

ACP ライブラリの通信性能およびメモリ使用量の評価

森江 善之^{1,2,a)} 本田 宏明^{1,2,b)} 南里 豪志^{1,2,c)}

概要：ACE (Advanced Communication for Exa) プロジェクトでは、省メモリかつ低遅延な低レベル通信ライブラリ ACP (Advanced Communication Primitives) の開発、公開を実施している。本稿では、開発通信ライブラリ ACP 基本層と同様に片側通信のインターフェースを持つ MPI RMA をそれぞれ使用した場合のメモリ使用量および通信性能について比較する。本実験では、まず、ACP 基本層の InfiniBand 版のメモリ使用量が Open MPI-1.10.1 に対して最大で 1/80 となることを示し、ACP 基本層が省メモリな実装であることを確認した。また、通信性能について ACP 基本層の片側通信 `acp_copy` が Open MPI-1.10.1 および MVAPICH2-2.2b の `MPLPut/MPLGet` と同等の性能を示すことがわかった。一方、小メッセージサイズおよび大メッセージサイズにおいては性能改善の余地があることがわかった。

1. はじめに

エクサスケール時代における大規模並列計算システムでは、その性能を達成するため、プロセッサのメモリアクタリがさらに進むと考えられる。一方、各プロセッサあたりのメモリ量は、対電力性能比の制約が厳しいことからメモリアクタリの進行に対して十分に増加することは望めない。このような文脈において、ユーザはメモリ領域を効率よく使用することが必要になると予想される。他方で、通信ライブラリ側は、ユーザのメモリ領域を圧迫しないことが重要となる。このため、大規模並列計算システムにおいて省メモリかつ低遅延な通信ライブラリを実現することが課題となる。

そこで、我々は、ACE (Advanced Communication for Exa) プロジェクト [1] においてこの課題に対応した低レベル通信ライブラリ ACP (Advanced Communication Primitives) の開発を行うこととした [2]。この ACP ライブラリは、低レベルな通信を抽象化する基本層とその上に実装される中間層で構成される。この中間層はコミュニケーションライブラリおよびデータライブラリというサブライブラリで構成される。この ACP 基本層の仕様策定 [3] を終えてその実装を公開した。また、ACP 基本層に関しては実装お

よび単体性能の評価を終えた [4][5]。さらに中間層の大部分の仕様策定を終えてその実装を公開した。

本稿では、我々が開発している ACP ライブラリの基本層と MPI ライブラリの実装とのメモリ使用量および片側通信の性能を比較を行い、その省メモリ性と低遅延性を示す。

本稿の構成は以下のようになっている。まず、2 節でエクサスケール環境での MPI の状況について述べる。次に 3 節で ACP 基本層の説明を行う。また、4 節で ACP 基本層の実装について述べ、5 節で ACP 基本層の実メモリ使用量および通信性能の評価について述べる。最後にまとめと今後の課題を述べる。

2. エクサスケール環境下の MPI

MPI のような既存の通信ライブラリのインターフェースは両側通信を基本とするが、両側通信ではその通信を実行するには通信バッファを利用する。さらに通信バッファを確保した場合に通信性能との関係などから通信バッファの解放を行わない。特にエクサスケール環境下では、通信のみに関するメモリの要求量が 10GB/プロセスに至るという予測結果がある [6]。

そこで、エクサスケール時代には通信バッファの使用を前提としない通信が重要になると考えられる。これについては通信バッファを用いずに RDMA 等の片側通信の機能を使うことにより通信バッファの使用を排除することが可能である。MPI にも片側通信がインターフェースとしては提供されている。しかし、現在の実装においては `MPLPut` などの片側通信インターフェースのみを使用した場合にも

¹ 九州大学情報基盤研究開発センター
Research Institute for Information Technology, Kyushu University, Fukuoka, 812-8586, Japan

² 独立法人科学技術振興機構 戦略的創造研究推進事業
Japan Science and Technology Agency, CREST, Chiyoda, Tokyo 101-0076, Japan

a) morie.yoshiyuki.404@m.kyushu-u.ac.jp

b) honda.hiroaki.971@m.kyushu-u.ac.jp

c) nanri@cc.kyushu-u.ac.jp

通信バッファが生成される。

3. ACP 基本層

先に述べた通りエクサスケール環境下の通信ライブラリを用いる場合は通信バッファの削減が重要となる。そこで我々は片側通信を基本とする通信ライブラリ“ACP ライブラリ”を提案する。

この ACP ライブラリは基本層と中間層からなり、基本層では、図 1 に示すように RDMA 等の片側通信機能を直接使用できるインターフェースを提供する。これにより両側通信では必要である通信バッファの生成を排除する。その結果としてユーザのメモリ領域が圧迫されることがなくなる。

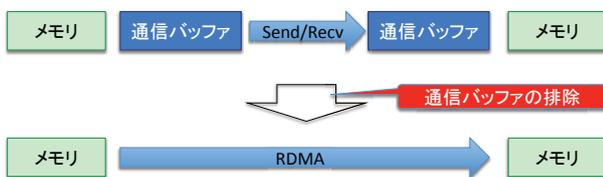


図 1 RDMA などの片側通信機能により通信バッファを排除

ACP 基本層では、グローバルメモリモデルを導入することで片側通信インターフェースによりリモートメモリを一樣にアクセスすることが可能である。これにより通信バッファを削減するだけでなく、リモート側のメモリ領域を自由に利用することができる。このことにより、並列計算機全体のメモリ利用効率を向上させることが可能である。表 1 に ACP 基本層のコピー、完了確認、メモリ登録、グローバルアドレス取得関数を示す。これらの関数群によりグローバルメモリ空間における片側通信を記述できるようになる。

3.1 グローバルメモリアクセス関数

グローバルメモリアクセス関数とは、グローバルメモリにおいて片側通信を発行する関数である。acp_copy の他にもアトミック操作を実行する片側通信関数が提供されている。グローバルメモリアクセス関数はコピー元およびコピー先に ACP 基本層が提供するグローバルアドレスによりグローバルメモリに対してアクセスが可能となる。

さらに、acp_copy 関数では図 2 のようにコピー元およびコピー先のグローバルアドレスがともにローカルメモリ領域でなくても通信を発行できる。この機能をリモート間コピーと呼ぶ。リモート間コピーは Put や Get では一時的なデータの中継地が必要となる場合などの無駄なデータの移動や同期を削減することが可能となる。

3.2 メモリ登録

ACP 基本層におけるグローバルメモリは、各ランクが

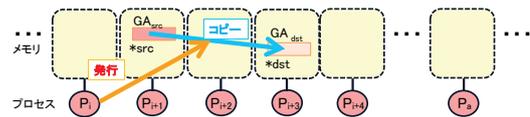


図 2 リモート間コピー

acp_register_memory 関数でローカルのメモリ領域を登録することで生成される。この関数の処理はローカル側で閉じており、メモリ領域の登録情報を非明示に他ランクへ通知することはしない。これは、すべてのランクが対応するメモリ領域へアクセスするとは限らないためである。メモリ領域の情報は acp_query_ga 関数により生成されたグローバルアドレスをユーザが交換することで伝える。これによりそのメモリ領域へのアクセスを行わないランクでは、不必要にメモリを使用することがなくなる。

3.3 オーダーハンドリング機能

オーダーハンドリング機能とはグローバルメモリアクセスの実行順序を制御するものである。

図 3 は GA0 から GA1 へのコピーが完了してから GA1 から GA2 へのコピーを発行する例である。2 行目で発行される acp_copy の第 4 引数に 1 行目で発行される acp_copy のハンドル hnd0 を指定することで 1 行目の acp_copy が完了するまで 2 行目の acp_copy の発行をブロックできる。

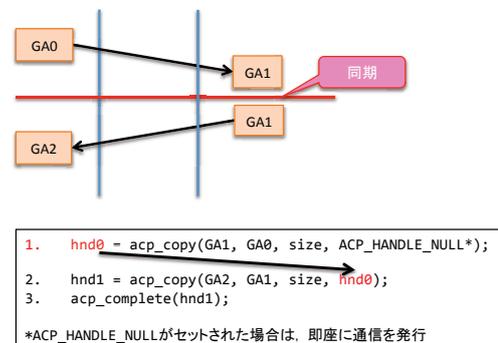


図 3 オーダーハンドリング

また、オーダーハンドルとして指定されたグローバルメモリアクセスより以前に発行されたグローバルメモリアクセスに関しても全て完了するまで呼び出しグローバルメモリアクセスをブロックする。これにより、複数のメモリアクセスの依存関係を簡便に記述できる。

4. ACP 基本層の実装

4.1 基本設計

ACP 基本層は図 4 のようにメインスレッドと通信スレッドの 2 スレッドで構成される。メインスレッドでは ACP 基本層の関数の呼び出しに対する処理を行う。通信スレッドは ACP ライブラリの通信関連の処理全般を実行する。両スレッドの間にはコマンドキューが配置されメインス

表 1 主な提供関数

名称	インターフェース
コピー	acp_handle_t acp_copy(acp_ga_t dst, acp_ga_t src, size_t size, acp_handle_t order)
完了	void acp_complete(acp_handle_t handle)
メモリ登録	acp_atkey_t acp_register_memory(void * addr, size_t size, int color)
グローバルアドレス取得	acp_ga_t acp_query_ga(acp_atkey_t atkey, void * addr)

レッドがコマンドキュー経由で通信スレッドに通信等の処理を依頼する仕組みをとる。

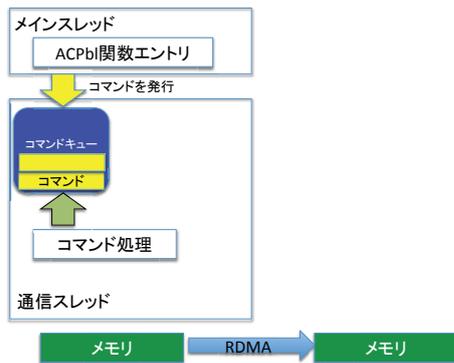


図 4 ACP 基本層の構造

まず、ACP 基本層の同期関数の処理について述べる。この処理は通信スレッドに依頼されることなくメインスレッドのみで実行される。通信完了待ち関数 `acp_complete` などがこれに当たる。

一方、`acp_copy` のような非同期関数は、メインスレッドでは、処理内容からコマンドを生成してコマンドキューにコマンドをエンキューする。この時、キューエントリを特定できるハンドルを返すと即座に制御を戻す。他方、通信スレッドではコマンドキューを監視してコマンドがあれば、そのコマンドに従い RDMA 通信等の処理を実施する。コマンドはエンキューされた順序で処理される。これによりオーダーハンドリング機能を実現する。

また、リモート間コピーについてであるが、これを発行可能なハードウェアが現状存在しない。このため、この機能をソフトウェア的に実装する必要がある。そこで、通信スレッド上にリモートコマンド受信バッファを置く。ここで他ランクからのコマンドを受け取る。そのコマンドを受け取ったランクはそのコマンドの内容に従い RDMA 通信やアトミック操作等を適宜処理する。

図 2 にリモート間コピーの動作概要を示す。まず、発行ランクがコマンドキューからコマンドを受け取る。このコマンドがリモート間コピーである場合にリクエストコマンドを送信元ランクのリモートコマンド受信バッファに渡す。送信元ランクはリモートコマンド受信バッファからそのコマンドを受け取り解釈して宛先ランクに対するリモートメモリアクセスを実施する。送信ランクがリモートメモリアクセスの完了を確認したら、発行ランクのリモートコマン

ド受信バッファへ完了コマンドを送る。発行ランクが完了コマンドを受信したらリモート間コピーを完了させる。

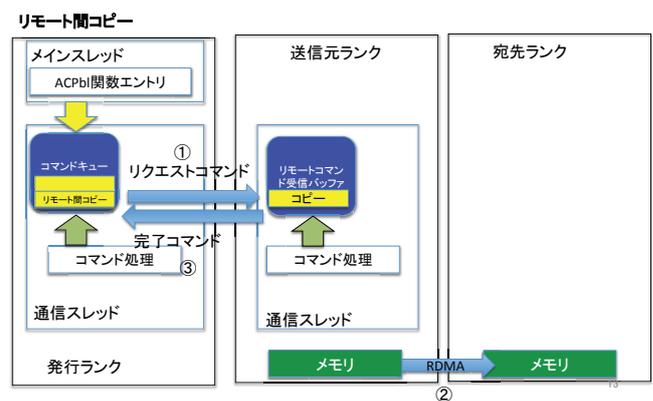


図 5 リモート間コピーの実装概要

4.2 通信デバイス依存の実装

ACP 基本層では、片側通信を提供するとともに異なる通信デバイスでも実行可能となるようにポータビリティを担保する。これにより中間層では通信デバイスに依存しない設計が可能となる。ここでは InfiniBand における実装について述べる。

4.2.1 グローバルアドレス

ACP 基本層ではグローバルアドレスを元に通信を行う。このため、通信時に必要な情報はグローバルアドレスから取得可能でなければならない。InfiniBand では、RDMA 通信を行う際はリモート側のメモリアドレスと Rkey が必要になる。グローバルアドレスは 64bit の変数でありメモリアドレスと Rkey をそのまま取めることができない。このため、図 6 のようにメモリ領域の登録情報を管理するテーブルを別に作成する。また、グローバルアドレスには作成したテーブルを参照するためのエントリを置くこととする。

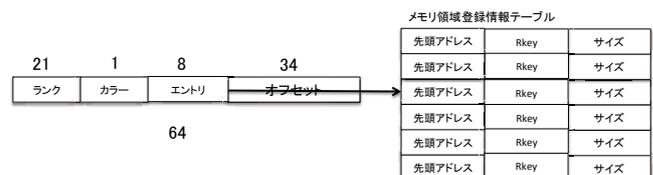


図 6 グローバルアドレス

4.2.2 メモリ領域登録情報テーブル

メモリ領域登録情報テーブルははじめ登録したローカルランクが持つ。グローバルアドレスは、ユーザによりリモートランクへ渡されるが、リモートランクにはメモリ領域登録情報がない。このため、グローバルアドレスのエントリからアクセスする先が無い。この問題に対応するため、リモートランクがローカルランクのメモリ領域登録情報テーブルをコピーする必要がある。

具体的には、はじめて当該メモリ領域にリモートランクがアクセスする際にメモリ領域を登録したローカルランク側にあるテーブルをコピーする。ローカルランク側のテーブルは初期化時に全リモートランクから RDMA によりアクセス可能とする。

このテーブルを 100 万プロセス分生成すると 10GB 使用することとなり、省メモリ性を損なう。そこで、このテーブルの最大保持数を 1024 個とするキャッシュを作成する。それ以上のランクとアクセスする場合には、古いテーブルを削除して新しいテーブルをコピーしてくる。これにより、ランク数に比例して使用メモリ量が増加することがなくなる。

このリモートランク側のテーブルのキャッシュの内容は、メモリアクセスが発生する時点においてローカルランク側のメモリ領域登録情報と一致していなければならない。ローカルランク側にあるメモリ領域の登録を解除して新たに異なるメモリ領域の登録を行った際に同じエントリが割り付けられる可能性があるからである。この際にリモートランク側から対象となるメモリ領域へのアクセスが発生した場合にそのメモリ領域の登録情報が更新されていなければ、正しいエントリであるにも関わらず、以前登録されていた異なるメモリ領域にアクセスすることとなる。

今回の実装では、メモリ領域が解除されて新たに異なるメモリ領域が同じエントリに割り付けられた場合に自ランクのテーブルのキャッシュを持つリモートランクへキャッシュの破棄を促すリクエストを発行する。リモートランク側では破棄を要求されたランクのテーブルのみを破棄する。更新ではなく破棄とするのは以後新たに登録されたメモリへアクセスするとは限らないためである。破棄されたあと再びメモリアクセスが発生した際に新たにテーブルをキャッシュすることとする。

4.2.3 メモリ使用量

ここで、InfiniBand 上に ACP 基本層を実装した際のメモリ使用量の概算について述べる。エクサスケール時代の大規模並列計算機を想定するため、100 万ランクでの実行を仮定する。表 2 に 1 ランクごとに必要なメモリ使用量を示す。

今回の実装で最もメモリ使用量が多いのは InfiniBand の接続資源である Queue Pair (QP) の生成でその値は 160MB となる。これは、Reliable Connection (RC) でラン

表 2 メモリ使用量

用途	容量
コマンドキュー	480KB
リモートコマンド受信バッファ	960KB
メモリ領域登録情報テーブル	約 10KB (40B * 255 個)
メモリ領域登録情報テーブルのキャッシュ	10MB (10KB * 1024 テーブル)
キャッシュ更新情報	4MB (4B * 1M ランク)
QP の生成	160MB (10KB * 1M ランク)
初期通信のためのメモリ領域情報テーブル	約 12MB (12B * 255 個)

ク間を接続するためにはターゲットランクごとに QP が必要となるからである。また、1QP のデータサイズが 160B と比較的大きいことも原因である。一方、メモリ領域登録情報テーブルのキャッシュについては最大保持数を 1024 個としたことにより 10MB に抑えられた。その他のメモリ使用量は合計でも 17MB 程度と十分に小さい値となった。

5. メモリ使用量および通信性能の評価

5.1 実験概要

ACP 基本層の通信性能と実メモリ使用量について評価する。

まず、メモリ使用量についてはプロセス数 2, 4, 8, 16 個の場合についてそれぞれ各通信ライブラリのメモリ使用量を評価する。評価プログラムでは、ランク 0 からその他の全ランクに対して Put でデータを送信する One_put_all を実行する。

One_put_all は、0KB から 32KB のメッセージを 1000 回、64KB のメッセージを 640 回、128KB のメッセージを 320 回、256KB のメッセージを 160 回、512KB のメッセージを 80 回、1MB のメッセージを 40 回、2MB のメッセージを 20 回、4MB のメッセージを 10 回を送る。この際のランク 0 のメモリ使用量について ACP 基本層と MPI ライブラリの比較を行う。

この際、Put としては ACP 基本層では acp_copy, MPI では MPI_Put 関数を用いた。また、比較する MPI ライブラリには、Open MPI-1.10.1 を用いた。この時、MPI プログラムは、mpirun のオプション等を設定せずにデフォルトの状態で行う。また、メモリ使用量の測定には、ライブラリや関数単位でのメモリ使用量の統計情報を取得することが可能な DMATP[7] を用いた。

一方、通信性能については ACP 基本層と Open MPI-1.10.1[8] および MVAPICH2-2.2b[9] について片側通信の性能評価を実施した。MPI ライブラリの実行にはベンチマークプログラムである Intel MPI Benchmarks 4.0 (IMB4.0) [10] を用いた。ACP 基本層は、IMB4.0 を模倣した ACP プログラムを記述した。この時、片側通信の時間を計測し、その際の実効通信帯域幅および通信遅延を示す。

5.2 実験環境

実験環境としては、PRIMERGY RX200 S7 を用いた。計算ノード数は 16 基で、各ノードに Intel Xeon プロセッサ E5-2609 (2.40 GHz) が搭載される。メモリは 8GB、計算ノード間は InfiniBand QDR スイッチで接続されており、そのスループットは片方向 4.0GB/s となる。また、OS カーネルは 2.6.32-220.el6.x86_64 で、コンパイラは gcc version 4.4.7 である。

5.3 実験結果

まず、表 3 に ACP と Open MPI-1.10.1 の One_put_all を実行したランク (ランク 0) のメモリ使用量を示す。

プロセス数	ACP	Open MPI-1.10.1
2	5.6MB	55.9MB
4	5.7MB	133.3MB
8	6.0MB	287.6MB
16	6.6MB	532.9MB

ACP 基本層は、Open MPI に対して 2 プロセスでメモリ使用量が 1/10、16 プロセスでメモリ使用量が 1/81 となった。これは、ACP 基本層では、そもそも通信バッファを利用しないのに対して Open MPI では MPI_Put の呼び出し時に通信バッファが生成されるからである。特に本実験では、MPI_Put が複数回数呼び出されて回数を重ねることで通信バッファが生成され、それらが積み上がったことがメモリ使用量に大きな差が生じた原因だと考えられる。

MPI_Put によるメモリ使用量は、プロセス 2 個の場合は 29.7MB となり全体の 53.1%、プロセス 4 個の場合は 89MB となり全体の 67.8%、プロセス 8 個の場合は 210.4MB となり全体の 73.2%、プロセス 16 個の場合は 451.2MB となり全体の 84.7% となる。送信先プロセス数が増えることで通信発行回数が増え、通信バッファがより多く生成されたと考えられる。

次に図 5.3 に ACP と Open MPI-1.10.1 および MVAPICH2-2.2b の Put 時の実効通信帯域幅と通信遅延を示す。これより、ACP 基本層は小メッセージサイズでは MVAPICH には劣るが、Open MPI より高速に動作することがわかった。また、中メッセージサイズにおいては MVAPICH と同等でかつ Open MPI より高速に動作する。さらに、大メッセージにおいては Open MPI には劣るが、MVAPICH と同等の性能で動作することがわかった。総じて既存の MPI ライブラリと同等の通信性能となっていることがわかる。また、小メッセージサイズおよび大メッセージサイズでは、通信性能のチューニングの余地があると考えられる。

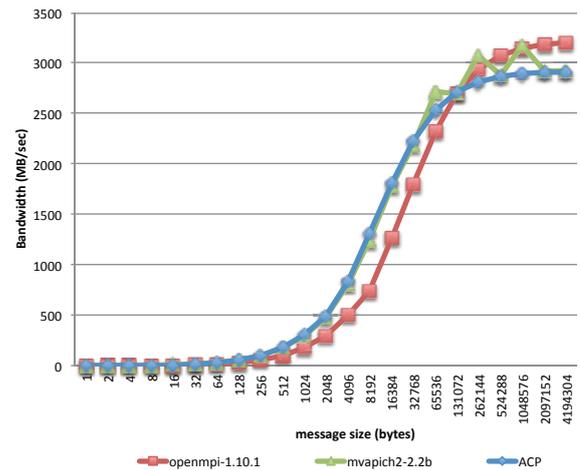


図 7 ACP 基本層と Open MPI および MVAPICH の Put の実効通信帯域幅

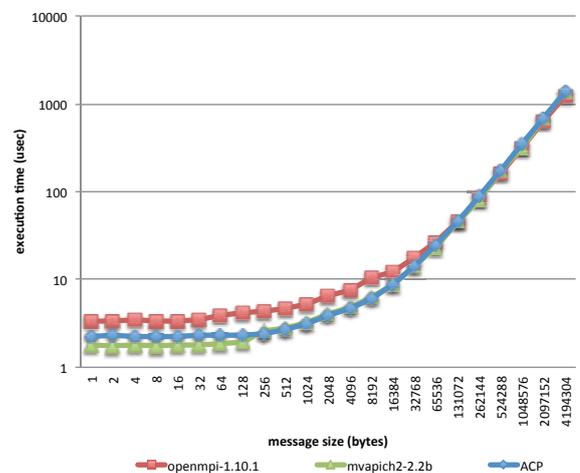


図 8 ACP 基本層と Open MPI および MVAPICH の Put の通信遅延

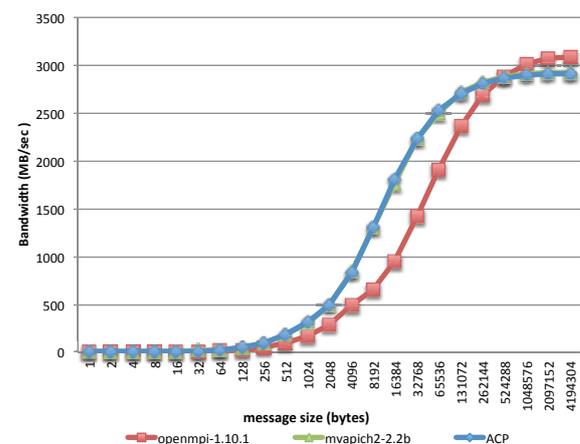


図 9 ACP 基本層と Open MPI および MVAPICH の Get の実効通信帯域幅

6. おわりに

本稿では、エクサスケール時代においては通信ライブラ

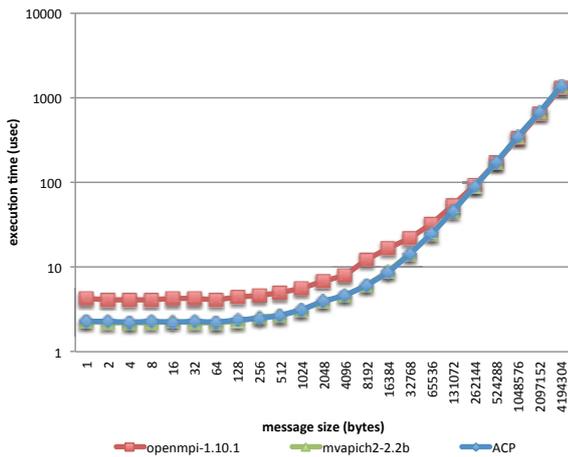


図 10 ACP 基本層と Open MPI および MVAPICH の Get の通信遅延

りの省メモリ化が必要であることを指摘し、通信バッファの削減が可能な片側通信を基本とした通信ライブラリ ACP 基本層を紹介した。また、ACP 基本層の片側通信 `acp_copy` と MPI RMA のメモリ使用量および通信性能の比較を行った。この時、InfiniBand 環境において Open MPI に対して十分に少ないメモリ使用量で片側通信を実行できることを示した。さらに、ACP 基本層の片側通信が既存の MPI ライブラリの片側通信と同等の性能であることを示した。

今後の課題としては、さらに大規模な並列計算環境において ACP 基本層の使用メモリ量の計測することと小メッセージサイズおよび大メッセージサイズにおける片側通信の性能改善を実施することがあげられる。

謝辞 本研究は、独立行政法人科学技術振興機構 戦略的創造研究推進事業 (CREST) 「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」研究領域、「省メモリ技術と動的最適化技術によるスケーラブル通信ライブラリの開発」の一部として実施された。ここに記して感謝致します。

参考文献

[1] ACE Project (online), 入手先 <http://ace-project.kyushu-u.ac.jp/index.html>.

[2] 住元 真司, 安島 雄一郎, 佐賀 一繁, 三浦 健一, 野瀬 貴史, 高見 利也, 南里 豪志, “エクサスケール通信向け ACP スタックの設計思想”, 情報処理学会研究会報告, vol.2014-HPC-143-8 pp.1-7, (2014).

[3] 安島 雄一郎, 佐賀 一繁, 三浦 健一, 野瀬 貴史, 住元 真司, “ACP 基本層の設計思想とインターフェース”, 報処理学会研究会報告, vol.2014-HPC-143-9, pp.1-6, (2014).

[4] 佐賀 一繁, 安島 雄一郎, 野瀬 貴史, 三浦 健一, 住元 真司, “ACP 基本層の実装と初期評価”, 報処理学会研究会報告, vol.2014-HPC-143-10, pp.1-6, (2014).

[5] 森江 善之, 南里 豪志, 安島 雄一郎, 本田 宏明, 曾我 武史, 小林 泰三, 住元 真司 “InfiniBand による ACP 基本層の実装と評価”, 情報処理学会研究会報告, vol.2015-HPC-148-33, pp.1-7, (2015).

[6] 住元 真司, 秋元 秀行, 安島 雄一郎, 安達 知也, 岡本 高

幸, 三浦 健一, “DMATP-MPI を用いた MPI ライブラリの関数別メモリ使用量評価”, 情報処理学会研究会報告, vol.2013-HPC-138-15, pp.1-7, (2013).

[7] 秋元 秀行, 安島 雄一郎, 安達 知也, 岡本 高幸, 三浦 健一, 住元 真司, “DMATP-MPI: MPI 向け動的メモリ割当分析ツール”, 情報処理学会研究会報告, vol.2013-HPC-138-14, pp.1-7, (2013).

[8] Open MPI: Open Source High Performance Computing (online), 入手先 <http://www.open-mpi.org/>.

[9] MVAPICH: MPI over InfiniBand, 10GigE/iWARP and RoCE (online), 入手先 <http://mvapich.cse.ohio-state.edu/>

[10] Intel MPI Benchmarks 4.0 (online), 入手先 <https://software.intel.com/en-us/articles/intel-mpi-benchmarks>).