情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2010-HPC-126 No.12
2010/8/3

片方向通信の実装方式の違いによる比較

堀　敦史[†1]　李　珍泌[†1]　佐藤三久[†1]

片方向通信は，ハードウェアによる RDMA（Remote Direct Memory Access）と相性が良く，高い通信性能を実現できる方式として注目されている．本稿では，片方向通信をソフトウェアで実装する際のポイントを挙げ，既存の片方向通信ライブラリである ARMCI と GASNet について実装方式を調査した．その結果，RDMA を用いずにメッセージ通信を下位レベルの通信レイヤーとして片方向通信が実現できること，RDMA を用いた場合の実装に比べ実装が容易であること，さらに RDMA 方式に劣らない性能を発揮する可能性があることを示す．また実際にこの方式による片方向通信ライブラリ Telemem を開発し，RDMA を用いた片方向通信ライブラリである ARMCI と性能をベンチマークプログラムを通じて比較し，Telemem の性能が ARMCI を上回る場合があることを示す．

A Comparison on One-Sided Communication Systems

Atsushi Hori,[†1] Jinpil Lee[†1] and Mistuhisa Sato[†1]

One-sided communication is thought to have an affinity with Remote Direct Memory Access (RDMA) done by a network interface hardware, and is attracting attentions as a way of implementing scalable high-performance communication system. In this paper, some points to a implement one-sided communication system will be listed and then it will be pointed out that a one-sided communication system without using RDMA but having low-level message communication layer can be implemented easily, avoiding some of the implemenation issues. The proposed one-sided communication system is implemented and named “Telemem,” and evaluated with ARMCI one-sided communication system using RDMA. Through the benchmark evaluations, it will be shown that the application performance with Telemem may overcome the performance with ARMCI.

†1 筑波大学 / Tsukuba University

1. はじめに

メッセージ通信（両方向通信）は，例えば MPI の通信を例にすると，送信プロセスの送信処理と受信プロセスの受信処理は 1 対 1 の関係が基本になっている．これに対し，片方向通信では，通信を起動するイニシエータプロセスと通信の対象となるターゲットプロセスがあって，データの流れる向きはイニシエータが一方的に決め，ターゲット側では明示的になんら対応する処理は行わないという特徴がある．

具体的には，片方向通信において，イニシエータはターゲットのメモリをアクセスする要求をターゲットプロセスに向かって発行する．ターゲットのメモリの内容を要求する処理は get（操作），ターゲットのメモリの内容を変更する処理は put（操作）と呼ばれている (図 1)．通信する 2 つのプロセスを，イニシエータとターゲットという名前で区別するのは制御の観点である．データ流という観点から，ソースとデスティネーションとして流れの起点と終点を区別する場合もある．

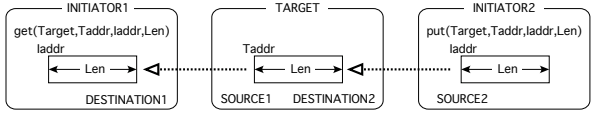

図 1　Get/Put におけるイニシエータとターゲット

片方向通信は，ハードウェアによる RDMA（Remote Direct Memory Access）との相性が良く，特に最近の並列計算の分野では，新しい並列計算モデル Partitioned Global Address Space（PGAS）[1),6),16)] における下位通信レイヤーとしても注目されている．また，通信ハードウェアの性能を引き出す通信方式として，並列計算の通信方式だけでなく，広く高速な通信方式としても注目されている[12)]．

しかしながら，RDMA をサポートするハードウェアの必要性，対象メモリ領域を論理アドレスと物理アドレスのマッピングを固定するためのピンダウンの必要性，などの実装上の問題点があり，オープンソースで使える実装は ARMCI[9)] と GASNet[3)] の 2 つしかないといっても過言ではない状況である．

本稿では，最初に片方向通信を実装する際の問題点を挙げ，RDMA と一見相性が良く見えるが，実装上の問題点が少なくないことを示す．次に，実システムとして ARMCI と GASNet の実装を概観し，その上で，従来のハードウェア RDMA に依存した片方向通信

の実装ではなく，両方向通信のメッセージ通信を下位レイヤーとする片方向通信の実装方式を考え，実装上の問題の多くが解決できることを示す．このようなシステムは実際に実装され，既存の RDMA ベースの ARMCI とともに NetPipe[15] を HPC Challeng の FFT[7] を用いて評価，比較される．

MPI-2 にも片方向通信の API が定義されているが，文献[4] で指摘されているように，例えば PGAS モデルの下位通信レイヤーとして用いるのは問題点が多い．本稿では，ARMCI や GASNet のような片方向通信を対象とし，MPI-2 における片方向通信は本稿の範囲外とする．

2. 片方向通信の実装のポイント

本章では，片方向通信の実装における重要と思われる点について検討する．

2.1 Pin-down

ハードウェアによる RDMA は，ノード内の処理だけを見れば DMA（Direct Memory Access）であり，ハードウェア（多くの場合はネットワークインターフェイス）が CPU の介在なしに直接メモリに書込む．この時のアドレスは通常物理アドレスである．メモリページの物理アドレスと論理アドレスの対応は，スワップされると変わってしまうため，基本的に DMA の対象となるメモリページはスワップを禁止しなければならない．このスワップの禁止は，pin-down とも呼ばれ，pin-down 処理はカーネルのページテーブルの変更と，それに伴う TLB のフラッシュが必要で実行コストが高い．このため RDMA を用いる多くの実装では，pin-down されたメモリページをキャッシュ[13] する機構を設け，ページ属性の変更を最小限に抑えるようにしている．

2.2 ス レッド

片方向通信の最大の特徴はターゲット側の処理が明示的でないことである．実際にはイニシエータで発行された要求を処理するため，なんらかのスレッドが必要となる．ハードウェアによる RDMA では，例えば Myrinet[2] では NIC 上の LANai プロセッサ上で RDMA の要求を処理するスレッドが走っていると考えることができる．ただし，ハードウェアのスレッドを仮定すると，特定のハードウェア上でしか片方向通信が実現できなくなってしまう．ハードウェア依存性を無くすためには，計算とは別のプロセスあるいはスレッドを作り，そこでイニシエータの片方向通信要求を処理する方式がある．

2.3 メッセージ通信

片方向通信におけるターゲット側は見掛け上通信になんら関与しないのが片方向通信の特徴であるが，実際にはイニシエータによるターゲットメモリのアクセスの完了を知る必要があるなど，内部的に（両方向）メッセージ通信を実装している場合が多い．例えば，片方向通信では事前にターゲットプロセスの対象となる領域のアドレスを知る必要があるため，メッセージ通信でメモリアドレスの受け渡しをする場合がある．また，ターゲットプロセスはイニシエータによる get の対象領域を更新するのは，get の後である必要があり，put の後でないと更新されたデータを知ることはできないからである．

イニシエータ側のアクセスの完了をターゲット側に伝えるためである．通信用のフラグ領域を予め確保しておき，put でそのフラグをセットし，イニシエータ側はそのフラグがセットされるまで待つ，というやり方もあるが，片方向通信処理のための専用サーバスレッド／プロセスでは，このフラグをビジーループで待つことになり，他の計算プロセスあるいはスレッドの実行を阻害してしまう．メッセージの送信順序が保存されないネットワークの場合，最後のアクセスの終了を待ってからフラグをセットしなければならないため，レイテンシが悪化する．

また，多くの片方向通信ライブラリでは，バリア同期の機能が提供されており，get/put の完了後に，バリア同期することで，完了が分かるようになっている．バリア同期を片方向通信だけで効率的に実装するのは難しい．結果的に，多くの片方向通信ライブラリで，内部にメッセージ通信機構を持っている．実装によっては内部で使っているメッセージ通信をユーザに見せているものもある．

2.4 不連続領域の Get/Put

RDMA による片方向通信は，pin-down 処理や DMA の設定等のオーバヘッドにより，小さいメッセージの転送バンド幅が低下する傾向にある．このため不連続で細粒度のメモリ領域をそのまま複数の RDMA とすると，バンド幅が著しくて低下する場合がある．これを回避するためには，不連続な領域を pack してから転送し，デスティネーション側で unpack する方式がある．

Infiniband のオープンソースライブラリの OpenIB[10] には，pack して送信する”gather send” と受信してから unpack する”scatter receive” が提供されているが，ソース側でデスティネーション側の unpack 方法を規定することができない．このため，pack されたメッセージの送信と，デスティネーション側で対応する unpack を記述した受信と正しく対応することを保証しなければならない．このため，gather send と scatter receive で不連続領域の get/put を実現するには別途プロトコルが必要となる[14]．

ユーザがプログラムの中で不連続領域の pack/unpack を記述する場合を考える．Pack

されたデータを受信した場合，そのデータを unpack する必要がある．Put の場合，このためターゲット側で RDMA の完了をなんらかの方法で待ち，受信完了後に unpack する必要が生じてしまう．Put のターゲット側は逐一受信の処理をおこなわないのが原則なので，もはや片方向通信とは呼べなくなってしまう．

3. 片方向通信ライブラリの実装例

3.1 ARMCI[9)]

Aggregate Remote Memory Copy Interface (ARMCI)[9)] はノードにひとつのスレッド（プロセスの場合もある）を立ち上げ，片方向通信サービスのサーバとしている[5)]．このため，ユーザプログラムのプロセスあるいはスレッドの動きを邪魔しないよう，このスレッドは blocking 受信を基本とする．下位通信レイヤーがこれを許さない場合は，ポーリングと自発的なプロセス切替を促し，ユーザのプロセスやスレッドにできるだけ多くの CPU 時間が割り当てられるよう努力する必要がある．一方，このサーバスレッドがあるおかげで不連続量の get/put のための pack/unpack が可能であり，ARMCI では不連続領域の get/put がサポートされている．下位の通信レイヤーとしては，様々ネットワークや MPP をサポートしているが，ドキュメントに乏しく，下位通信レイヤーの API が明確にはなっていない．

3.2 GASNet[3)]

GASNet[3)] は “Global-Address Space Networking” の略でありカリフォルニア大学バークレー校（UCB）で開発された片方向通信ライブラリである．下位通信レイヤーとしては同じ UCB で開発された AM (Active Messages) メッセージ通信ライブラリが使われており，GASNet のコア API として AM がユーザに見えている．通信ネットワークは conduit と呼ばれる API が規定されている．AM の仕様上，ポーリング関数を適時呼ぶ必要がある．Infiniband の実装では，Infiniband conduit で定義されているポーリング関数で，RDMA の CQ（Completion Queue）の管理をおこなっている．またポーリング関数があるため，下位の通信ネットワークあるいはライブラリに制約が少なく，MPI の conduit も用意されている．

GASNet の片方向通信では，大きなメッセージの場合，ネットワークに RDMA 機能がある場合は，その RDMA 機能を用いて get/put を実装し，小さなメッセージや RDMA 機能がサポートされていない場合には AM によるメッセージ通信で片方向通信が実現されている．

Conduit によってはポーリング関数が内部的にスレッドになっている場合もある．この場合 GASNet のポーリング関数は nop になっている．多くの場合，GASNet の関数を呼び出し時に自動的にポーリング関数が呼び出されているため，ユーザがプログラム中に明示的にポーリング関数を呼ぶような機会は多くないと考えられる．

4. ポーリングベースの片方向通信

GASNet は ARMCI と並び，実用的な片方向通信ライブラリである．前述したように，GASNet はポーリングをベースとしている．もしポーリングが片方向通信ライブラリで許容されるのなら，RDMA を使わずにポーリングベースで片方向通信も実現可能となる．逆に，ポーリングベースの片方向通信が，RDMA を使わずにメッセージ通信を上回る性能を発揮できたとすると，メッセージ通信（両方向通信）と片方向通信の差異は API ということになる．

以下，RDMA を用いた片方向通信の実装を「RDMA ベースの片方向通信」と呼び，下位通信レイヤーがポーリングベースのメッセージ通信で API が片方向通信の実装を「ポーリングベースの片方向通信」と呼ぶことにする．

ポーリングベースの実装の利点は，

- RDMA 機能を持たない通信ネットワークでも片方向通信を実現可能
- バリア等の同期に必要なメッセージ通信機構を別途必要としない
- 不連続領域のための pack/unpack が容易に実現可能
- get/put の対象となるメモリ領域を pin-down する必要がない

である．一方，欠点は，

- CPU がメッセージをコピーするため，その分，通信と計算がオーバラップできる割合が減る

となる．ポーリングベースの場合は適時ポーリング関数を呼ぶ必要があるが，RDMA ベースでも GASNet のように呼ぶ必要があるものがあるため，必ずしもポーリングベースの欠点とは言えない．

RDMA ベースの実装では，MPI に比べ通信性能が高いという主張がある（例えば文献[1)]）．この点については次節で検討したい．

4.1 通信のデータフロー

図 2 は典型的なコンピュータのデータパスを示したものである．CPU は基本的にはキャッシュを通じてメモリにアクセスし，通信メッセージは DMA によりデバイスとメモリとの間で転送される．ここで注意すべきはキャッシュとメモリの関係である．キャッシュの内容

はメモリの内容を反映しているので，DMA によりメモリの内容が変更されるとキャッシュの内容と矛盾してしまう．このような矛盾を生じないようにしているアーキテクチャ（x86 系）では，DMA によりメモリが書込まれる場合は，そのアドレスに対応するキャッシュエントリを invalidate する．逆に DMA によりメモリの内容がデバイスにより読み出される場合は，DMA に先立ちキャッシュにある dirty なエントリをメモリに掃出する．

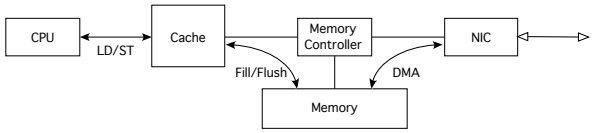

図 2　通信におけるデータフロー

ここで通信メッセージの受信について考える．NIC によって受信されたメッセージは DMA によりメモリに書込まれる．ここまでは RDMA でもメッセージ通信でも同じである．RDMA の場合はこれでメッセージの受信は終了である．一方，メッセージ通信の場合，メモリの内容は最終的なメモリ領域に memcpy() される．

受信されたメッセージの多くはその後の計算に使われることが多い．このため受信メッセージがキャッシュに載っていた方がより高速に計算できる．RDMA では受信されたメッセージを DMA する際，結果的に受信したメッセージの全てがキャッシュから外れてしまう．この状況はメッセージ通信でも同じではあるが，memcpy() によりキャッシュにデータが載る可能性がある．また，受信バッファの総量よりも大きな領域を get あるいは put した場合，RDMA ではその全てがキャッシュから外れてしまうのに対し，メッセージ通信でキャッシュから外れるのは受信バッファの総量に等しい量が上限となる．

送信されるメッセージが DMA でデバイスに送られる時，キャッシュに載っていた部分は必要に応じてメモリに書き出されるだけであり，実行時間に対し大きな影響は及ぼさないと考えられる．

RDMA による通信が高性能であるとの主張の根拠のひとつは，memcpy() の必要がなく，memcpy() に要する時間だけ短縮される，というものである．しかしながら，I/O バスのバンド幅はメモリバスのバンド幅に劣るのが常であるため，メモリバスが I/O バスあるいはネットワークの速度に比較して十分に大きい場合は memcpy() は大きな性能低下には結びつかない．受信したデータの全てがキャッシュから外れてしまう状況は，受信後の計算速

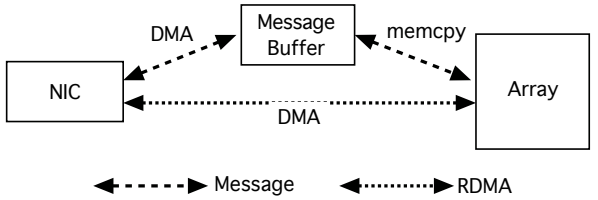

図 3　通信時の DMA

度の低下を招くため，通信だけの時間で RDMA とメッセージ通信の性能を云々するのは不公平であり，通信と計算を含めた時間で議論すべきと考える．その結果として，並列アプリケーションの実行時間が，ポーリングベースの片方向通信を用いた方が RDMA ベースの場合よりも速くなる可能性はあると考えられる．

4.2 Telemem

これまで議論したように，ポーリングベースの片方向通信は実装上の利点があるだけでなく，性能的にも RDMA ベースに匹敵する可能性があると考えることができる．本説では新たに開発したポーリングベースの片方向通信ライブラリである Telemem の概要について記す．

Telemem は SCore[11] の PMX メッセージ通信ライブラリを下位レイヤーとする片方向通信ライブラリである．PMX はネットワークでバイスとして Ethernet，Myrinet（MX），Infiniband (OFED) をサポートする，通信デバイスに依存しない共通の API を提供する．このため Telemem も PMX がサポートするネットワークを全てサポートする．表 1 に Telemem の主要な API を記す．

表 1　Telemem の主要 API

telemem_initialize()	初期化
telemem_finalize()	終了
telemem_get()	同期 get
telemem_put()	同期 put
telemem_quiet()	これ以前の put の完了待ち
telemem_barrier()	バリア同期
telemem_poll()	ポーリング関数

Telemem はまた ARMCI や GASNet 等と等価なヘッダーファイルが提供されており，

Vol.2010-HPC-126 No.12
2010/8/3

これらの片方向通信ライブラリを用いて記述されているプログラムが Telemem を使うように変更する手間が少なくなるようになっている．

5. 評　　価

評価に用いたクラスタは Nehalem Xeon（L5520），クロック 2.27 GHz，L3 キャッシュは 8 MB を 4 ノード用いた．ネットワークは Mellanox の ConnectX QDR である．以下の評価では，ノード間通信の特性に注目するため，ノード内のプロセス数は 1 とした．MPI は SCore[11] Version 7 のものを用いた．Telemem は同じ SCore に含まれている PMX を使用した．比較のために用いた ARMCI[9] は GlobalArray Tool Kit 4.2 に付属するものである．

5.1 NetPipe[15]

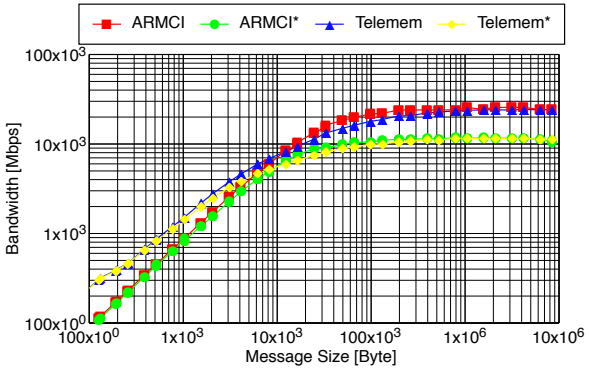

図 4　バンド幅

図 4 は NetPipe[15] を使い，Telemem と ARMCI のバースト put のバンド幅を計測したものである（横軸：メッセージサイズ，縦軸：バンド幅）．”ARMCI*” および “Telemem*” とあるのは，NetPipe を改修し，メッセージ送出の前とメッセージ受信の後に，メッセージの領域を全て参照（読込み）した場合のグラフである．

図 5 は ARMCI と Telemem のバンド幅の違いを明確にするために，図?? と同じデータを使い，Telemem のバンド幅を 1 としたときの ARMCI の相対バンド幅を表したものである．”A/T” とあるのはメッセージの領域を触らなかった場合，”A*/T*” をあるのはメッセージの領域を触った場合である．

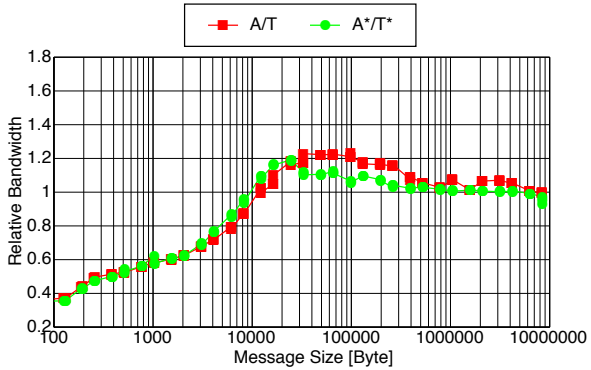

図 5　Telemem に対する ARMCI の相対バンド幅

図 4 と図 5 から，ARMCI はメッセージサイズが 10KB 以下の小さいメッセージで Telemem よりバンド幅が大きく劣るが，それより大きい場合では Telemem を上回っている．RDMA のためのオーバヘッドのためと考えられる．しかし，メッセージ領域を参照した場合には ARMCI の性能的な優位は小さくなっていることが分かる．

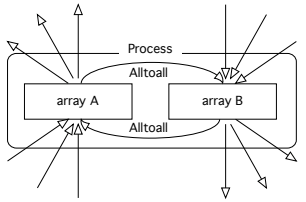

図 6　FFT のデータの流れ

5.2 FFT

ここでの評価に用いたプログラムは HPC Challenge[7] の FFT である．このプログラムのデータの流れを図 6 に示す．この FFT では計算に用いる配列が 2 つあり，alltoall 通信の都度，配列が入れ替わるように計算をおこなっている．Alltoall の集団通信では，配列の大きさの $(N-1)/N$，N はプロセス数，だけのデータ量が通信の対象となる．なお，alltoll

Vol.2010-HPC-126 No.12
2010/8/3

は 3 回呼び出される．MPI と Telemem のメッセージバッファは送信，受信それぞれで 16 KB のバッファを 256，バッファ総量は 4 MB で L3 キャッシュの半分の量に相当，持つように設定した．

オリジナルのコードは MPI で記述されているが，alltoall の通信を Telemem と ARMCI の片方向通信を用いて図 7 のようにナイーブに記述した．単純なアルゴリズムにしたのは，凝った方式では MPI との差異が出難いと考えたからである．また配列の大きさをプログラムのオプションで変更できるようにオリジナルのコードを改変した．

```
void alltoall( void *snd, void *rcv, size_t blk ) {
  int  n, to from;
  for( n=0; n<nranks; n++ ) {
    if( n == myrank ) {
      from = snd + ( blk * myrank );
      to   = rcv + ( blk * myrank );
      memcpy( to, from, blk );
    } else {
      from = snd + ( blk * n );
      to   = get_raddr( rcv, n ) + ( blk * myrank );
    }
#ifdef TELEMEM
    telemem_put( n, to, from, blk );
#endif
#ifdef ARMCI
    ARMCI_Put( n, to, from, blk );
#endif
  }
#ifdef TELEMEM
  telemem_quiet();
  telemem_barrier();
#endif
#ifdef ARMCI
  ARMCI_AllFence();
  ARMCI_Barrier();
#endif
}
```

図 7　ナイーブな実装の Alltoall

図 8 は横軸に問題の大きさ（配列のサイズ），縦軸に Telemem，MPI，ARMCI のそれぞれで実行したときの FFT の計算速度（HPCC FFT が出力した値）をプロットしたグラフである．ARMCI では問題サイズが 1,048,576 を超えたとたんに 2 GFLOPS 以下になっている．この問題サイズは，2 つの配列の大きさの和をバイト数に換算すると 8 MB であり，ちょうど L3 キャッシュの値と一致する．このため第 4.1 節で述べたように，RDMA がキャッシュに悪影響を及ぼしたものと想像される．一方，Telemem と MPI ではキャッシュのサイズの前後で大きな変化は見られない．

図 9 は，実行時間から alltoall 通信の時間を引いた時間を計算し、これは計算のみの時間に相当する，MPI の時の値を 1 とした時の，ARMCI と Telemem の相対時間を示したグラフである．図 8 で見られたように，問題サイズが 1,048,576 を超えたとたんに実行時間がおよそ 1.5 倍になっているのが分かる．

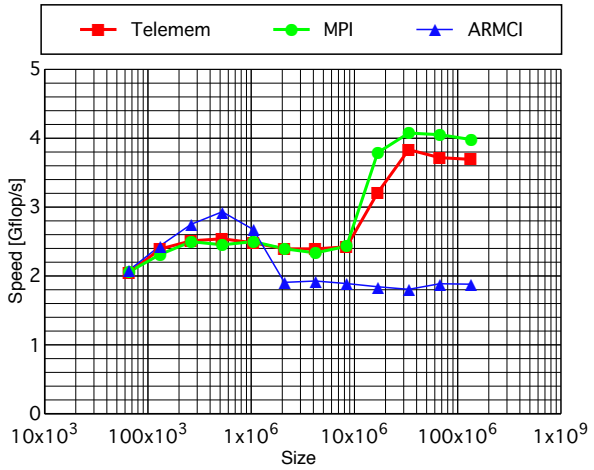

図 8　FFT の性能

図 8 では Telemem と MPI の計算速度が問題サイズが 16,777,216 を超えたところから急に計算速度が速くなっている．図 9 でも同じところから ARMCI との計算時間の比がさらに遅くなっている．図 8 ではこの問題サイズで ARMCI の計算速度に大きな変化は見られないことから，Telemem および MPI に対し何かが作用していることが考えられるが，残念ながら現時点では理由は不明である．

図 10 は FFT の計算時間（実行時間から alltoall の時間を除いた値）に対する alltoall の時間の比を示したグラフである．問題サイズが大きくなるにつれ，ARMCI は Telemem よりも alltoall が占める割合が大きく減っているのが分かる．図 8 と併せて解釈すると，計算時間が大きくなっていると判断される．一方，Telemem は MPI に比べ問題サイズが大きい時に alltoall の時間の割合が増えている．これはナイーブな実装の alltoall の影響と推測される．

6. 考　　察

NetPipe および FFT の検証結果から，RDMA ベースの片方向通信では，通信性能では

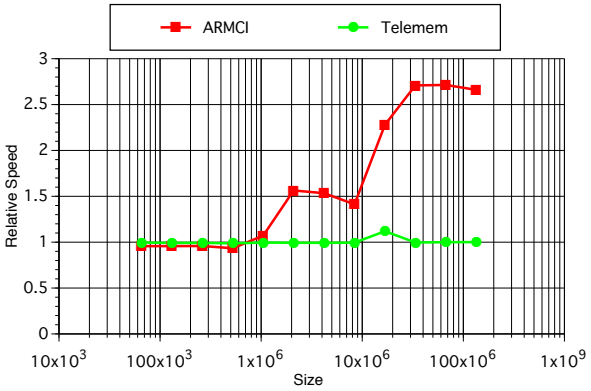

図 9　MPI を 1 とした時のの計算時間の比

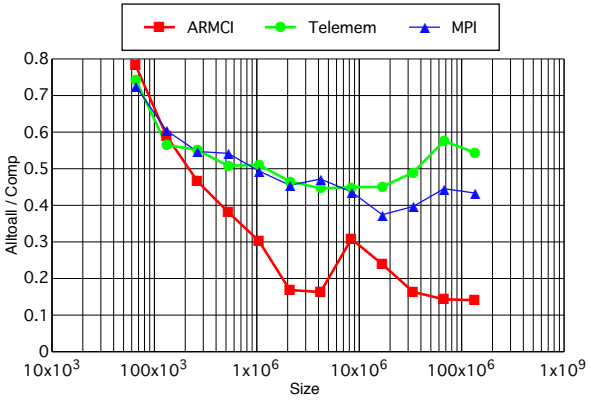

図 10　FFT の計算時間に対する alltoall 時間の割合

ポーリングベースを上回るが，アプリケーションの実行時間を含めた場合にはポーリングベースの性能が上回る場合があることが確認された．この理由のひとつとして RDMA がキャッシュに与える悪影響が考えられるが，今回の検証結果からはそれを立証するまでには至っていないと考える．別途，PAPI[8] などを用いた検証が必要と考える．また，RDMA ベースとポーリングベースのどちらが最終的なアプリケーションの性能が良いかは，アーキテクチャの違いやアプリケーションの特性からくるいくつかのパラメータの違いの影響を受ける物と推測される．今後，様々なアーキテクチャやアプリケーションでの評価を通じ，何がどのように影響を及ぼすのかをより明確にすべきと考える．

一方で，MPI とポーリングベースの Telemem では下位通信レイヤーが同じということもあり，大きな性能の違いを見いだすことができなかった．逆にいえば，ポーリングベースの片方向通信は，メッセージ通信の API を変えたものという程度の違いしかないという結論もあり得る．しかしながら，ポーリングベースの片方向通信がハードウェアではないソフトウェア主導の実装方式であり，ハードウェアの制約に縛られない新たな API を導入することが可能であり，これにより MPI を上回る性能を発揮することが可能と考える．

おわりに

本稿では，片方向通信を実装する際に問題となるポイントを挙げ，既存の RDMA を用いた片方向通信の実装方式の調査結果から，下位通信レイヤーとしてメッセージ通信を用いた，ポーリングベースの片方向通信が，それらの問題点のいくつかを解決すると同時に，RDMA を用いた場合よりもアプリケーションの実行において，高い性能を発揮する可能性について検討した．同時に，提案されたポーリングベースの片方向通信ライブラリを実装し，RDMA ベースの片方向通信ライブラリ ARMCI とベンチマークプログラムを用いて評価した．その結果，通信性能では ARMCI に劣る場合があっても，特にキャッシュに収まらないような大きな問題の実行性能において，RDMA ベースの片方向通信において，計算時間がポーリングベースの場合に比べ著しく遅くなる場合があり，その結果，ポーリングベースの片方向通信の方が高いアプリケーション性能を示すことが示された．

今回の結果から，ポーリングベースの片方向通信が，常に RDMA ベースの片方向通信の性能を上回るという結論には至らない．しかし，片方向通信は必ずしも RDMA を用いなくても高いアプリケーション性能を発揮できるという点について実証できたと考える．今後，検証を重ね，ポーリングベースと RDMA ベースの性能の境界を明確にしていきたいと考える．

Vol.2010-HPC-126 No.12
2010/8/3

謝　　辞

本研究の一部は文部科学省「e-サイエンス実現のためのシステム統合・連携ソフトウェアの研究開発」からの支援を受けている．

参 考 文 献

1) Bell, C., Bonachea, D., Nishtala, R. and Yelick, K.: Optimizing Bandwidth Limited Problems Using One-Sided Communication and Overlap, *IPDPS 2006* (2006).
2) Boden, N.J., Cohen, D., Felderman, R.E., Kulawik, A.E., Seitz, C.L., Seizovic, J.N. and Su, W.-K.: Myrinet: A Gigabit-per-Second Local Area Network, *IEEE Micro*, Vol.15, No.1, pp.29–36 (1995).
3) Bonachea, D.: GASNet Specification, v1.1, Technical Memorandum UCB/CSD-02-1207, University of California (2002).
4) Bonachea, D. and Duell, J.: Problems with using MPI 1.1 and 2.0 as compilation targets for parallel language implementations.
5) Buntinas, D., Saify, A., Panda, D.K. and Nieplocha, J.: Optimizing Synchronization Operations for Remote Memory Communication Systems, *IPDPS'03* (2003).
6) Coarfa, C., Dotsenko, Y., Mellor-Crummey, J., Cantonnet, F., El-Ghazawi, T., Mohanti, A., Yao, Y. and Chavarría-Miranda, D.: An evaluation of global address space languages: co-array fortran and unified parallel C, *PPoPP '05: Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, ACM, pp.36–47 (2005).
7) Innovative Computing Laboratory: HPC Challenge. http://icl.cs.utk.edu/hpcc/software/.
8) Innovative Computing Laboratory: Performance Application Programming Interface. http://icl.cs.utk.edu/papi/.
9) Nieplocha, J., Tipparaju, V., Krishnan, M. and Panda, D.K.: High Performance Remote Memory Access Communication: The Armci Approach, *Int. J. High Perform. Comput. Appl.*, Vol.20, No.2, pp.233–253 (2006).
10) OpenFabrics Alliance: OFED. http://www.openfabrics.org/.
11) PC Cluster Consortium: SCore. http://www.pccluster.org/.
12) RDMA Consortium: . http://www.rdmaconsortium.org/.
13) Tezuka, H., O'Carroll, F., Hori, A. and Ishikawa, Y.: Pin-down Cache: A Virtual Memory Management Technique for Zero-copy Communication, *IPPS/SPDP*, IEEE Computer Society (1998).
14) Tipparaju, V., Santhanaraman, G., Nieplocha, J. and Panda, D.K.: Host-Assisted Zero-Copy Remote Memory Access Communication on InfiniBand, *Parallel and Distributed Processing Symposium, International*, Vol.1, p.31a (2004).
15) Turner, D., Oline, A., Chen, X. and Benjegerdes, T.: Integrating new capabilities into NetPIPE, *Lecture Notes in Computer Science*, Springer, pp.37–44 (2003).
16) 李珍泌，佐藤三久：分散メモリ向け並列言語 XcalableMP コンパイラの実装と性能評価，先端的計算基盤システムシンポジウム SACSIS 2010 (2010).