Vol.2016-HPC-157 No.5
2016/12/21

Tofu2プロトコルオフロード機能を使ったMPI永続隣接集団操作のプロトタイプ実装と評価

畑中 正行[1]　小倉 崇浩[1]　高木 将通[1]　堀 敦史[1]　石川 裕[1]

概要：本稿では，FX100 Tofu2 インターコネクトに新しく導入された Session モードと呼ばれるプロコトル・オフロード機能を利用し，集団通信における計算と通信のオーバーラップを改善する手法を設計・実装する．本モードのコマンドキューでは，通常の FIFO ポインタに加え，通信スケジューリングを制御するためのセッション・オフセットと呼ばれるポインタが追加される．これを利用することで CPU の介在なしに Tofu2 インターコネクト側で通信の同期や契機が実現可能となる．集団通信のうち，MPI Forum によって策定中の MPI-4 版候補である永続集団通信の隣接集団操作を対象とする．一次元 3-point ステンシル計算の袖通信での評価結果，メッセージ長 16KiB で約 5 [us] のレイテンシが得られた．

Prototype Implementation and Evaluation of an MPI Persistent Neighborhood Collective Operation Using Tofu2 Protocol Offlaoding Capability

Masayuki Hatanaka[1]　Takahiro Ogura[1]　Masamichi Takagi[1]　Atsushi Hori[1]
Yutaka Ishikawa[1]

Abstract: In this paper, we design and implement an improvement method of overlap between computation and communication using a protocol offloading capability, called session mode, in the FX100's Tofu2 interconnect. In the session mode, in addition to FIFO read and write pointers in a normal command queue, an additional pointer, called a session offset, has been introduced in order to control scheduling of the queue. Using this feature, the basic building blocks of collective communications such as synchronization protocols and triggers of events can be implemented. One of the persistent neighborhood collective operations, proposed to the MPI-4 standard at the MPI Forum, is implemented using the Tofu2 session mode. The execution of a communication pattern exchanging ghost regions in one dimensional 3-point stencil computation is evaluated using the proposed method. It shows that our implementation achieves about 5 microseconds for data exchange of 16 KiB.

1. はじめに

近年，HPC アプリケーションにおける MPI 通信の通信レイテンシの向上に対する要求はますます高くなっている．それを受けて MPI 標準もさまざまな機能追加が図られている．例えば，計算と通信のオーバーラップを促進することは，実質的に通信を隠蔽する効果があり，MPI-3.0 仕様で非ブロッキング集団通信が追加された．また，袖通信のように，ある決まった複数の相手との特定の通信パターンが繰返し発生するケースでも，`MPI_Isend` および `MPI_Irecv` のばらばらな呼出しを避け，集団通信化することで，複数の呼出しオーバヘッドが削減できるため，MPI-3.0 仕様で隣接集団通信が追加された (非ブロッキング型を含む)．それにより別な利点として，最適化が効くか否かは別であるが，ユーザ独自の集団通信を作成できるようになった．さらに，現在策定中の MPI-4.0 仕様では，永続集団通信が検討されている．これも，毎回の集団通信の引数チェック，通信資源割当て，および相手との通信パラメタの交換等，多くのオーバーヘッドを削減できると期待されている．これらにより，毎回の煩雑なプロトコル交換は不要になり，

[1] 理化学研究所
RIKEN

MPI プロセス間の同期と RDMA データ転送機能さえあれば，集団通信プロトコルのほとんどをネットワーク・ハードウェアにオフロードできると見込まれている．

本稿の概要は次のとおりである．

(1) FX100 Tofu2 のプロトコルオフローディング機能に対する，永続型隣接集団プロトコルの写像方式の提案 (高速型)
(2) 当該ハードウェア資源の枯渇時に使われる，低資源消費型の写像方法の提案
(3) 1D 3-point ステンシル計算の袖領域交換のベンチマークによる高速型の提案手法の基本性能と性能妥当性の検証: 16 KiB のメッセージ長で 約 5 [μ s] のレイテンシを確認
(4) 同ベンチマークによる低消費型の提案手法の基本性能: 16 KiB のメッセージ長で高速型の約 1.5 倍の劣化に抑えられることを確認

以降では，節 2 で，非同期通信に対する関連研究について述べ，節 3 で，MPI の基本的な用語について概説する．次に，節 4 で，プロトコルをハードウェアにオフロードするための提案する写像方法について説明し，節 5 で，その実装に対する予備的評価について述べる．最後に節 6 で，まとめと今後の課題について述べる．

2. 関連研究

Caspter [1] は，近年のメニーコアを採用したシステムで，アプリケーションで使い切れない CPU コアを通信プログレス用のスレッドとして使用するアプローチである．また，Blue Gene/Q や PRIMEHPC FX100 のように，OS やシステム・ソフトウェア専用のコアを装備したシステムもある．いずれのアプローチも，CPU クロックが漸減傾向にあるなか，階層化された CPU によるマルチスレッドのプロトコル処理では，通信レイテンシを大幅に向上することが難しくなってきている．

京の Tofu Barrier Interface のように，CPU を介さず，複雑な集団通信をよりインターコネクトに近いところで処理することでレイテンシを改善する，ネットワーク・インターフェイス・レベルの試みは以前よりなされてきた．しかし，主に barrier や reduce や bcast に限られていた．これに対し，Schneider ら [2] は，Portals 4.0 API を使用して，さまざまな集団通信をハードウェアにオフロードできることを示した．

上述の FX100 では，Portals ほどではないものの，やや汎用化されたオフローディング機能，Session モードの制御キュー (CQ)，をサポートしている．本稿では，Session モード CQ を使った永続型の隣接集団通信の実装について述べる．

3. 背景

3.1 隣接集団通信

例えば, 二次元 5-point ステンシル計算での袖通信に対し，既存の `MPI_Alltoallw` を使うことも机上では可能であったが，その引数 *comm* に属する全プロセスについて，*sendcounts[]* や *recvcounts[]* 等の各配列を完全に埋めなければならない．従って，ジョブあたり万オーダ以上の MPI プロセスが起動するような近年のシステムでは，これらの配列のサイズはまったくスケーラブルでなかった．後述するプロセス・トポロジ生成関数 `MPI_Dist_graph_create_adjacent` を使って，ユーザが実際に通信が発生するプロセスを明示し，その通信トポロジ情報を (関係づけされたコミュニケータを介し) 使用することで，隣接集団操作はこの問題を解消している．

具体的には，`MPI_Neighbor_` で始まる集団操作は，MPI-3.0 標準 [3] で新たに導入された隣接集団操作であり，`MPI_Neighbor_allgather*` および `MPI_Neighbor_alltoall*` の 2 種類の集団操作がサポートされる．これらの操作の引数は，対応する従来の `MPI_Allgather*` および `MPI_Alltoall*` と同一である．実際，図 1 に示すように，`MPI_Neighbor_alltoallw` は，通信相手ごとに，バッファの先頭，データ型，データ型の個数を送信および受信それぞれに指定する．しかしながら，従来の `MPI_Alltoallw` と異なり，指定するコミュニケータ `comm` は，通信するプロセスのトポロジが決定できるコミュニケータでなければならない．例えば，`MPI_Cart_create` で生成された `MPI_CART` 型のトポロジや，`MPI_Dist_graph_create_adjacent` で生成された `MPI_DIST_GRAPH` 型のトポロジをもつコミュニケータでなければならない．

```
int MPI_Neighbor_alltoallw(
  const void *sendbuf, const int sendcounts[],
    const MPI_Aint sdispls[], const MPI_Datatype sendtypes[],
  const void *recvbuf, const int recvcounts[],
    const MPI_Aint rdispls[], const MPI_Datatype recvtypes[],
  MPI_Comm comm);
```

図 1 MPI_Neighbor_alltoallw の形式

3.2 プロセス・トポロジの生成

前節 3.1 で述べたように，隣接集団通信操作に指定するコミュニケータはプロセス・トポロジが設定されていなければならない．そうしたコミュニケータを生成するトポロジ生成関数は複数提供されているが，すべて集団呼出しであることに注意が必要である．ここでは，不規則な通信トポロジも表現でき，`MPI_Isend` および `MPI_Irecv` を使った既存コードとの対応づけ容易な

MPI_Dist_graph_create_adjacent を例にプロセス・トポロジつきのコミュニケータと MPI_Neighbor_alltoallw の引数との関係を説明する．

MPI_Dist_graph_create_adjacent は，通信トポロジを分散形式のグラフで記述する．つまり，頂点 (vertex) を MPI プロセス，有向の辺 (edge) をメッセージとし，各プロセスの MPI_Dist_graph_create_adjacent はサブグラフを形成し，その集団呼出し全体で，一つの分散グラフを形成する．この分散グラフでは，重複辺，自己ループ，および孤立点が可能である．本関数は図 2 に示すように，呼出しプロセスに対する流入辺 (incoming edge) と流出辺 (outgoing edge) との情報を指定する．

```
int MPI_Dist_graph_create_adjacent( MPI_Comm comm_old,
  int indegree, const int srcs[], const int srcweights[],
  int outdegree, const int dsts[], const int dstweights[],
  MPI_info info, int reorder, MPI_Comm comm_dg);
```

図 2 MPI_Dist_graph_create_adjacent の形式

例えば，MPI_Isend および MPI_Irecv を使った一式の通信があった場合，引数 *indegree* は MPI_Irecv の数，*srcs[]* は MPI_Irecv の送信元プロセスのランク (source) を要素とする *indegree* 個の配列である．同様に，引数 *outdegree* は MPI_Isend の数，*dsts[]* は MPI_Isend の送信先プロセスのランク (dest) を要素とする *outdegree* 個の配列である．

また，本関数を使用して生成されたコミュニケータ *comm_dg* を MPI_Neighbor_alltoallw に指定した場合，その引数 *sendcounts[]* や *recvcounts[]* 等の送信および受信配列はそれぞれ，*dsts[]* および *srcs[]* の配列要素に対応した MPI_Isend および MPI_Irecv の引数を指定しなければならない．

3.3 隣接集団通信の課題

図 3 の例を検討する．この例は隣接集団通信操作 MPI_Neighbor_alltoallw を 100 回呼出すループである．ランク 0 以 外は MPI_Neighbor_alltoallw が同じ引数で 100 回呼ばれる．これに対しランク 0 は 11 回め ($i == 10$) に，送受信のバッファを bufa から bufb に変更する．

こうしたケースをサポートするために典型的には，MPI_Neighbor_alltoallw の内部実装において，ユーザ・バッファ間で直接の RDMA 転送を行うとき，転送の前に毎回 RDMA メモリアドレスの交換プロトコルを実行する必要がある．しかしながら，同じ引数で何度も集団操作が呼出される場合，この RDMA メモリアドレスの交換を一度だけに省略することが可能であり，事前に繰返しの呼出しであることを知ることができれば，通信レイテンシの面で改善が見込める．不幸にも，プログラマからの指示やコンパイラによる自動検出がない限り，リモートの通信相手の通信パラメタが変更されたか否かを実行中に低レイテンシで知る方法は一般にはない．

```
int ii, ni = 100, rank;

MPI_Comm_rank(comm, &rank);
for (ii = 0; ii < ni; ii++) {
  computeA(bufa); /* update bufa */
  if ((rank == 0) && (ii == 10)) {
    copy_buf(bufb, bufa); /* from bufa to bufb */
    MPI_Neighbor_alltoallw(bufb, ..., bufb, ...  , comm);
    copy_buf(bufa, bufb); /* from bufb to bufa */
  } else {
    MPI_Neighbor_alltoallw(bufa, ..., bufa, ...  , comm);
  }
}
```

図 3 集団呼出しごとに通信パラメタが変化する例

この問題を解決する方法の一つが，次節 3.4 で述べる永続集団通信である．プログラマが，ブロッキング集団操作 MPI_Neighbor_alltoallw または非ブロッキング集団操作 MPI_Ineighbor_alltoallw の代わりに，永続集団操作 MPI_Neighbor_alltoallw_init に書き換えることにより，すべてのプロセスが固定された通信パラメタで呼出されることが保証される．

3.4 永続集団通信

永続集団通信は，永続型の非ブロッキング集団通信であり，一対一通信での永続通信 (MPI_Send_init 等) の集団通信版である．永続集団通信は MPI 標準の次版 MPI-4.0 での機能追加の候補として MPI Forum [4] で現在検討中である．現在の提案では，すべての非ブロッキング集団通信 MPI_I*coll* に対し MPI_*coll*_init を追加し，その引数は元の MPI_I*coll* と同じとしている．実際の通信は MPI_Start で開始し，MPI_Wait で完了する．また，集団通信にはタグの概念はないので，同じコミュニケータ上ではすべてのプロセスが同じ順序で集団操作を開始 (MPI_Start) する必要がある．つまり，実際の通信時は，すべてのプロセスが固定された通信パラメタで呼出されることが保証される．なお，永続集団操作は，ブロッキング集団操作や通常の非ブロッキング集団操作とはマッチングしないため，全プロセスが永続集団操作を呼び出すことが保証される．これにより，一対一通信での永続通信のように，MPI_Send_init の要求に対し通信相手が MPI_Irecv で応答するような，異なるアルゴリズムの混在の可能性は排除される．

本稿では，煩雑な RDMA メモリアドレスの交換プロトコルを排したより簡素化されたプロトコルを検討し，富士通 PRIMEHPC FX100 で導入されたプロトコル・オフローディング機能 Tofu2 Session モード CQ への適用を検討する．

4. 設計と実装

4.1 Tofu2 Session モード CQ

富士通 PRIMEHPC FX100 でサポートされる Tofu インターコネクト 2 (以降，Tofu2 と略す) [5] の場合，各計算ノードは CPU に統合された Tofu2 コントローラをもつ．Tofu2 コントローラ は (1) Tofu Network Router (TNR), (2) Tofu Network Interface (TNI), 及び (3) Tofu Barrier Interface (TBI) から構成される．ルータ部の TNR には 10 本の物理リンクが接続され，それぞれ 12.5 GB/s (× 双方向) のリンク速度をもつ．RDMA エンジン部の TNI は 4 基あり，全体で同時に 4 送信+4 受信可能である．また reduce 演算器つきの通信バリアインターフェイス TBI は `MPI_Barrier` や `MPI_Allreduce` 等の集団通信を高速化するためのハードウェア支援機構である．

Tofu2 では RDMA エンジン部 TNI あたり 12 個の制御キュー (以降 CQ と略す) をもつ．CQ は，RDMA 操作の実行を管理する 3 つの組の FIFO キューと見なすことができる．つまり，コマンド・キュー，コマンドの完了キュー，およびイベント・キューの 3 つである (それぞれ，TOQ, TCQ, および MRQ と略す)．Put 等の RDMA 操作をエンコードした RDMA ディスクリプタを作成し，TOQ に書き込む．RDMA ディスクリプタは，操作のタイプや宛先 CQ のネットワーク・アドレスや RDMA メモリ・アドレスを含む．TNI の送信 DMA は TOQ から RDMA ディスクリプタを順番に実行し，実行完了後ステータスを TCQ に書き込む．MRQ は，TNI の受信 DMA によってその CQ に対する，リモートでの実行ステータスや結果，受信完了通知が書き込まれる．

Tofu2 では，CQ は通常モードに加え，Session モードが追加されており，初期化時にどちらのモードで動作するかを選択できる．Session モード選択時には，次のように動作する．

(1) TOQ FIFO キューでは，通常の Read ポインタおよび Write ポインタに加え，*Scheduling* ポインタが有効にになる．
(2) Read ポインタは *Scheduling* ポインタを追い越すことはなく，TOQ キューのエントリは *Scheduling* ポインタがそのスロットより先を指すまで，実行が遅延される．
(3) TNI 受信 DMA は RDMA Put パケットを受信したとき，そのパケット・ヘッダの SPS フィールドの値に従って，TOQ の *Scheduling* ポインタを進める．

4.2 PRDMA プロトコル

PRDMA [6] [7] は受信側主導の永続通信 (一対一通信) 向けの RDMA プロトコルである．PRDMA プロトコルの概要を図 4 に示す．

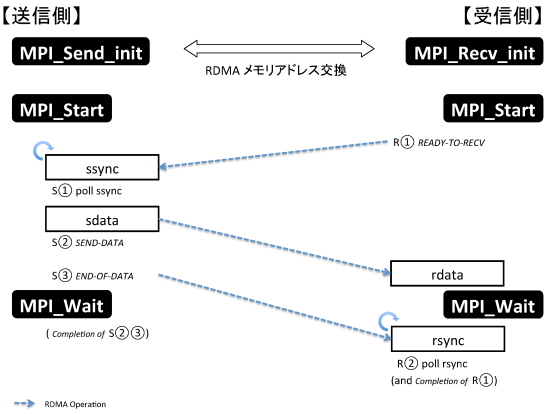

図 4 PRDMA プロトコルの概要

PRDMA ではまず，`MPI_Send_init` と `MPI_Recv_init` との間で，プロトコルの実行に必要な RDMA メモリアドレスを交換する．受信側では，送信側に受信準備完了を通知する ssync の RDMA メモリアドレスを入手する．送信側では，リモートの送信先 rdata のメモリアドレス、および受信側に送信完了を通知する rsync のアドレスを入手する．受信側で `MPI_Start` が呼出されると，受信準備完了を伝えるため，`READ-TO-RECV` を送信する (R①)．送信側で `MPI_Start` が呼出されると，受信準備完了を確認するため，ssync にメモリ・ポーリングを行う (S①)．ssync に特定の値がセットされたら，ポーリングを抜け，sdata から rdata にユーザ・データを転送ために RDMA Put ディスクリプタをコマンドキューに書き込む (S②)．さらに転送の完了を受信側に通知するために，END-OF-DATA をコマンドキューに書き込む (S③)．受信側で `MPI_Wait` が呼出されると，`READY-TO-RECV` の送信完了，及び転送の完了を確認するため，rsync をメモリ・ポーリングを行う (R②)．送信側で `MPI_Wait` が呼出されると，`SEND-DATA` および `END-OF-DATA` の送信完了をチェックする．

4.3 Session モード CQ 上の PRDMA

ここでは，前節 4.2 で述べた PRDMA プロトコルを Tofu2 Session モード CQ へ適用する場合の基本的な方式について述べる．

図 5 の例は，送信側が `MPI_Start` を呼び出す前に受信側から `READY-TO-RECV` が到着済である，プロセス間の同期が理想的なケースであった．図 6 は，受信側から `READY-TO-RECV` が到着するまで，`SEND-DATA` の実行が遅延する例である．

しかしながら，Session モード CQ によって，プロトコルを Tofu2 ハードウェアにオフロードしているため，CPU の介在なく，真に非同期通信を可能とする．

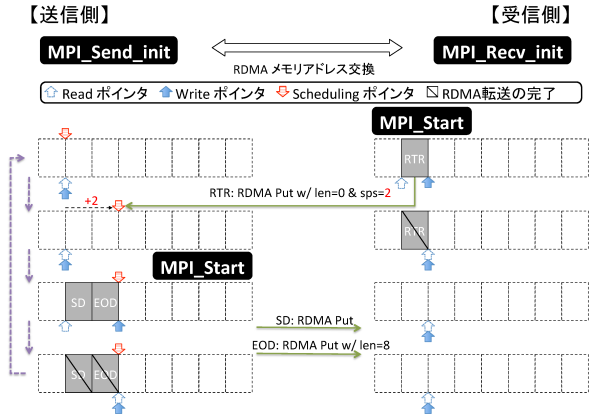

図 5 PRDMA プロトコルの Session モード CQ への適用
図中のボックスは，CQ のコマンドキュー (FIFO) のスロットを模式化したもの．また，図中の RTR, SD, および EOD はそれぞれ，PRDMA プロトコルの READY-TO-RECV, SEND-DATA, および END-OF-DATA の略号であることに注意．

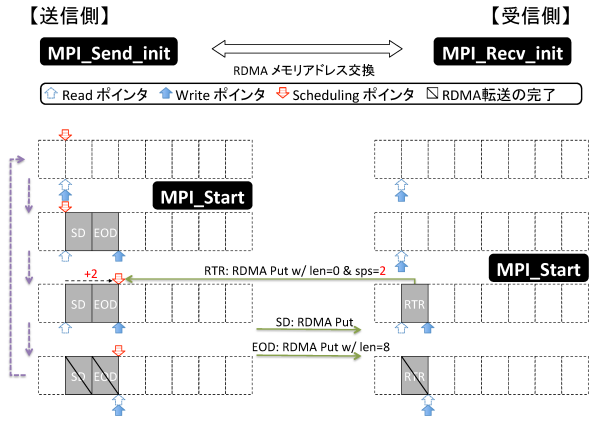

図 6 データ転送が遅延するケース

4.4 隣接集団通信用 PRDMA

前節 4.3 では，一対一通信向けのオリジナルの PRDMA プロトコルの Session モード CQ への写像について検討した．この節では，PRDMA-SMCQ と呼ぶ，永続型の隣接集団通信向けの写像方式について検討する．図 7 は，隣接する一方の奇数・偶数ランクどうしでデータを交換する通信パターンおいて，MPI_Neighbor_alltoallw_init に対し PRDMA-SMCQ 方式を使って写像した例である．

前節 4.3 の例と異なり，集団通信をオフロードするために，すべてのプロセスが Session モード CQ を使用している．この例ではまた，READY-TO-RECV を送信するのに通常のモードの CQ を使用せず，Session モード CQ のみで実現するために，初期の Scheduling ポインタの位置を Read ポインタより 1 つ分進めていること，および READY-TO-RECV 制御メッセージに付随する sps フィールドの値が 1 つ多いことを除き，以前の例と大きな違いはない．これにより，

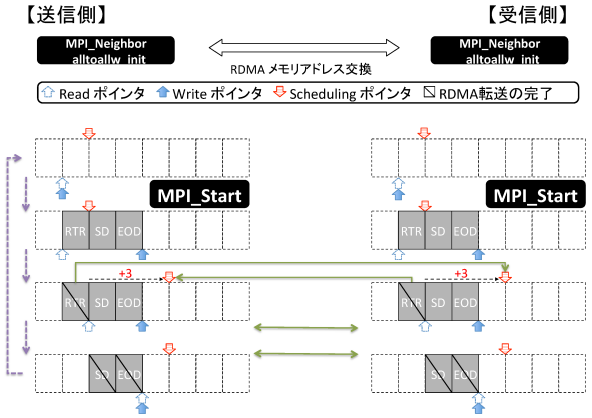

図 7 データ交換への適用例

Session モード CQ から READY-TO-RECV が直ちに送信される．

図 7 では，非常に単純化した交換通信パターンを示したが，より複雑かつ現実の通信パターンである二次元 5-point ステンシル計算における袖領域の交換でも，4 つの通信相手に対して 1 つの Session モード CQ を割当て，図 7 に示した交換を実行すれば，容易に拡張できる．この場合，Tofu2 ではノードあたり 4 基の RDMA エンジン部 TNI を有するので，各 TNI から 1 つの Session モード CQ を選べば，独立した DMA を 4 基使って，並列に転送可能である．

4.4.1 PRDMA-SMCQ の課題

Tofu2 ではノードあたり 48(= 4 × 12) 個の CQ があるが，FX100 の CPU は NUMA 構成であり，2 CMG (Core-Memory Group) からなるため，ノードあたりの推奨 MPI プロセスは 2 である．従って MPI プロセスあたりの利用可能な CQ は半分になる．さらに，MPI の基本動作に加え，OS や ARMCI 等によって専有される CQ もあるので，集団通信に利用可能な Session モード CQ は上限がある．

また，プロトコル・オフローディング機能を提供する Session モード CQ は隣接集団通信以外の永続集団通信でも活用されるされることが望ましい．しかしその多くは，隣接集団通信のような疎な集団通信ではないため，例えば，recursive doubling のようなアルゴリズムのように $log_2(N)$ 個 (N はコミューケータに属するプロセス数) の通信相手とステップごとに通信するために，ジョブの規模によって，大量に CQ が消費される恐れがある．このために，高速型と低資源消費型の複数の方式を実装することが望ましい．

4.5 低資源消費型 PRDMA-SMCQ

この節では，節 4.4.1 の課題を解決するために，低資源消費型の PRDMA-SMCQ (以降，PRDMA-SMCQ-MRC と略す) 方式について検討する．

図 8(B) 及び (C) では，一次元 3-point ステンシル計算

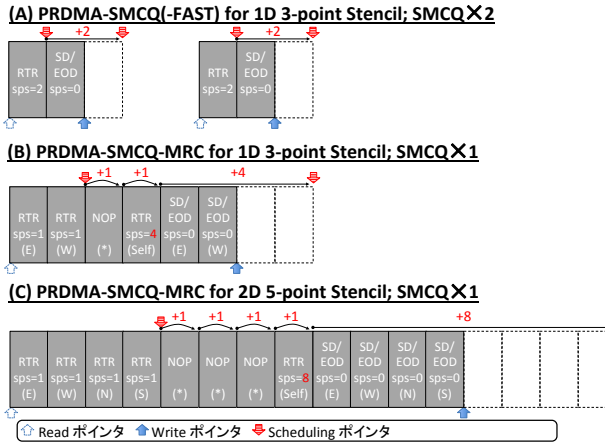

図 8 低資源消費型 PRDMA-SMCQ の例

および二次元 5-point ステンシル計算における袖領域の交換に対する，低資源消費型の PRDMA-SMCQ-MRC の適用例について説明している．なお，データ交換パターンであるため，通信は同型であり，一つの MPI プロセスのコマンドキューのみ記述している．また Tofu では，RDMA Put において，送信先の CQ のイベントキューに受信通知イベントの生成を指示することができるので，送信側は `SEND-DATA` でその指示つきの RDMA Put をコマンドキューに書き込むことにより，`SEND-DATA` と `END-OF-DATA` を一つの RDMA Put で実現することができる．受信側は rsync をメモリポーリングする代わりに，イベントキューで受信通知イベントを待つことができる．これを `SEND-DATA/END-OF-DATA` と呼ぶ (図中では SD/EOD)．

この PRDMA-SMCQ-MRC (Minimal Resource Consumption) の基本的なプロトコルの動作は，最初に必要な相手に `READY-TO-RECV` を送信，次に `READY-TO-RECV` が全員から届くのを待って，最後に `SEND-DATA/END-OF-DATA` でデータ転送を開始する．全員から `READY-TO-RECV` が届くのを待つ際，最後の `READY-TO-RECV` の受信時に，自分自身の CQ に対し，適切な sps を持った `READY-TO-RECV` を送信するように，当該 Session モード CQ を制御している．なお，NOP とは Tofu での何もしない RDMA NOP である．

以降，PRDMA-SMCQ-MRC と区別が必要な場合，前節 4.4 の PRDMA-SMCQ を，文脈に応じて PRDMA-SMCQ-FAST と呼ぶことにする．

5. 評価

5.1 測定環境

理研計算科学研究機構に設置されている 36 ノード構成の PRIMEHPC FX100 システムを使用した．Session モード CQ 使用にあたっては，Tofu2 ハードウェア直上で動作し，Tofu2 に対する低レベルのアクセスを提供する Tofu ライブラリを使用した．なお，このライブラリの API は公開されていない．

5.2 測定結果

本評価では，図 8 の (A) 及び (B) を実装し，Tofu2 のハードウェアの基本特性を知る目的なので，一次元 3-point ステンシルの袖領域交換ベンチマークで測定した．このベンチマークでは，2 つの通信相手とデータ交換を行い，メッセージを 0 ～ 65535 バイトの範囲で 1024 バイトずつ増やして `MPI_Start` と `MPI_Wait` コストを測定した (計算は 0 で、通信のみ)．測定では，MPI プロセストポロジと物理ネットワークトポロジを適切に写像しており，通信相手との間で隣接ノード間通信となっている．

図 9 は，PRDMA-SMCQ-FAST の測定結果である．

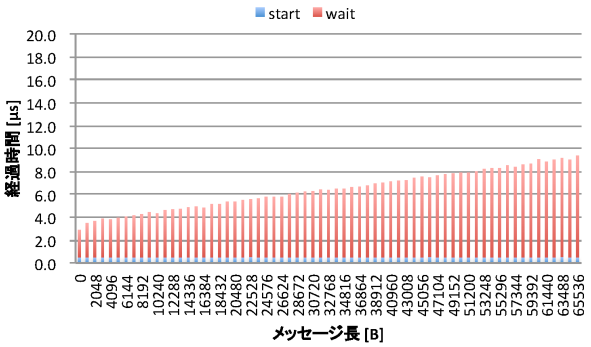

図 9 PRDMA-SMCQ (高速型) のベンチマーク結果

この方式では，通信相手の MPI プロセスに対し，1 つの CQ が割当てられており，各 CQ は独立した RDMA エンジンに属するよう選択されているので，並列転送の効果が期待できる．測定結果からも，メッセージ長に対し線形であり，その傾きから概ね 10 GB/s 出ており，理論リンク帯域 12.5 GB/s に対し，データ交換の同時送受信，並びに 2 方向との同時データ交換がほぼ隠蔽できており，妥当な結果と言える．

これに対し，PRDMA-SMCQ-MRC (最小資源消費) 方式では 1 つの CQ で 2 方向のデータ交換を行うため，使用される RDMA エンジンの実体は 1 基であり転送はシリアライズされ，PRDMA-SMCQ-FAST に対し約 1/2 の性能になることが予想される．図 10 に示す PRDMA-SMCQ-MRC の測定結果からも，コストが PRDMA-SMCQ-FAST に対し，約 2 倍になっている．しかしながら，図 10 の MRC/FAST は PRDMA-SMCQ-FAST に対する PRDMA-SMCQ-MRC の比であり，MRC/FAST に対する右縦軸の値を詳細に見ると，約 1.2 ～ 1.8 倍になっており，特にメッセージ長 16 KiB 付近では劣化度合いは 1.5 倍程度になっている．メッセージ長によっては，最小資源消費型でも通信と計算のオーバーラップがうまくゆく可能性がある．

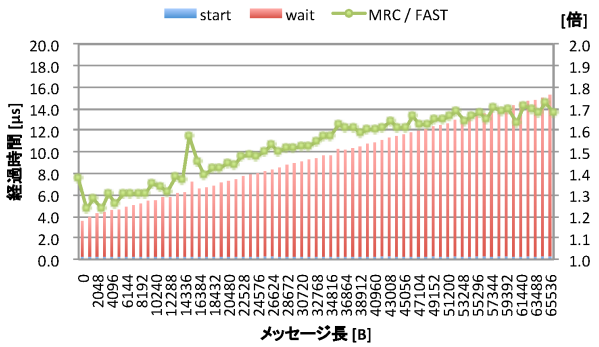

図 10 PRDMA-SMCQ (最小資源型) のベンチマーク結果

6. おわりに

本稿では，MPI の永続型隣接集団操作である `MPI_Neighbor_alltoallw_init` を実現するために，PRDMA と呼ばれる一対一通信の永続通信向けの RDMA プロトコルを PRIMEHPC FX100 でサポートされる Tofu2 インターコネクト・ハードウェアにオフロードするための写像方法，並びにその実装に対する予備的な性能評価を論じた．なお，MPI 永続集団通信については，現在標準化団体 MPI Forum にて策定作業中であり，執筆時点で最新の成果を元に設計を試みたが，Tofu2 のプロトコル・オフローディング機能に関して，永続集団通信の仕様上の問題は発生しなかった．本実装では，プロトコル・オフローディング機能の核となる Session モード CQ の枯渇を想定し，高速型 (CQ 高消費) と最小資源型 (CQ 低消費) の 2 つを，FX100 Tofu2 ハードウェア直上の Tofu2 ライブラリを使って実装した．性能評価をとおして，本方式に対し，高速型実装の性能妥当性を示した．また，最小資源型実装は，高速型実装に対し，一次元 3-point ステンシルにおける 16 KiB のデータ交換パターンで約 1.5 倍のレイテンシ劣化に抑えられることを示した．

高速型と最小資源型の間には，中間的な CQ 資源消費実装も考えられる．また，NUMA 構成に対応し，ノード内複数プロセスの通信全体を代表プロセス及び CQ に集約するような CQ 消費抑制案も考えられる．今後，これらの実現可能性について検討してゆく予定である．

参考文献

[1] Si, M. and et al: Casper: An Asynchronous Progress Model for MPI RMA on Many-Core Architectures, *IPDPS '15*, pp. 665–676 (2015).
[2] Schneider, T., Hoefler, T., Grant, R. E., Barrett, B. W. and Brightwell, R.: Protocols for Fully Offloaded Collective Operations on Accelerated Network Adapters, *ICPP '13*, pp. 593–602 (2013).
[3] Message Passing Interface Forum: MPI: A Message-Passing Interface Standard, Version 3.0, Technical report (2012).
[4] Message Passing Interface Forum: MPI STANDARD EFFORTS / MPI 4.0, `http://www.mpi-forum.org/mpi-40/`.
[5] Ajima, Y. et al.: Tofu Interconnect 2: System-on-Chip Integration of High-Performance Interconnect, *ISC '14*, pp. 498–507 (2014).
[6] Ishikawa, Y., Nakajima, K. and Hori, A.: Revisiting Persistent Communication in MPI, *EuroMPI '12*, pp. 296–297 (2012).
[7] Hatanaka, M., Hori, A. and Ishikawa, Y.: Optimization of MPI Persistent Communication, *EuroMPI '13*, pp. 79–84 (2013).