Vol.2016-HPC-156 No.4
2016/9/15

ACP 基本層 UDP 版におけるノード内複数プロセス時のノード間通信性能の改善

安島雄一郎[†1†2]　野瀬貴史[†1†2]　佐賀一繁[†1†2]　志田直之[†1†2]　住元真司[†1†2]

概要：本論文では Advanced Communication for Exa (ACE)プロジェクトで開発している Advanced Communication Primitives (ACP)ライブラリの UDP 版基本層において，各ノードに複数プロセスが存在し，ノード内の複数プロセスが同時にノード間通信を行う際の性能低下を改善する．従来の UDP 版基本層実装ではノード内に複数プロセスがある場合も各プロセスがそれぞれ帯域制御，輻輳制御を行っていた．しかし通信が競合する場合，輻輳制御を行っても実効通信帯域の低下は避けられない．本研究では 1 ノードあたり 1 つの代表プロセスのみノード間通信を行う方式により，ノード内の複数プロセスによる送信の競合を回避し，通信性能を改善する．

1. はじめに

エクサスケール時代の HPC システムでは，メニーコア・プロセッサと広帯域の三次元積層メモリがキーテクノロジとして期待されている．メニーコアと広帯域メモリの組合せでは，コアあたりのメモリ容量が減少することが課題である．エクサスケール時代では，通信ライブラリを含むシステムソフトウェアにおいてメモリ消費量の削減が重要となる．従来の多くの通信ライブラリでは，通信バッファは自動的に割当てられ，解放せずに割り当てられ続ける．これは通信バッファの解放，再割当てとも処理コストが高いためである．

Advanced Communication for Exa (ACE) プロジェクト[1]ではプロセスあたりのメモリ消費量を抑制しつつ，低遅延通信を実現する通信ソフトウェア技術の創出に取り組んでいる．我々は ACE プロジェクトの中核技術の一つとして，利用者がメモリ消費量を意識したプログラミングが可能であるように，明示的にメモリを使用するインタフェースを備える低レベル通信ライブラリ Advanced Communication Primitives (ACP)を開発している[2].

ACP は省メモリに適した PGAS モデルでインターコネクトデバイスを抽象化する基本層[3]と，様々な省メモリアルゴリズムがポータブルに実装される中間層で構成される．基本層は InfiniBand，Tofu インターコネクトなどの高速なインターコネクトに加え，UDP [4]にも対応する．UDP 版は幅広い環境で利用できるのでアプリケーション開発環境向けに最適であり，同じく Ethernet を使用する TCP と比べるとプロトコルスタックによるメモリ使用量が小さい．

従来の UDP 版 ACP 基本層では全てのプロセス間通信において UDP を使用していたが，1 ノードに複数のプロセスがあるばあい，複数ノードが同時に UDP 通信を行うと輻輳制御の影響で性能が低下していた．

本論文では UDP 版 ACP 基本層において 1 ノードに複数プロセスがある場合，1 ノードが代表で通信する方式を導入し，輻輳制御の競合を避ける方式を提案，評価する．以降では，2 章で UDP 版 ACP 基本層の概要と実装方式を説明し，3 章で課題を示すとともに改良方式を提案する．4 章で評価し，5 章では今後の課題について議論し，最後に 6 章でまとめる．

†1 富士通株式会社
Fujitsu Limited
†2 独立行政法人科学技術振興機構
Japan Science and Technology Agency (JST)

2. UDP 版 ACP 基本層

2.1 ACP 基本層のデータ転送

ACP 基本層では登録したメモリに対するグローバルアドレスを提供し，グローバルアドレスを使用したデータ転送関数を提供する．データ転送関数 acp_copy()は任意のグローバルアドレスから任意のグローバルアドレスへデータを転送する．一般に低レベル通信ライブラリでは他プロセスから自プロセスにデータ転送する Get，自プロセスから他プロセスにデータ転送をする Put を提供することが多いが，ACP では acp_copy 関数 1 つで Get，Put 両方のデータ転送が可能である．さらに acp_copy 関数では他プロセスのデータを別のプロセスに転送することもできる．

acp_copy 関数は非ブロッキングであり，データ転送の開始を待たずに戻る．acp_copy 関数は返り値として GMA ハンドル（acp_handle_t 型）を返す．GMA ハンドルは，その値を返した acp_copy 関数呼び出しによるデータ転送の完了待ち合わせに使用する．acp_complete 関数は GMA ハンドルを引数に取り，GMA ハンドルを返した acp_copy 関数およびそれ以前に呼び出された関数によるデータ転送の完了を全て待ち合わせてから戻る．また，acp_copy 関数自身も他のデータ転送の完了を待ち合わせる機能を有しており，GMA ハンドルを引数に取る．連続して acp_copy 関数を呼び出す場合，直前の acp_copy 関数呼び出しの返り値を引数として与えるとデータ転送が逐次に実行され，直前の acp_copy 関数と同じ引数を与えるとデータ転送が並列に実行される．GMA ハンドルとして ACP_HANDLE_ALL 定数を指定すると，直前に返された GMA ハンドルを指定したことになる．また GMA ハンドルとして ACP_HANDLE_NULL 定数を指定すると，他のデータ転送を待ち合わせずにデータ転送を実行する．

ⓒ2016 Information Processing Society of Japan

2.2 UDP 版 ACP 基本層の実装

UDP 版 ACP 基本層では初期化時に各プロセスで通信スレッドを 1 つ生成し,通信処理は全て通信スレッドで行う．データ転送関数は，通信スレッドに通信指示を格納したコマンドを供給する．ここで関数を呼び出したプロセスを開始元と呼ぶ．データ転送の宛先と送信元はグローバルアドレスで指定されるので，実際にデータ転送を実行するのは宛先もしくは送信元のプロセスの通信スレッドである．そこで，開始元の通信スレッドは通信の内容に従って，宛先もしくは送信元の通信スレッドにコマンドを転送し，実行を依頼する．コマンドを受信した通信スレッドは通信を実行し，コマンド実行の完了を開始元に通知する．

コマンドやデータは UDP データグラムで転送される．UDP は送達保証がないので，UDP 版 ACP 基本層自身が送達保証機能を実装している．受信側は受信したデータグラムに対して送達を通知する応答を返す．送信側は送達確認のタイムアウトを検出し，データグラムを再送信する．コマンド転送では順序の入れ替わりや重複を起こさないために，データグラムにシーケンス番号を付与する．受信側はシーケンス番号の抜けを検出すると送信側に通知し，送信側は送出帯域を下げる．パケット破棄は通信競合によるバッファ溢れが主な原因であるので，送出帯域を下げることでバッファに滞留するパケット数を減らし，パケット破棄の頻度を下げる．

3. 課題と提案

3.1 課題

UDP 版 ACP 基本層では 1 ノードに複数プロセスがある場合，各プロセスに異なる UDP ポート番号が割り当てる．各プロセスの通信スレッドは独立しており，個別に UDP 通信の輻輳制御を行う．すなわち，1 ノードの複数プロセスが同時に通信しようとするとポートの異なる UDP 送信が競合し，パケットが破棄される．パケット破棄は UDP 版 ACP 基本層の送達保障プロトコルで検出され,輻輳制御によってパケット破棄が起きない水準まで送出帯域が引き下げられることで通信が安定する．しかし，パケット破棄を契機とする制御過程はオーバーヘッドが大きく，望ましくない.

3.2 提案

ノード内の複数プロセスが UDP データグラムを送信する場合，ノード内の複数プロセス同士が事前に調停することで，送出帯域を制限することが望ましい．そこで，本研究ではノード内複数プロセスの 1 つをゲートウェイプロセス，その他を内部プロセスとし，ゲートウェイプロセスのみが UDP 送受信を行う方式を提案する．

内部プロセスは送出すべきデータグラムをゲートウェイプロセスに渡し，ゲートウェイプロセスは渡されたデータグラムを依頼元内部プロセスの UDP ポート番号で送出する．ゲートウェイプロセスはノード内全プロセスの UDP ポート番号でデータグラムを受信し，受信データグラムを受信ポート番号に対応する内部プロセスに渡す.

ゲートウェイプロセスと内部プロセス間のデータグラム受け渡しには Unix Domain Socket を使用するのが簡便であるが，システムコールとソケットバッファを介したコピーのオーバーヘッドが大きい. そこで, 本提案では無名 mmap を使用する共有メモリでデータグラムを受け渡す．共有メモリ上の共有変数によるデータグラム受け渡し制御はシステムコールよりもオーバーヘッドが小さく，またゲートウェイプロセスが共有メモリ上のデータグラムを直接 UDP 送信し，共有メモリに直接 UDP 受信することにより，コピーオーバーヘッドも最小化される．

再送信，輻輳制御については全てゲートウェイプロセスの通信スレッドで行い，ノード内全プロセス分の送出帯域を制御する方式に変更した．従来の実装では帯域制御として TCP と似たウィンドウ制御方式を採用していたが, 本提案では複数プロセスによる通信の合計帯域を制御するため, 高精度タイマーにより送出帯域を直接制御し，輻輳制御と分離した.

ゲートウェイプロセスの通信スレッドで再送信，輻輳制御を行う変更に伴い，内部プロセスの通信スレッドは，再送信，輻輳制御を行わなくなった．そのため，内部プロセスの通信スレッドは通常待機状態とし，メインスレッドからコマンド処理を依頼された，もしくはゲートウェイプロセスからプロトコル処理を依頼された場合にだけ実行を再開して処理を行う．本提案では，内部プロセスの通信スレッドは POSIX スレッドの条件変数で待機する実装とした.

新実装の帯域，再送信，輻輳制御は旧実装のプロセス対プロセスではなく，ノード対ノードを制御対象とする．ここで新実装では，内部プロセスからゲートウェイプロセスに受け渡すデータグラムの順序をノード間の転送でも維持する．また，宛先ノードが同じデータグラムの再送信順序は，最初にノード内の各内部プロセスから受け取った順序を維持する．この方式により，帯域，再送信，輻輳制御のプロトコル処理を軽量化した．旧実装は制御通信の経路上でも通信コマンドの順序変更を許容していたが，再送信のためのテーブル検索などのプロトコル処理が重く，1 スレッドでノード内複数プロセス分のプロトコル処理を行うとオーバーヘッド増加の懸念があった.

さらに，新実装では送信元が同じ逐次データ転送の場合は,順序制御を送信元で行う遠隔順序制御機能を実装した．この機能は旧実装でも試験的に実装したが，輻輳制御が破綻したため採用を見送っていた．新実装では再送信，輻輳制御とプロトコルが分離しており，原理的には遠隔順序制御による輻輳制御の破綻は起きないと想定される.

ⓒ2016 Information Processing Society of Japan

4. 評価

本章では UDP 版 ACP 基本層の新旧実装によるデータ転送のスループット，遅延を評価し，さらに複数プロセスが同時に通信する場合の性能を比較する．

評価は自ノードのプロセスから他ノードのプロセスにデータを転送する local-to-remote, 他ノードのプロセスから自ノードのプロセスにデータを転送する remote-to-local, 他ノードのプロセスからさらに別の他ノードのプロセスにデータを転送する remote-to-remote で行う．

また，新実装においてはデータ転送の開始元，データの送信元，データの宛先がそれぞれゲートウェイプロセスである場合と内部プロセスである場合を分け，ゲートウェイプロセスと内部プロセス間のデータグラム受け渡しのオーバーヘッドを評価する．具体的には 1 ノード 2 プロセス，合計 6 プロセスで起動し，各ノードのゲートウェイ 3 プロセス間のデータ転送と，内部 3 プロセス間のデータ転送を評価する．複数プロセス同時通信ではゲートウェイ 3 プロセス間のデータ転送を計測しつつ，同時に内部 3 プロセス間のデータ転送を行って干渉させる．

以上に追加して，新旧実装の UDP 版 ACP 基本層を使用した場合の ACP データライブラリ[5]の性能評価も行う．

4.1 評価環境

評価は PC クラスタで実施した. 表 1 に評価環境を示す. プロセッサのハイパースレッディングはオンになっており，1 ノードあたりのスレッド数はコア数の 2 倍の 16 であった. インターコネクトは Gigabit Ethernet を使用して評価した. データ転送は 4 バイトから 2MiB までのメッセージをそれぞれ小さいサイズは 1000 回ずつ，大きいサイズでは合計 40MiB，逐次転送する時間を計測し，データ転送 1 回あたりの平均時間を算出した．逐次転送であるので，各データ転送間の順序を保証しており，その制御時間も結果に含まれている．新旧実装では帯域制御方式が異なり，どちらも輻輳を起こさない保守的な設定となっているため，スループットの評価結果の絶対値，最大値は比較できない．

表 1 評価環境

Table 1 Evaluation environment

Node	Fujitsu PRIMERGY RX200 S5
CPU	Intel Xeon E5520 (4 cores, 2.27 GHz), 2 sockets
Memory	DDR3 SDRAM 48GB, 51.2 GB/s
Network	Gigabit Ethernet (125 Mbyte/sec)

4.2 スループット評価結果

図 1 は自ノードから他ノードにデータ転送する場合のスループット評価結果を示す．旧実装が 2MiB でも帯域が飽和していないのに比べ，新実装は 256KiB あたりで帯域が飽和しており，プロトコルオーバーヘッドが小さいことが分かる．さらにメッセージサイズが 1 データグラムの最大長を超える 2,048 バイトにおいて旧実装では帯域の低下が見られるのに対し，新実装ではほぼ帯域を維持している．この点からもプロトコルオーバーヘッドが削減されていることが読み取れる．

また，新実装ではゲートウェイプロセスでも内部プロセスでもほぼ同じ性能が得られており，共有メモリによるデータグラム受け渡しのオーバーヘッドは，UDP で実装されたプロトコルのオーバーヘッドに比べて十分小さいことが分かる．

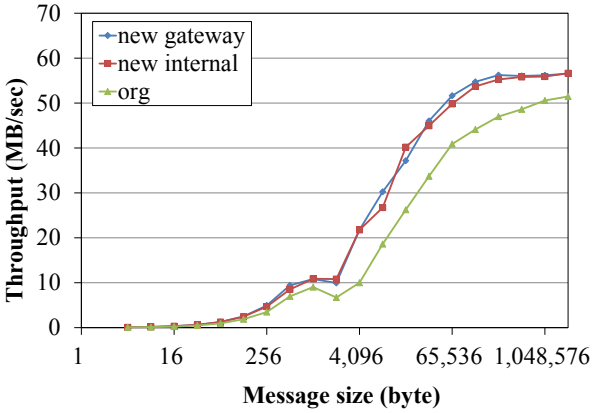

図 1 Local-to-remote データ転送のスループット

Figure 1 Throughput results of local-to-remote data transfer

図 2 は他ノードから自ノードにデータ転送する場合のスループットの評価結果を示す．旧実装では 1 データグラムにメッセージが収まる場合を別プロトコルで処理しており，この範囲では新実装の方が，帯域が低くなっている．メッセージサイズが大きい範囲では，旧実装は帯域が飽和していないのに対し，新実装は 256KiB あたりで帯域が飽和しており，プロトコルオーバーヘッドが小さいことが分かる. また, メッセージサイズが大きい範囲の新実装では，ゲートウェイプロセス間転送より内部プロセス間転送の方が，帯域が高くなっている．

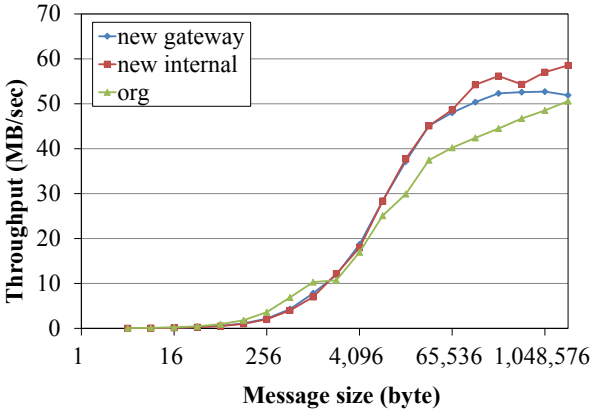

図 2 Remote-to-local データ転送のスループット

Figure 2 Throughput results of remote-to-local data transfer.

図 3 は他ノードから別の他ノードにデータ転送する場

ⓒ2016 Information Processing Society of Japan

合のスループットの評価結果を示す．この場合はプロトコルオーバーヘッドが大きいため，新実装でも 1MiB あたりまで帯域が飽和していない．また，旧実装では 512KiB 以降で帯域低下が見られる．これは送信元と宛先間のウィンドウ制御のパラメータ調整が不十分でパケットロスが起き，輻輳制御が働いていると考えられる．

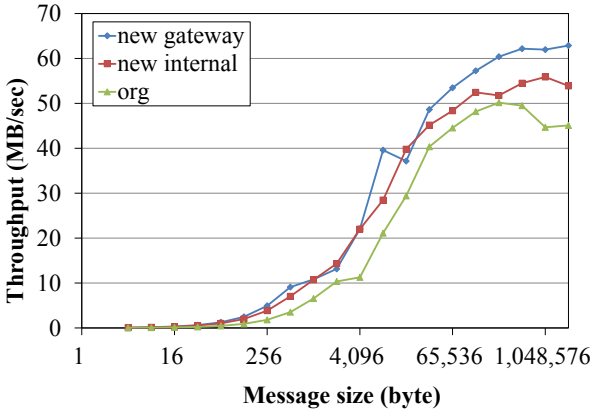

図 3　Remote-to-remote データ転送のスループット
Figure 3　Throughput results of remote-to-remote data transfer.

4.3 レイテンシ評価結果

図 4 は自ノードから他ノードにデータ転送する場合のレイテンシの評価結果を示す．メッセージサイズが小さい領域のレイテンシは旧実装が約 68 μ 秒，新実装が約 52 μ 秒で，ゲートウェイプロセスと内部プロセスで差がほとんどない．このことから，新実装は旧実装のプロトコル処理オーバーヘッドを約 16 μ 秒削減したことが分かる．新実装の共有メモリによるプロセス間のデータグラム受け渡しのオーバーヘッドは，全くないように見える．これは何らかの機構がオーバーヘッドを隠蔽していると考えられ，もっと詳細に調べる必要がある．

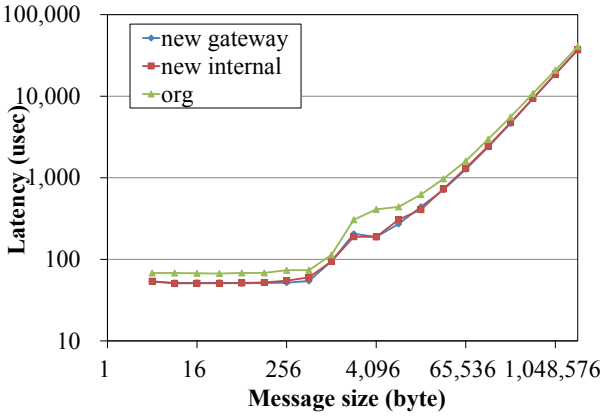

図 4　Local-to-remote データ転送のレイテンシ
Figure 4　Latency results of local-to-remote data transfer.

図 5 は他ノードから自ノードにデータ転送する場合のレイテンシの評価結果を示す．メッセージサイズが小さい領域のレイテンシは旧実装が約 69 μ 秒，新実装のゲートウェイプロセスが約 118 μ 秒，内部プロセスが約 128 μ 秒と旧実装の方が小さくなった．旧実装のレイテンシが小さい理由は，宛先が自ノードの場合に 1 往復で終わる別プロトコルで処理するためである．新実装では逐次のデータ転送は送信元で遠隔順序制御されるため開始元と送信元間のプロトコルによるオーバーヘッドが隠蔽され，1 往復で終わるプロトコルと同等のレイテンシとなることが期待されていた．しかし，この評価では開始元と宛先が同じであるため，開始元と送信元間のプロトコルと送信元と宛先間のプロトコルによる UDP 通信が干渉したと考えられる．

また，新実装の 2 つの評価結果の差から，ゲートウェイプロセスと内部プロセス間で開始元，送信元，宛先，送信元，開始先と合計 8 回データグラムが受け渡されるオーバーヘッドは約 10 μ 秒であることが分かる．

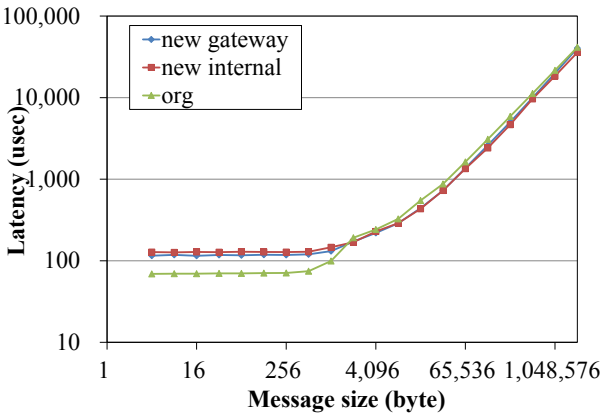

図 5　Remote-to-local データ転送のレイテンシ
Figure 5　Latency results of remote-to-local data transfer.

図 6 は他ノードから別の他ノードにデータ転送する場合のレイテンシの評価結果を示す．メッセージサイズが小さい領域のレイテンシは旧実装が約 142 μ 秒，新実装のゲートウェイプロセスが約 52 μ 秒，内部プロセスが約 63 μ 秒となった．新実装のゲートウェイプロセス間データ転送の評価結果は自ノードから他ノードに転送する場合とほぼ同じであり，遠隔順序制御によって開始元と送信元間のプロトコルによるオーバーヘッドが隠蔽されていることが分かる．また，新実装の 2 つの評価結果より，ゲートウェイプロセスと内部プロセス間での合計 8 回データグラム受け渡しのオーバーヘッドは約 11 μ 秒であった．旧実装では，開始元から送信元，宛先，送信元，開始先と合計 4 回行われるデータグラム転送が，全てオーバーヘッドに表れたと考えられる．

ⓒ2016 Information Processing Society of Japan

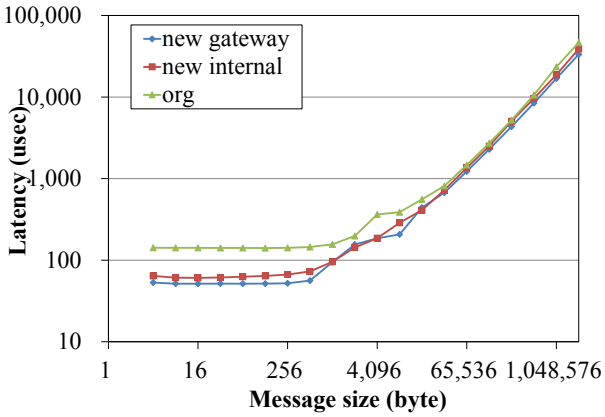

図 6　Remote-to-remote データ転送のレイテンシ

Figure 6　Latency results of remote-to-remote data transfer.

4.4 同一ノード内 2 プロセス同時転送評価結果

図 7 は自ノードから他ノードに，ゲートウェイプロセスと内部プロセスが同時にデータ転送する場合のスループット評価結果を示す．旧実装ではメッセージサイズ 32KiB 以上で輻輳制御の結果と思われる帯域低下が観測される．2MiB では 4.2 節の単独スループット評価結果に比べて約 3 割のスループットに留まる．もう一方のプロセスも同等のスループットとすると，合計で約 6 割である．新実装でもメッセージサイズ 32KiB 以上で帯域の伸びが鈍化しているが，4.2 節の単独評価結果に比べて約 8 割のスループットを維持している．パケットロスは起こさずに内部プロセス間のデータ転送と帯域をシェアしていると考えられるが，調停方式が公平でなく，ゲートウェイがより多くの帯域を使用していると考えられる．何らかの原因で合計帯域が 4.2 節の単独スループット評価結果よりも高くなっている可能性もあり，より詳細な評価が必要である．

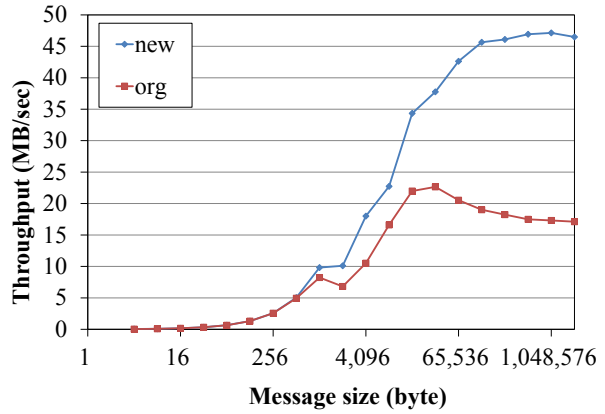

図 7　同一ノード内 2 プロセス同時 local-to-remote データ転送のスループット

Figure 7　Throughput results of simultaneous local-to-remote data transfer by two processes in the same node.

図 8 は他ノードから自ノードに，ゲートウェイプロセスと内部プロセスが同時にデータ転送する場合のスループット評価結果を示す．自ノードから他ノードの場合と同様に，旧実装ではメッセージサイズ 32KiB 以上でパケットロスと輻輳制御によると考えられる帯域低下が見られる．

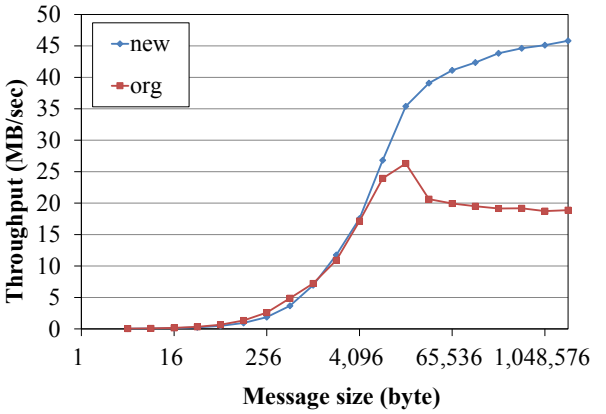

図 8　同一ノード内 2 プロセス同時 Remote-to-local データ転送のスループット

Figure 8　Throughput results of simultaneous remote-to-local data transfer by two processes in the same node.

図 9 は他ノードから別の他ノードに，ゲートウェイプロセスと内部プロセスが同時にデータ転送する場合のスループット評価結果を示す．前の 2 つの評価結果と同様に，旧実装ではメッセージサイズ 32KiB 以上でパケットロスと輻輳制御によると考えられる帯域低下が見られる．

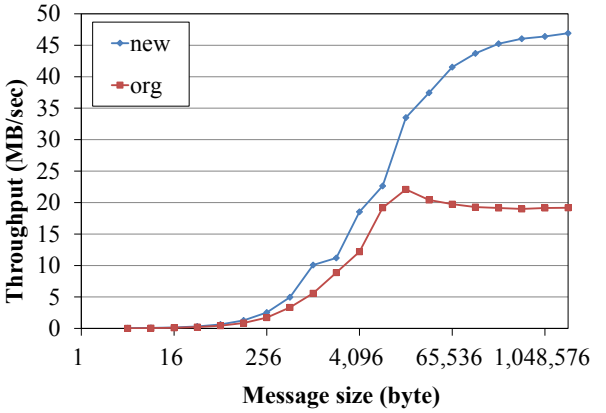

図 9　同一ノード内 2 プロセス同時 Remote-to-remote データ転送のスループット

Figure 9　Throughput results of simultaneous remote-to-remote data transfer by two processes in the same node.

4.5 データライブラリ性能評価結果

ACP データライブラリの性能評価として，ベクタ型データの先頭要素削除の平均実行時間を測定した．ベクタ型ではデータの一部を削除すると後続データを前に詰める必要があるが，後続データを削除サイズのチャンクに分け，逐次的に移動する，省メモリな in-place アルゴリズムを使用する．この方式では後続データの要素数に比例して，後続データの前詰に要する逐次データ転送の回数が増える．

ⓒ2016 Information Processing Society of Japan

図 10 に，予備評価として文献[6]において UDP 版 ACP 基本層の旧実装を使用した場合と，旧実装に遠隔順序制御を試験導入した場合の評価結果を示す．評価環境は本論文の評価環境と同一である．旧実装では入力ベクタが 1 要素増える毎に実行時間が約 70 μ 秒増加している．ここに遠隔順序制御を導入すると実行時間が入力要素数に対して指数的に増加する．この急激な性能低下は輻輳制御が破綻した結果と考えられる．

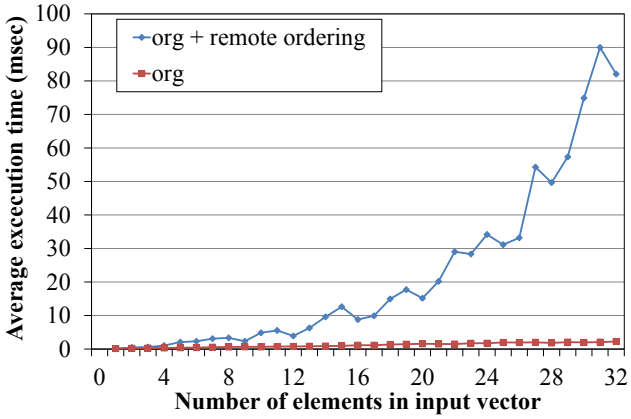

図 10　UDP 版基本層の旧実装を使用したベクタ削除の平均実行時間

Figure 10　Average execution time of the vector erasure function using the original version of ACPbl/UDP.

図 11 に，UDP 版 ACP 基本層の新旧実装を使用した，ベクタ型先頭要素削除の平均実行時間の評価結果を示す．新実装で入力 1 要素あたり約 13 μ 秒と旧実装の 1/5 以下に短縮された．この大幅な実行時間短縮は，in-place のデータ転送では送信元と宛先のプロセスが同一であるのでデータ転送が CPU コピーで行われ，実行時間の大半が開始元と送信元間のプロトコルオーバーヘッドで占められていたことを示している．

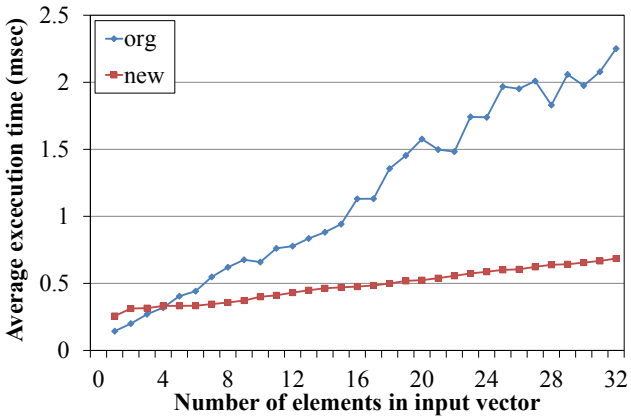

図 11　UDP 版基本層の新実装を使用したベクタ削除の平均実行時間

Figure 11　Average execution time of the vector erasure function using the new version of ACPbl/UDP.

5. 今後の課題

今後は本論文の評価では原因の分からなかった事象，すなわち自ノードから他ノードにデータ転送する場合のゲートウェイプロセスと内部プロセス間のデータグラム受け渡しレイテンシが隠蔽される事象，同一ノード内 2 プロセス同時転送においてゲートウェイプロセスで計測されたスループットが単独で計測した値の半分ではなく約 8 割に達していた事象について，より詳細に調査する必要がある．

新実装では他プロセスから自プロセスにデータ転送する場合に同じ経路をデータグラムが 2 往復して干渉しており，別プロトコル化も含めた対処を検討する必要がある．

本論文では同一ノード内のプロセス間のデータ転送性能を評価しなかったが，旧実装の UDP のループバックから新実装では共有メモリ経由に変わっており，大きな性能向上が見込まれるので，今後の研究で評価したい．

6. まとめ

本論文では UDP 版 ACP 基本層において，各ノードに複数プロセスが存在する場合に 1 ノードあたり 1 つのゲートウェイプロセスのみノード間通信を行う新方式を提案し，実装した．新旧実装を評価し，新実装はプロトコル処理のオーバーヘッドを旧実装から約 16 μ 秒削減し，ゲートウィプロセスと内部プロセス間のデータグラム受け渡しは 8 回あたり約 10 μ 秒と小さいことが分かった．ノード内 2 プロセスが同時にデータ転送を行った場合，新実装では輻輳制御に起因するスループット低下が起きないことを確認した．また，新実装では遠隔順序制御と輻輳制御が干渉しないため，逐次データ転送における開始元と送信元間のプロトコルオーバーヘッドを隠蔽できることを確認した．遠隔順序制御により，データライブラリのベクタ削除関数は入力 1 要素あたりの実行時間を 1/5 以下に削減した．

謝辞　本研究は JST CREST の支援により実施された．

参考文献

[1] ACE Project, http://ace-project.kyushu-u.ac.jp/index.html
[2] 住元真司, 安島雄一郎, 佐賀一繁, 野瀬貴史, 三浦健一, 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 2014-HPC-143-8 (2014)
[3] 安島雄一郎, 佐賀一繁, 野瀬貴史, 三浦健一, 住元真司: ACP 基本層の設計思想とインタフェース, 情報処理学会研究会報告 2014-HPC-143-9 (2014)
[4] Jonathan B. Postel (editor): User Datagram Protocol, RFC 768 (1980)
[5] 安島雄一郎, 野瀬貴史, 佐賀一繁, 志田直之, 住元真司: ACP の分散動的データ構造インタフェース, 情報処理学会研究会報告 2014-HPC-146-18 (2014)
[6] 安島雄一郎, 野瀬貴史, 佐賀一繁, 志田直之, 住元真司: 基本層データ転送方式の改良による ACP データ構造操作関数の高速化, 情報処理学会研究会報告 2016-HPC-153-31 (2016)

ⓒ2016 Information Processing Society of Japan