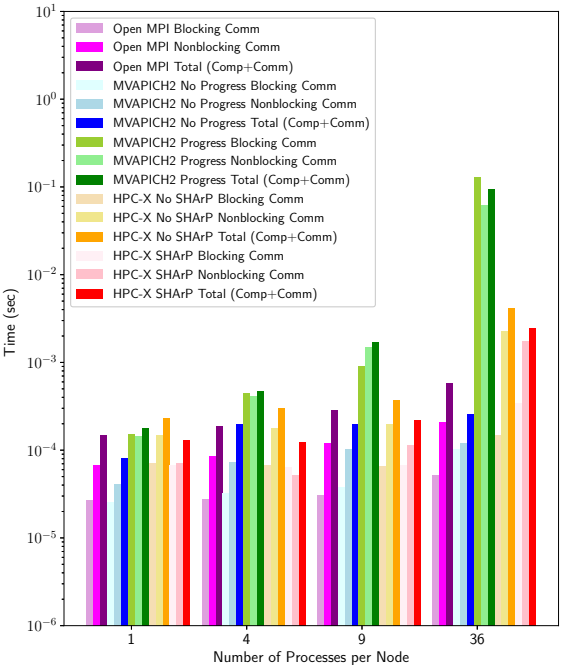

図 12 OSU Micro Benchmarks Allreduce 所要時間（128 ノード 8192 バイト）

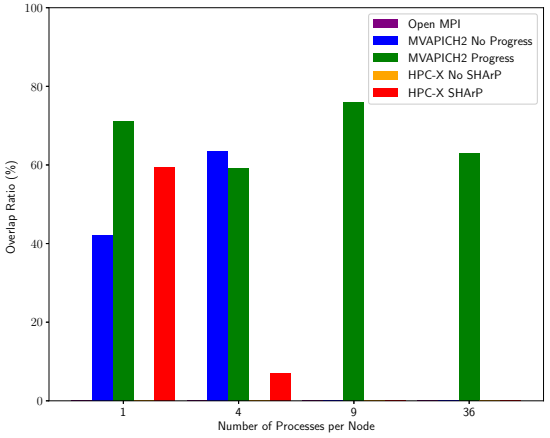

図 13 OSU Micro Benchmarks Allreduce 通信隠蔽率（32 ノード 16 バイト）

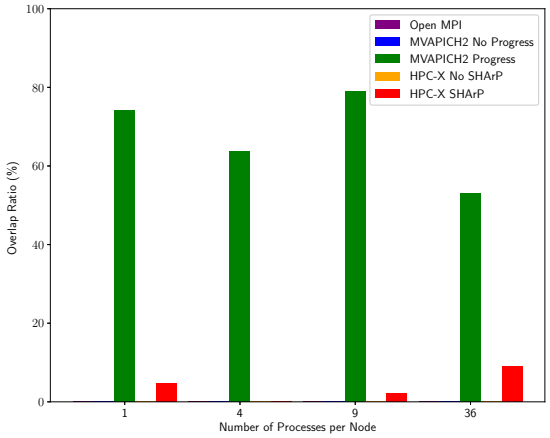

図 14 OSU Micro Benchmarks Allreduce 通信隠蔽率 （32 ノード 512 バイト）

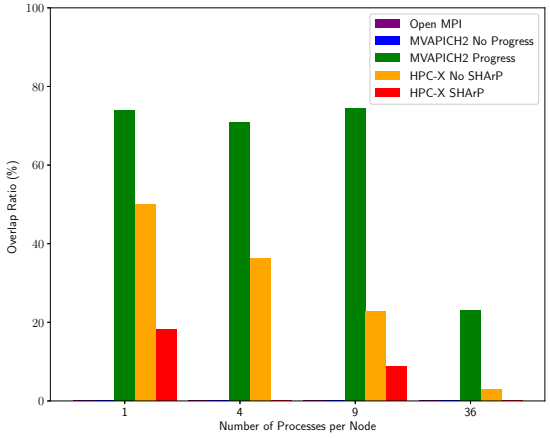

図 15 OSU Micro Benchmarks Allreduce 通信隠蔽率 （32 ノード 8192 バイト）

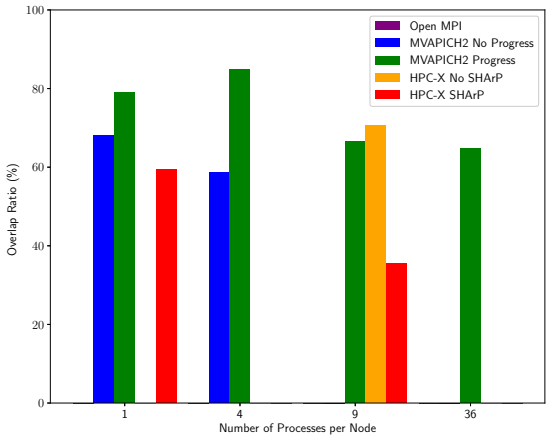

図 16 OSU Micro Benchmarks Allreduce 通信隠蔽率 （128 ノード 16 バイト）

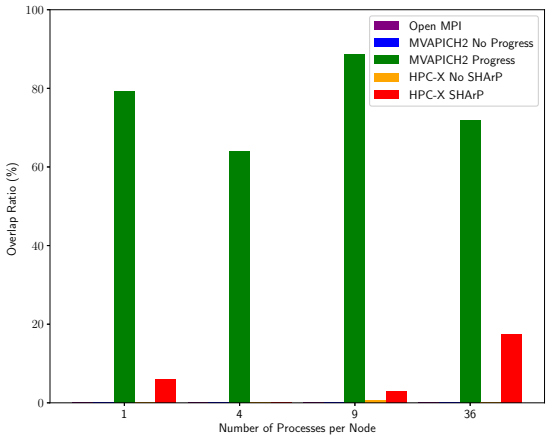

図 17 OSU Micro Benchmarks Allreduce 通信隠蔽率 （128 ノード 512 バイト）

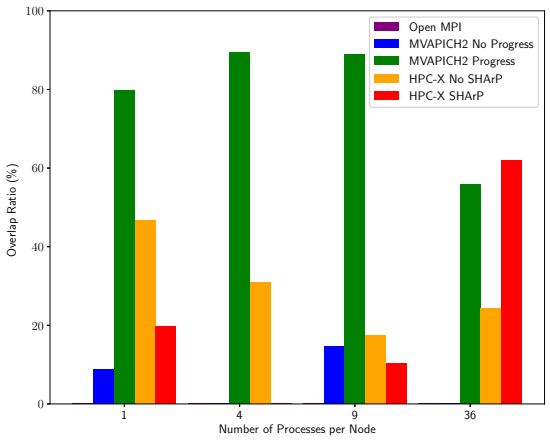

図 18 OSU Micro Benchmarks Allreduce 通信隠蔽率 （128 ノード 8192 バイト）

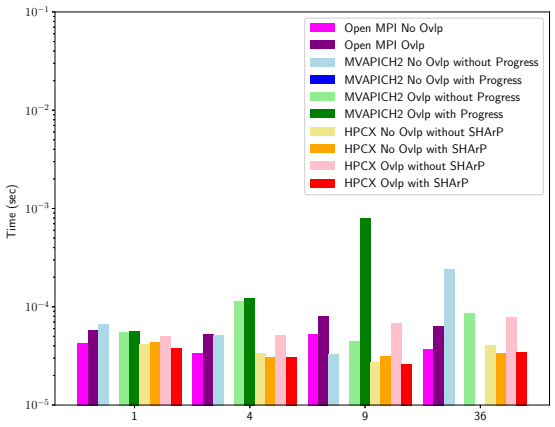

図 19 Overlap Test 実行時間 （32 ノード 16 バイト）

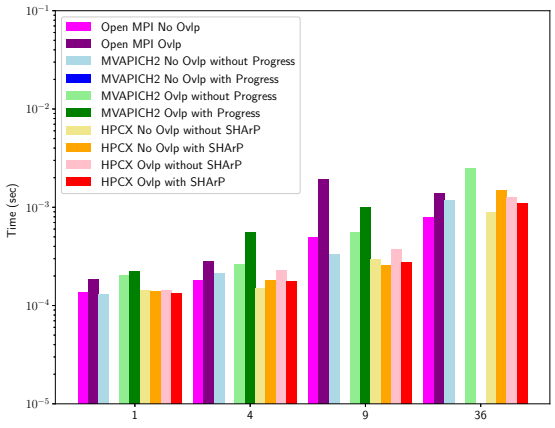

図 **20**　Overlap Test 実行時間 （32 ノード 4096 バイト）

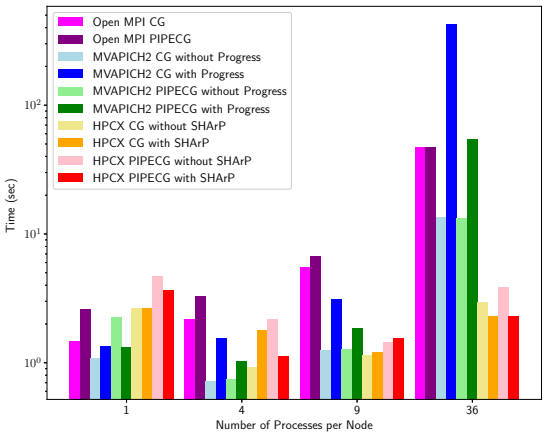

図 **23**　PETSc ex3 実行時間 （32 ノード 行列サイズ 1536）

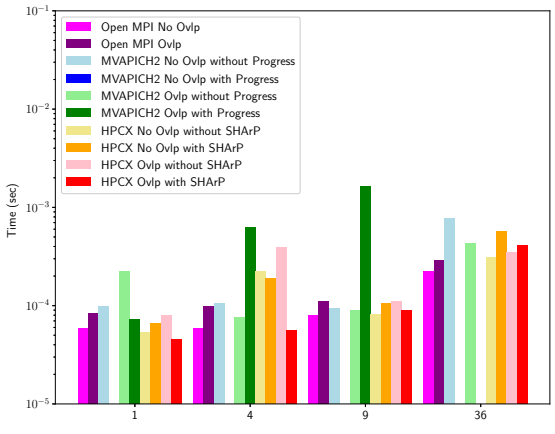

図 **21**　Overlap Test 実行時間 （128 ノード 16 バイト）

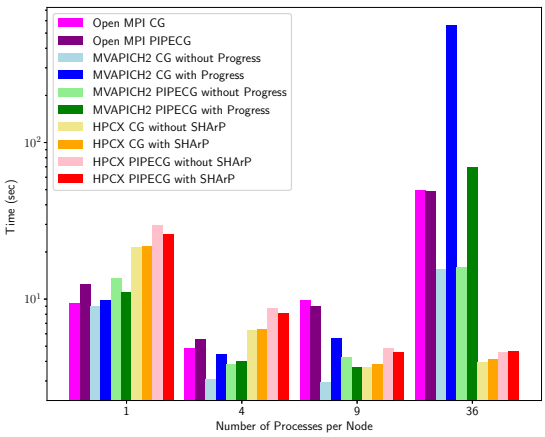

図 **24**　PETSc ex3 実行時間 （32 ノード 行列サイズ 3072）

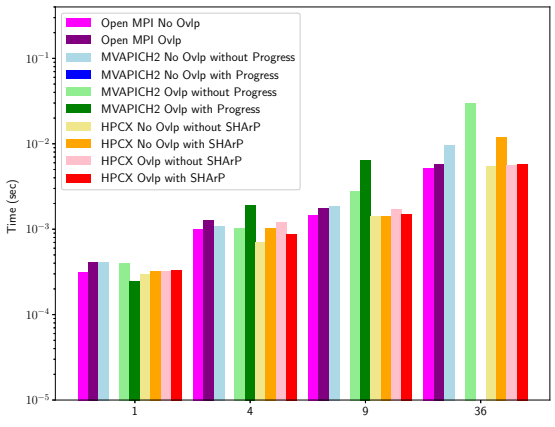

図 **22**　Overlap Test 実行時間 （128 ノード 4096 バイト）

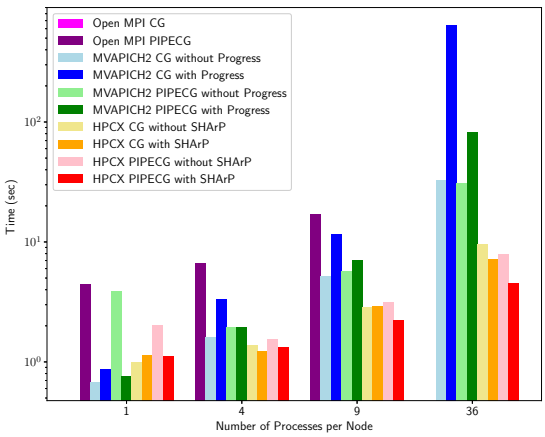

図 **25**　PETSc ex3 実行時間 （128 ノード 行列サイズ 1536）

[2] 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.
[3] Richard Graham, Gil Bloch, Devendar Bureddy, Gilad Shainer, Brian Smith, Towards A Data Centric System Architecture: SHARP *Supercomputing Frontiers and Innovations*, v. 4, n. 4, p. 4-16, nov. 2017. ISSN 2313-8734.
[4] Mellanox HPC-X `http://www.mellanox.com/page/hpcx_overview`
[5] SHARP API `http://www.mellanox.com/related-docs/prod_acceleration_software/Mellanox_SHARP_SW_API_Guide.pdf`
[6] HPC-X Software Toolkit Manual `http://www.mellanox.com/related-docs/prod_acceleration_software/HPC-X_Toolkit_User_Manual_v2.0.pdf`
[7] OSU Micro-Benchmarks. `http://mvapich.cse.ohio-state.edu/benchmarks/`

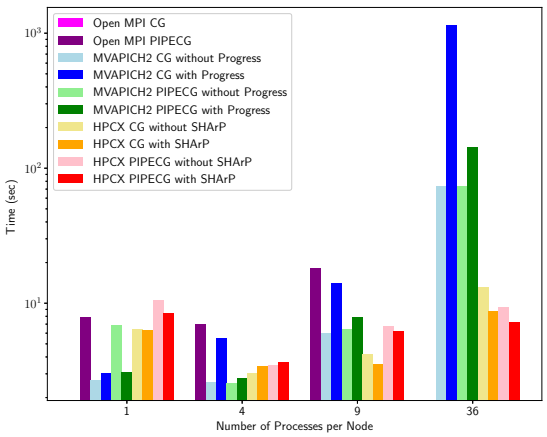

図 26 PETSc ex3 実行時間 （128 ノード 行列サイズ 3072）