Vol.2019-HPC-168 No.1
2019/3/5

マルチタスクPGASモデルをサポートするノード間軽量通信レイヤの検討

小田嶋 哲哉[1,a)]　森江 善之[1]　李 珍泌[1]　佐藤 三久[1]

概要：多くの PC クラスタやスーパーコンピュータでは，メニーコアプロセッサが広く利用されている．コア数の増加は，ループ文のワークシェアリングなどのデータ並列だけではなく，タスク並列でも性能向上が期待されている．我々は，並列分散環境において PGAS のリモートメモリアクセスによりデータを転送し，異なるノード間においてもデータの依存グラフに応じてタスクを実行するマルチタスク PGAS モデルを提案している．マルチタスク PGAS モデルではタスク対タスクの通信を行い，グローバルな同期を削減することで性能向上を図ることを目的としている．異なるノード間の複数のタスク間で同時にデータ通信・同期が発生するため，タスクを実行する複数のスレッド間で通信が必要である．しかし，ノード間通信における MPI は，マルチスレッド環境において著しく性能が悪化することが問題となっている．我々は，異なるノードの複数タスクの通信・同期は，チャネルを仮定したメッセージ通信よりも，メモリ上でデータ通信・同期を行う PGAS モデルとの親和性が高く，最小限の RDMA ベースの通信を行うことで，より高い通信性能を達成する事ができると考えている．本研究では，マルチタスク PGAS モデルを対象とした，低レベル通信 API を用いたノード間軽量 PGAS 通信レイヤを検討する．予備評価では InfiniBand Verbs を対象に，提案するノード間軽量 PGAS 通信レイヤにおいて，複数の Queue Pair を使用することで，マルチスレッド環境での通信性能の低下を抑えることができることがわかった．

1. はじめに

近年，High Performance Computing の分野では，多数のコアを搭載したメニーコアプロセッサが広く利用されている．コア数の増加により，大規模なループ文のワークシェアリングの性能だけではなく，実行の最小単位をタスクとして動的にコアに割り当てるタスク並列の性能向上も期待することができる．我々は，並列分散環境において，PGAS のリモートメモリアクセスによりデータを転送し，異なるノード間においてもデータの依存グラフに応じてタスクを実行するマルチタスク PGAS モデルを提案している．タスク並列は，データの依存関係によって連鎖的にタスクが実行されるモデルであるため，タスク間で多数のデータを通信でやり取りする必要がある．多数のタスクはスレッドという形で実行されており，ノードをまたぐタスク間では複数のスレッド間通信が発生している．MPI は，“MPI_THREAD_MULTIPLE” をサポートすることで，複数のスレッドがスレッドセーフに MPI 関数を実行することを保証しているが，複数のスレッドで通信を行うと非常に性能が悪くなるという問題がある．Balaji らの研究 [1] では，スレッド数が増加すると，粗粒土の lock により MPI の通信性能が低下していることを指摘している．通信のエンドポイントが複数のスレッドで共有されていると，それを奪い合うことで性能が低下すると考えられる．

本研究では，マルチタスクを使用する PGAS モデルを対象に，マルチスレッドによるノード間通信を効率的に行う軽量な通信レイヤを検討する．我々は，異なるノードの複数タスクの通信・同期は，チャネルを仮定したメッセージ通信よりも，メモリ上でデータ通信・同期を行う PGAS モデルとの親和性が高く，最小限の RDMA ベースの通信を行うことでより高い通信性能を達成する事ができると考えている．MPI は，集団通信や通信のキューイングを行うなど非常に高機能であり，必要不可欠ではあるが，一方で PGAS モデルでは必要がない機能も多い．本研究で提案する軽量通信レイヤは，InfiniBand [2] 向けの InfiniBand Verbs（以降「IBV」と略す）や Omni-Path Architecture（OPA） [3] 向けの PSM2（Performance Scaled Messaging 2） [4]，Tofu インターコネクト 2 や Tofu インターコネクト D [5] 向けの uTofu といった，アーキテクチャごとの低レベル通信 API を直接利用し，性能とプログラム可搬性の両立を図る．本稿では，IBV を対象に，マルチスレッドの通信性能を評価する．

1　理化学研究所 計算科学研究センター
a)　tetsuya.odajima@riken.jp

ⓒ 2019 Information Processing Society of Japan

表 1 評価環境

CPU	Intel Xeon E5-2670 2.6GHz 8 cores × 2 sockets
Main Memory	DDR3 1600MHz 32GB
Interconnection	InfiniBand: Mellanox ConnectX-3 Single-port FDR
OFED	MLNX_OFED_LINUX-4.4-2.0.7.0
OS	CentOS release 7.5
MPI	MVAPICH2 2.3 (MV2_ENABLE_AFFINITY=0, MV2_USE_AFFINITY=0)

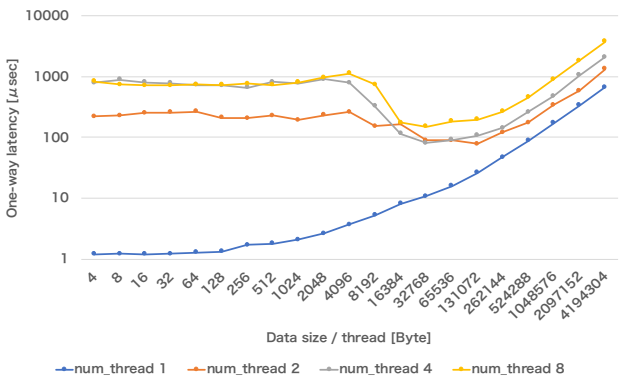

図 1 PingPong multi-thread latency: MPI_Send/Recv

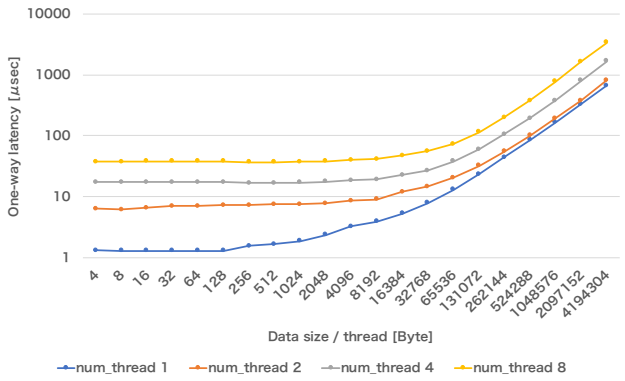

図 2 PingPong multi-thread latency: MPI_Put

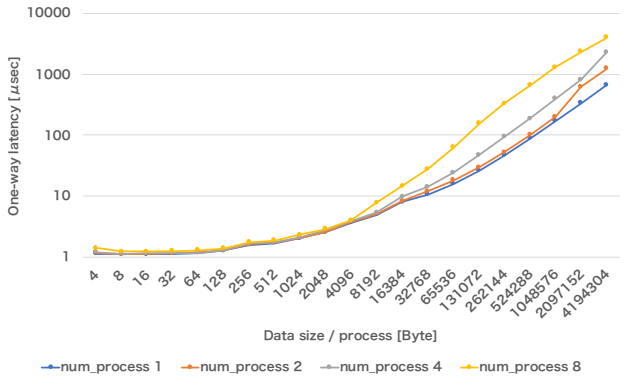

図 3 PingPong multi-process latency: MPI_Send/Recv

2. マルチスレッド環境における MPI の問題点

現在の MPI は，マルチスレッド環境において通信性能が悪化するという問題点が指摘されている．本節では，その予備評価として MPI を用いたマルチスレッド環境における基本的な通信性能を示す．評価には**表 1** に示すノード 2 台を用いて，スイッチ経由の InfiniBand で接続した．MPI として，オハイオ州立大学で開発されている MVAPICH2 を用いた．マルチスレッド環境で，スレッドが自由に MPI 関数を呼ぶためには MPI_THREAD_MULTIPLE を満たしていることが必要になる．MVAPICH2 では同時に，表 1 に示す環境変数を指定し，通信ライブラリ内の affinity 設定を無効にする必要がある．測定には，InfiniBand が接続されている側の CPU のみを使用した．

ノード内では OpenMP によるスレッド並列を行い，ノードごとに同じスレッド番号同士で MPI_Send/Recv による PingPong を行うプログラムを実行した．**図 1** に MPI_Send/Recv によるマルチスレッド PingPong ベンチマークの性能を示す．縦軸は片道のレイテンシでログスケールの数値，横軸はスレッドごとに通信したデータサイズを示している．凡例は，ノード内で展開しているスレッド数を示している．1 スレッド実行時はデータサイズによってレイテンシが徐々に大きくなっているが，2 スレッド以上では最小のデータサイズからすでにレイテンシが 1 スレッド実行の 100 倍以上大きくなっていることがわかる．ページサイズ（4KB）より大きくなると緩やかにレイテンシが低下して，おおよそ 128KB 以上からレイテンシが一定に増加している．

同様に，片方向通信である MPI_Put による PingPong のレイテンシを測定する．**図 2** に MPI_Put によるマルチスレッド PingPong ベンチマークの性能を示す．片方向通信を用いて PingPong を行う場合，Send/Recv とは異なり受信側はデータを受け取ったことを検出できない．MPI-3 の片方向通信は，MPI_Win_flush() 関数を使って最粒度の同期をとることが可能であるが，タグマッチの機能がないため特定のスレッド間で同期をとることは難しい．そのため本稿では，スレッド間の同期としてデータに加えてフラグを事前にノード間で取り決めたアドレスに書き込み，受信側はそのアドレスをメモリポーリングする手法を選択した．MPI_Send/Recv は通信と同時にスレッド間の同期をすることが可能であるが，片方向通信の場合は通信に加えて同期が必要であるため 1 スレッドのレイテンシは若干大きくなる．2 スレッド以上では図 1 と比較すると，レイテンシの増加が抑えられている事がわかる．しかし，通信するスレッド数に比例してレイテンシが増加しており，アプリケーションの性能低下に繋がる可能性がある．

一方，比較としてマルチスレッド実行ではなく，フラット MPI によるマルチプロセスの通信性能を測定する．**図 3** に MPI_Send/Recv によるマルチプロセス PingPong ベン

チマークの性能を示す．縦軸および横軸はこれまでのグラフと同様あるが，凡例は 1 ノード内に配置したプロセス数を示している．図 3 より，データサイズがページサイズ（4KB）に収まる場合，同時に通信するプロセスが増えたとしてもレイテンシは大きく低下しない事がわかる．

これらの結果から，マルチプロセスによる多数通信では通信性能の悪化がほとんど見られないが，マルチスレッドでは性能が極端に悪化することがわかる．マルチプロセス環境では，通信のエンドポイントがプロセスごとに独立しているため，同時に通信してもエンドポイントを奪い合うことがない．もちろん，ノードの出口である InfiniBand HCA が行うハードウェア処理は直列化されているが，性能悪化には寄与していない．一方，マルチスレッド環境ではメッセージ通信において，単一の通信エンドポイントを複数のスレッドが共有するため，スレッドセーフを維持するために atomic な操作が必要となり，性能低下の原因になったと考えられる．MPI では，物理的なエンドポイントの共有だけでなく，メッセージのタグマッチの処理なども排他制御する必要があり，大きなオーバーヘッドの要因となっていると考えられる．つまり，マルチスレッド環境において通信性能の低下を低減するためには，プロセス内に複数のエンドポイントが必要になると考えられる．

3. 関連研究

マルチスレッド環境下での通信性能が悪いということを示した文献がいくつかある．津金らの研究 [6] では，タスク並列のプログラムにおいて，MPI_THREAD_MULTIPLE の通信性能が低く，結果としてアプリケーションの性能が低くなってしまうことを指摘している．この問題を解決するために通信専用のスレッドを作成し，タスクが通信するときにグローバルキューに通信リクエストを追加し，通信専用スレッドがリクエストを実行するという最適化を行った．これによって，最適化前よりも約 10%の性能向上が得られている．

遠藤らの研究 [7] では，low-level communication library を実装し，通信専用スレッドとして Executor を動かし，通信リクエストをオフロードするという実装を行っている．IBV 実装では，複数の QP を使用することで高いメッセージレートを実現している．文献 [8] では，IBV だけでなく Tofu インターコネクト向けの実装も評価している．Tofu インターコネクトでは，“MPI_THREAD_MULTIPLE” をサポートしていないため通信を集約する必要がある．本研究では，送信するスレッドを固定しないためこのままでは Tofu インターコネクトに対応できない可能性もある．しかし，uTofu と呼ばれる次世代の低レベル通信ライブラリを使用することで，多数のスレッドが通信できる可能性があるため，今後，動作と実装を検討していく．

Meadows らの研究 [9] では，最新の Intel MPI ライブラリでサポートした OPA 向けの Multi-EP（Multiple Endpoints）を使用している．MPI とスレッド並列を組み合わせた環境において，スレッド数が増えても CCS-QCD アプリケーションの性能低下を抑え，かつ，ネットワークの BW を十分に使い切れていることを示している．Multi-EP を使用するには環境変数を設定する事が必要であるが，MPI のレイヤでマルチスレッドの通信性能低下を抑えていることはユーザビリティが非常に高い．しかしながら，現状，Multi-EP は OPA のみのサポートであるため，InfiniBand や Tofu インターコネクトでは効果を発揮することができない．我々が提案する軽量通信レイヤでは，OPA だけでなく InfiniBand や Tofu インターコネクトをサポートすることで，ポータビリティの向上を図る．

PGAS 向けの軽量通信ライブラリとして Lawrence Berkeley National Laboratory で開発されている GASNet [10] がある．UPC や Co-array Fortran，XcalableMP などが GASNet を通信ライブラリとして使用している．しかしながら，GASNet は MPI の通信と同時に使用することができないという制限がある．高速な片方向通信だけでなく，集団通信など MPI の機能を使うことが難しいということは，ユーザのプログラマビリティが低下してしまう．また，GASNet はマルチスレッドをサポートしていないため，本研究で扱うマルチタスクモデルでは使用することができない．現在，これらの問題を解決しうる次世代の通信ライブラリである GASNet-EX [11] が開発されている．これは，現在のメニーコアアーキテクチャに対応するために，マルチスレッド対応も改善されるということが報告されている．今日まで，実際に使用できる環境がないためマルチスレッド性能については未知数であるが，MPI との同時使用性を含め引き続き調査を継続していく．

UCX（Unified Communication X）[12] も通信ライブラリとして広く知られている．UCX は GASNet のような機能だけでなく，タグマッチなどより高級な機能を有している．一方，マルチタスクモデルを対象とする場合，そのようなリッチな機能は必要なく，より軽量な通信レイヤが必要な場合オーバヘッドになりうる可能性がある．

ComEX [13] は以前 ARMCI という名前で開発が行われていた GASNet に類似した通信ライブラリである．Active Message の機能は有してはいないが，Accumulate 通信をそのまま利用することができる．中尾らの研究 [14] では，PGAS 言語 XcalableMP の Coarray 通信部の実装に GASNet と ARMCI を用いて性能を比較している．これより，一般的な片方向通信は GASNet を用いる方が高速であることが示されており，現在の XcalableMP の Coarray 実装の基になっている．

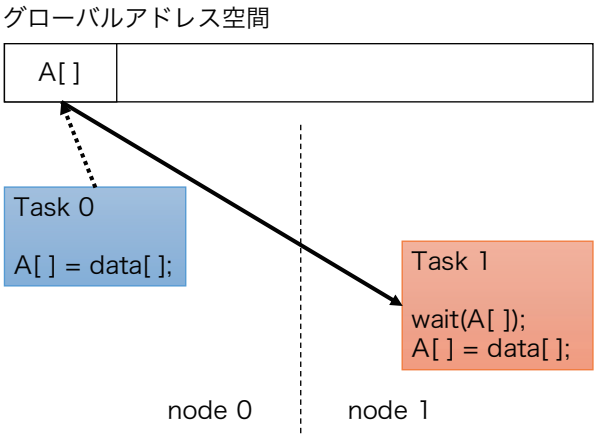

図 4 マルチタスク PGAS プログラミングの通信例

4. 軽量通信レイヤ Omni PGAS Communication Layer: OMPL

本節では，提案するマルチタスク PGAS モデルをサポートする軽量通信レイヤ Omni PGAS Communication Layer（以降「OMPL」と略す）の概要を示す．

4.1 マルチタスク PGAS プログラミングモデル

マルチタスク PGAS プログラミングにおける通信の一例を**図 4** に示す．これより，ノード 0 にある “Task0” がグローバルアドレス空間にある “A[ ]” に対してデータの代入を行っている．一方，ノード 1 にある “Task1” は，計算に必要な “A[ ]” が更新されるまで待つ必要があり，データが更新されたら計算を進めるという依存関係がある．プログラムにおいて，“Task0” の “A[ ]” への代入は，“Task1” から見ればノード間通信によるデータ書き込みである．マルチタスク PGAS プログラミングでは，このようにグローバルアドレスへの代入，またはグローバルアドレスからの代入によってデータの交換を行う．このデータのやり取りがノード間で発生した場合には，ノード間通信が発生するということになる．

マルチタスク PGAS プログラミングでは，グローバルアドレス空間へのデータの書き込み順序はユーザプログラムで保証する必要がある．つまり，図 4 における “A[ ]” は “Task0” がデータを更新し，それを “Task1” が計算に使うまで，他のタスクから “A[ ]” を更新してはならない依存関係を保証したタスクグラフを作成する必要があるということである．これより，通信レイヤから見れば，通信の順序関係は上位のユーザプログラムで保証されているため，通信順序保証のためにキューイングなどを行う必要はないということがわかる．同時に，メモリ上での通信・同期であるため，あるタスクから見て他のタスク全てに通信する可能性がありうるということである．

4.2 OMPL の API

OMPL は，PGAS のグローバルアドレス空間を使用したデータの通信を行うための軽量通信レイヤである．通信の順序保証などの制限は，上位のユーザプログラムで保証していることが前提であるため，API は非常に簡易にすることができる．以降，OMPL で定義する API を示す．

int ompl_init(), int ompl_finalize()

ompl_init() 関数は，OMPL で使用するグローバルアドレス空間をメモリにピンダウンや，通信のエンドポイントを確保しノード間で共有する．すべてのノードが同容量のアドレス空間を確保する．ompl_finalize() 関数は，使用したメモリ領域の開放などを行う．

int ompl_put(uint64_t src_addr, uint32_t target_rank, uin64_t dest_addr, size_t length)

ompl_put() 関数は，RDMA Write を用いて target_rank の dest_addr に src_addr のデータを length Byte だけ書き込む．

int ompl_get(uint64_t src_addr, uint32_t target_rank, uin64_t dest_addr, size_t length)

ompl_get() 関数は，RDMA Get を用いて target_rank の dest_addr から length Byte だけのデータを src_addr に読み込む．

int ompl_put_word(uint32_t target_rank, uint64_t flag_addr)

ompl_put_word() 関数は，RDMA Write を用いて 32Byte の値を書き込むフラグ操作に用いる．

int ompl_wait(uint64_t flag_addr)

ompl_wait() 関数は，ompl_put() 関数でフラグの書き込みが行われたアドレスをフラグが切り替わるまでブロッキングして待つ．フラグが切り替わっていない場合，システムコール sched_yield() で他のタスクへ切り替えを行い，フラグが切り替わるのを待つ．

int ompl_poll(uint64_t flag_addr)

ompl_poll() 関数は，ompl_put() 関数でフラグの書き込みが行われたアドレスを atomic 操作で確認する．

なお，以上の関数は現在検討中であるため，今後仕様が変更される可能性がある．

4.3 OMPL による PingPong ベンチマーク

ここでは 4.2 節で示した API を用いたプログラム例を示す．**図 5** に OMPL で記述した PingPong ベンチマークのサンプルプログラムを示す．これより，MPI の初期化後に OMPL の初期化を行っている．これは，ompl_init() 関数の内部では MPI を使って通信のエンドポイントなどの情報をノード間で共有するためである．OMPL では，データの転送とフラグの転送を分離している．タスク間のデータのやり取りにおいて複数のデータを転送する可能性があり，逐一通信の同期を行うとオーバヘッドが大きくなって

```
MPI_Init();
ompl_init();    // グローバルアドレスの確保
...
uint64_t src_addr, dest_addr, flag0_addr, flag1_addr;
// 各アドレス情報はノード間で共有する

if (rank == 0) {
  ompl_put(src_addr, target_rank, dest_addr, size); // dataの転送
  ompl_put_word(flag0_addr);      // flagの転送

  ompl_wait(flag1_addr);                  // flagアドレスのポーリング
} else {
  ompl_wait(flag0_addr);

  ompl_put(src_addr, target_rank, dest_addr, size);
  ompl_put_word(flag1_addr);
}
...
ompl_finalize();
MPI_Finalize();
```

図 5 OMPL による PingPong ベンチマークのプログラム例

しまうことが懸念される．そのため OMPL では，データをトリガとした同期ではなく，フラグを別に送るという方式を選択している．岩下らの研究 [15] では，多変数を片方向通信で転送し最後に最粒度の同期をすることで，同期オーバヘッドを削減しバンド幅が向上していることを示している．本提案実装でも同様の効果が得られると考えられる．最後に，ompl_finalize() 関数と MPI の完了関数を呼び出してプログラムを終了する．

5. OMPL の実装

本稿では，提案通信ライブラリ OMPL の IBV による実装について述べる．

5.1 InfiniBand Verbs

IBV については文献 [2], [16] に詳しいが，ここでは本稿を説明するために必要な概要を示す．IBV は InfiniBand 向けの低レベル通信 API である．InfiniBand の最大の特徴は，RDMA（Remote Data Memory Access）によるゼロコピー通信である．RDMA には，指定したアドレスにデータをコピーする RDMA Write と，逆に指定したアドレスからデータを読み込む RDMA Read がある．IBV は，RDMA Write だけでなく，データのコピーが完了したタイミングを通知する RDMA Write with Immediate というプロトコルも備えている

InfiniBand のトランスポート層は，Reliable/Unreliable と Connection/Diagram の組み合わせから得られる 4 種類のサービスを提供している．本実装では，再送処理などをサポートするコネクション型の RC（Reliable Connection）を使用する．RC では，通信のエンドポイントを 1 対 1 でコネクションを張り，その組を用いて通信を行おう．

QP (Queue Pair)

QP は InfiniBand における通信のエンドポイントである．QP は SQ (Send Queue) と RQ（Receive Queue）を持ち，それぞれ通信要求と受信要求を格納する FIFO（First-In First-Out）である．送信や受信を行う場合，SQ と RQ それぞれに Send WR（Worker Request）と Receive WR を投入する．WR には，通信に必要なアドレスやデータサイズ，通信先の QP 情報などが含まれている．このデータに基づき，InfiniBand のハードウェアが通信を実行する．また Receive に限って，複数の PQ 間で同じ RQ を共有する SRQ（Shared RQ）を設定することも可能である．

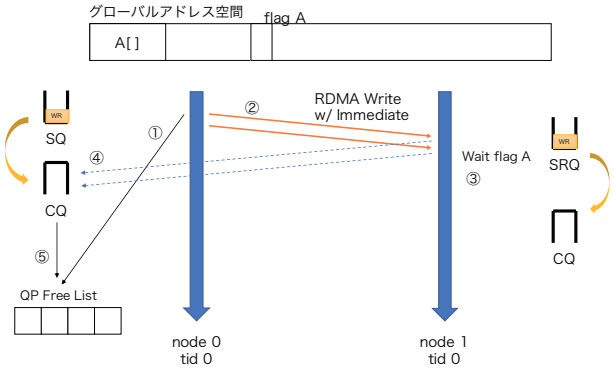

図 6 提案手法：マルチタスク PGAS 向け IBV 低レベル通信

CQ (Completion Queue)

InfiniBand は，通信の成功/失敗にかかわらず CQ と呼ばれる FIFO に結果を投入することができる．CQ は，Send および Receive の結果をまとめて 1 つの CQ に貯めることができるが，本実装では Send/Receive でそれぞれ独立した CQ を持つこととする．WR が完了したかどうかは，CQ をポーリングする関数を用いることで積まれていた結果を得ることができる．これによって，自分が通信を完了したか，または，データを受け取ったかを検出することができる．

5.2 IBV による OMPL の試作

本稿で述べる OMPL の IBV による試作実装では，従来の “MPI_THREAD_MULTIPLE” においてスレッドがエンドポイントを奪い合うことを改善したときの性能を検証する．IBV における通信のエンドポイントは QP であり，MPI はこれを複数のスレッドで共有しながら通信を行っている．例えば，QP があるプロセスに対して 1 組しかない場合，キューのポーリングを基本としている IBV では 1 つのスレッドが長時間 QP を専有してしまう可能性があり，この間，他のスレッドは通信を行うことができない．本稿では，すべてのスレッドが独立した QP を持つという前提で提案手法の IBV 実装の性能評価を行う．これによって，マルチスレッド環境における通信性能の低下は，エンドポイントのボトルネックが原因となっているかを確認する．本評価は OMPL による記述ではなく，IBV の低レベル通信 API で直接プログラムを記述して測定を行っている．**図 7** に，IBV 実装による PingPong のレイテンシを示す．予備評価と同じノードを 2 台用いる．縦軸は片道の

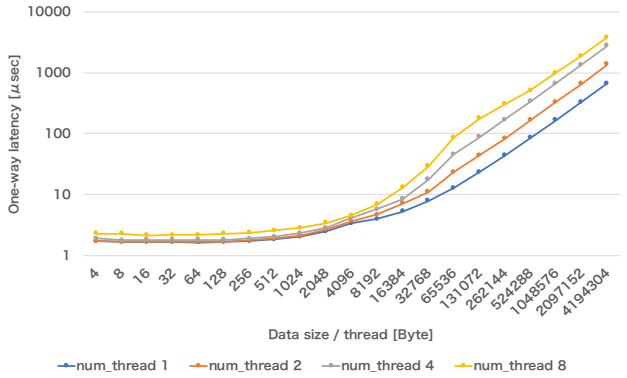

図 7 PingPong multi-thread latency: IBV RDMA Write with Immediate

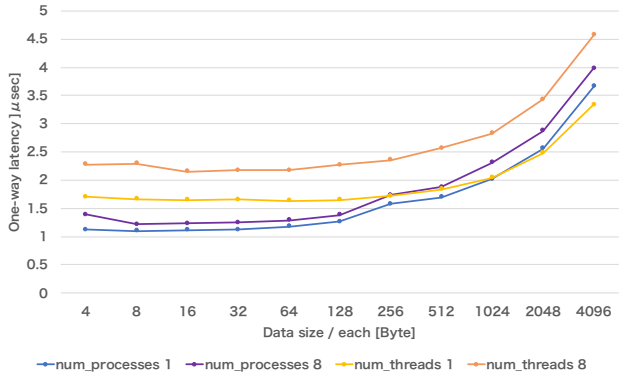

図 8 PingPong multi-process and multi-thread latency

レイテンシでログスケールの数値，横軸はスレッドごとに通信したデータサイズ，凡例はノード内で通信を行ったスレッド数を示している．各スレッドが QP を独立して持つという最もオーバヘッドが小さい環境では，図 1 や図 2 のように，スレッド数が 2 以上になったときに性能低下がほとんど見られない．本実装と同様に，図 2 の MPI を用いた片方向通信の実装と比較してもレイテンシが小さい事がわかる．つまり，MPI におけるマルチスレッドの性能低下は QP など単一のエンドポイントの奪い合いが原因であると言える．

次に，IBV 実装とマルチプロセス実行のレイテンシを比較する．**図 8** に，図 3 と図 7 から，通信の組が 1 と 8 の結果を抜粋して示す．縦軸がリニアスケールになっていることに注意されたい．これより，128KB まではプロセス並列のレイテンシが小さい．これは MPI が Send/Recv プロトコルを用いているため，小メッセージにおいてデータ転送と同時に同期処理を行うため，片方向通信よりも高速になったと考えている．スレッド並列では，8 スレッド実行は他よりもレイテンシが大きい．この理由に関しては現在検証中であるが，本稿における試作プログラムは未だ冗長なものが多く，オーバヘッドが大きい可能性がある．そのため，より最適化したコードによる検証を続けていく．

また，メモリ使用量の面から，スレッド数よりも少ない QP を用意し，それらをマルチスレッドで共有する必要があると考えている．本来であれば図 7 の結果にスレッドの切り替えなどのオーバヘッドが追加されることになるが，これについては現在実装中であり，今後検証と評価を行う．

5.3 QP を再利用する実装の検討

関連研究より，多くの通信ライブラリやアプリケーションの通信最適化において，通信専用のスレッドを作成し，計算スレッドから通信スレッドに通信を移譲する手法が取られている．通信スレッドだけがエンドポイントを使用することで，複数スレッドによるロックの問題を回避している．しかしながら，これらの通信専用スレッドは移譲された通信のキューイングなどを行う必要があるが，マルチタスク PGAS モデルではユーザプログラムで順序が保証されているため，この機能を削ることができると考えている．これによって，より軽量な通信による性能向上を期待している．本提案手法では，関連研究でいくつか挙げた通信専用スレッドのようなものは作成せず，各スレッドが各々通信を発生させる．そこで，スレッドごとに独立した QP を作成する，または，ノードごとに複数の QP を作成し，スレッドで共有することを考える．IBV の RC サービスでは，QP による 1 対 1 通信が必要になるが，全ノード対全ノードで通信が発生しうる PGAS プログラミングモデルでは，ノード数の 2 乗の QP を作成する必要があり，さらにスレッド数によって数倍から数十倍の QP を作成する必要がある．そのため，大規模並列環境ではメモリ消費量が非常に大きくなってしまうが，XRC（eXtensible Reliable Connection）や DC (Dynamically Connection) などのサービスを使用することで，これを抑える事ができると考えている．このような最適化は今後の課題としたい．本提案手法では，通知付きの RDMA Write である “RDMA Write with Immediate” プロトコルを用いる．RDMA Write ではない理由として，メモリの書き込み順序が保証されていないということにある．IBV では，同一の QP から発生する RDMA 通信はハードウェアが順序保証することができるが，リモート側でのメモリ書き込み順序の保証は仕様では決まっていない．そのため，複数のデータを Write して最後にフラグを Write すると，リモートでメモリにデータが書き込まれているうちにフラグが立ってしまう可能性がある．これを防ぐために，RDMA Write with Immediate を選択する．

図 6 に本提案手法におけるノード間のデータの書き込みと受信方法について示す．前提条件として，通信するノード間ではデータやフラグが書き込まれるアドレス（グローバルアドレス空間のオフセット）は既知である．

(1) データを書き込む場合，送信側は空いている QP を選ぶために QP Free List をロックし，空いている QP のフラグを立てる．この際，QP が埋まっている場合は，

システムコール sched_yield() を使って別のタスクの実行を行う．

(2) RDMA Write with Immediate プロトコルを使用して，データとフラグを対象のノードに書き込む．データは 1 つだけでなく，複数同時に書き込みを行うことも可能である．

(3) 受信側は，送信側よりフラグの書き込みをポーリングすることで，データの受取を検出する．この際，QP を巡回して適宜 CQ を dequeue してあふれることを防ぐ．

(4) 送信側は RDMA Write with Immediate のデータ到達の Ack を受け，その結果が CQ に積まれる．

(5) 通信が完了した後，QP Free List で使用した QP のフラグをもとに戻す．

以上が，本提案手法の IBV 実装における通信の流れである．

6. まとめ

本稿ではマルチタスク PGAS モデルを対象に，複数のスレッドが自由にノード間通信を行ったときにも性能低下が小さい軽量な通信レイヤを検討してきた．従来のアプリケーションで，“MPI_THREAD_MULTIPLE” による性能低下は，通信のエンドポイントをスレッドがロックしてしまい，効率的に通信ができなかったことが原因と考えられる．本提案手法では，関連研究で示した通信専用スレッドを利用するライブラリよりも，より軽量な通信を行うために通信専用スレッドは設けていない．これによって，従来の MPI のマルチスレッド環境における通信性能の低下の問題を解決することができるということを確認した．しかしながら，ノード数やスレッド数が大きくなることで，非常に多くの通信のエンドポイントを作成することになる．そのため，メモリ使用量の増加や，エンドポイントをどうシェアリングしていくかという問題がある．IBV においては，RC プロトコルを拡張した XRC プロトコルや DC プロトコルを使用してメモリ使用量を低減することができると考えており，今後の検証課題としたい．

また，本稿では IBV の実装のみに言及していたが，OPA 向けの PSM2 や Tofu インターコネクト 2/D 向けの uTofu 用いた実装も検討していく．同時に，複数のインターコネクト向けの低レベル通信 API を包含した軽量かつ，ユーザビリティが高い通信レイヤの API を検討する．

謝辞 本稿の一部は，文部科学省「特定先端大型研究施設運営費等補助金（次世代超高速電子計算機システムの開発・整備等）」で実施された内容に基づくものである．

参考文献

[1] P. Balaji, D. Buntinas, D. Goodell, W. Gropp, and R.Thakur. Fine-Grained Multithreading Support for Hybrid Threaded MPI Programming. *The International Journal of High Performance Computing Applications*, Vol. 24, No. 1, pp. 49–57, Feb. 2010.

[2] Mellanox. RDMA Aware Networks Programming User Manual Rev 1.7. `http://www.mellanox.com/related-docs/prod_software/RDMA_Aware_Programming_user_manual.pdf`.

[3] M. S. Birrittella, M. Debbage, R. Huggahalli, J. Kunz, T. Lovett, T. Rimmer, K. D. Underwood, and R. C. Zak. Intel Omni-path Architecture: Enabling Scalable, High Performance Fabrics. In *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, pp. 1–9, Aug 2015.

[4] Intel Performance Scaled Messaging 2 (PSM2) Programmer’s Guide. `https://www.intel.com/content/dam/support/us/en/documents/network-and-i-o/fabric-products/Intel_PSM2_PG_H76473_v_11_0.pdf`.

[5] Y. Ajima, T. Kawashima, T. Okamoto, N. Shida, K. Hirai, T. Shimizu, S. Hiramoto, Y. Ikeda, T. Yoshikawa, K. Uchida, and T. Inoue. The Tofu Interconnect D. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 646–654, Sep 2018.

[6] K. Tsugane, J. Lee, H. Murai, and M. Sato. Multitasking Execution in PGAS Language XcalableMP and Communication Optimization on Many-core Clusters. In *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, HPC Asia 2018, pp. 75–85, 2018.

[7] W. Endo and K. Taura. Parallelized Software Offloading of Low-Level Communication with User-Level Threads. In *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, HPC Asia 2018, pp. 289–298, 2018.

[8] 遠藤亘, 田浦健次朗. PGAS 向け低水準通信レイヤーのマルチスレッド実装. 情報処理学会研究報告 (ハイパフォーマンスコンピューティング), Vol. 2016-HPC-155, No. 28, pp. 1–11, Aug 2016.

[9] L. Meadows, K. Ishikawa, T. Boku, and M. Horikoshi. Multiple Endpoints for Improved MPI Performance on a Lattice QCD Code. In *Proceedings of Workshops of HPC Asia*, HPC Asia ’18, pp. 67–70, 2018.

[10] GASNet Communication System. `http://gasnet.lbl.gov/`.

[11] D. Bonachea and P. H. Hargrove. GASNet-EX: A High-Performance, Portable Communication Library for Exascale. *Languages and Compilers for Parallel Computing (LCPC’18)*, Oct 2018.

[12] UCX-Unified Communication X. `http://www.openucx.org/`.

[13] ComEx: Communications Runtime for Extreme Scale. `http://hpc.pnl.gov/comex/`.

[14] 中尾昌広, Tuan Tran Minh, 李珍泌, 朴泰祐, 佐藤三久. PGAS 言語 XcalableMP における coarray 機能の実装と評価. 先進的計算基盤システムシンポジウム論文集（SACSIS）, 第 2012 巻, pp. 289–297, May 2012.

[15] H. Iwashitai, M. Nakao, H. Murai, and M. Sato. A Source-to-Source Translation of Coarray Fortran with MPI for High Performance. In *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, HPC Asia 2018, pp. 86–97, 2018.

[16] 中村実. InfiniBand プログラムの覚え書きのインデックス. `http://www.nminoru.jp/~nminoru/network/infiniband/`.