Vol.2018-HPC-163 No.12
2018/3/1

# FX100における永続型集団通信関数のプロトタイプ実装と評価

森江 善之[1,a)] 畑中 正行[1] 高木 将通[1] 堀 敦史[1] 石川 裕[1] 南里 豪志[2]

**概要**：本稿では，永続型集団通信についてオフロード機能を用いた実装およびプログレススレッドを用いた実装を行い，それぞれの通信性能や各種オーバーヘッドについて評価を行った．
オフロード機能は，通信コマンド列の事前準備やCPUを介在しない通信発行が行えるため，永続型集団通信の実装に適している．一方でアシスタントコアを利用することでプログレススレッドによる実装においても高速な永続型集団通信の実装が可能であると考えられる．このため，これらのプロトタイプ実装を行い，その性能の傾向を調査した．
今回は，オフロード機能をもつTofu2インターコネクトやアシスタントコアの両方を備えるFujitsu PRIMEHPC FX100上にて永続型ブロードキャストの実装を行い，その評価を行った．この時，オフロード機能による実装は，プログレススレッドによる実装に対して64MiB以下のすべてのメッセージサイズで通信性能が上回ることを確認した．一方，プログレススレッドによる実装では，メッセージサイズが128MiB以上ではオフロード機能による実装より高速になることを確認した．また，いずれの実装においても集団通信開始関数のコストは，十分に小さいことを確認した．

**キーワード**：永続型集団通信，オフロード機能，プログレススレッド，ブロードキャスト

# Evaluation of Prototyping Persistent Collective Communication on FX100

Yoshiyuki Morie[1,a)] Masayuki Hatanaka[1] Masamichi Tagaki[1] Atsushi Hori[1]
Yutaka Ishikawa[1] Takeshi Nanri[2]

## 1. はじめに

近年，スーパーコンピュータは大規模化が進み，数万から数十万規模の計算ノードが結合する並列計算機となっている．このような大規模並列計算機においては，多数のプロセス間での通信を実行する集団通信の開発は，その通信性能や資源管理を行う上でより重要となっている．

現在，このような問題に対して新しい集団通信インターフェースである永続型集団通信がMPI Forum[1]にて議論がされている．永続型集団通信とは集団通信で実行される初期化の処理を集団通信の準備関数として独立させるもので，この準備関数においてアルゴリズムや利用する資源の設定などを事前に行うことができる．

したがって，永続型集団通信は対象のプログラムにおいて集団通信が繰り返し実行されるものであれば，準備関数を事前に一度呼び出すことで各集団通信のインスタンスにおいて事前準備を省略することができる．さらに，永続型集団通信は非同期集団通信の一種であり，計算と通信のオーバラップの実行が可能となる．

このような性質を持つ永続型集団通信は，オフロード機能を利用することで通信性能が向上することが期待される．また，非同期集団通信の一種であることからプログレ

[1] 理化学研究所 計算科学研究機構
Riken Advanced Institute for Computational Science (AICS), kobe, hyogo, 650-0047, Japan
[2] 九州大学
Kyushu Uniersity
[a)] yoshiyuki.morie@riken.jp

スレッドを用いた実装も検討が必要となる．

そこで，本稿ではオフロード機能を持つ通信デバイスとプログレススレッドの用途で用いることが可能なアシスタントコアの両方を備える Fujitsu PRIMEHPC FX100 (FX100) において永続型集団通信のプロトタイプ実装を行う．また，その通信性能や各種オーバヘッドに関して評価を行う．

本稿は以下のような構成となる．まず，第 2 節にて永続型集団通信の説明を行う．次に，第 3 節で永続型集団通信の各種実装の説明を行い，第 4 節でこれらの性能を評価するための実験を行う．また，第 5 節では，関連研究について述べ，最後にまとめと今後の課題を述べる．

## 2. 永続型集団通信

永続型集団通信は，MPI_( NAME )_init,MPI_Start(), MPI_Wait(), MPI_Request_free() の 4 個の関数で構成される．

MPI_( NAME )_init は，永続型集団通信の準備関数で事前の各種パラメタの設定や資源の確保などを行う関数である．また，MPI_Start() は，永続型集団通信の開始関数，MPI_Wait() は，完了確認関数である．これらにより，計算と通信のオーバラップ実行などの非同期な動作が可能となる．また，MPI_Request_free() は，準備関数などで確保された資源を解放する関数である．これにより，他の永続型集団通信の発行などによる通信資源の枯渇を防ぐことが可能となる．

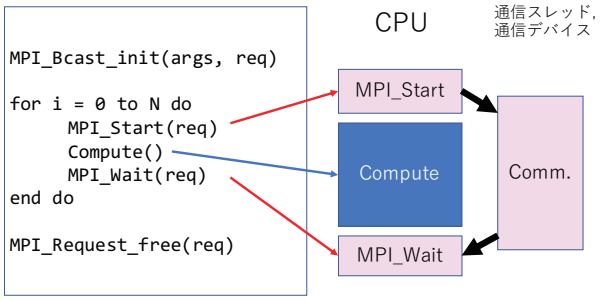


**図 1** 永続型集団通信の動作概要

永続型集団通信の実行例として図 1 に疑似コードを示す．この疑似コードのように集団通信本体のみを独立して繰り返し実行するように記述することで，準備関数において以降で用いる事前の処理をあらかじめ実行できる．また，集団通信の開始関数にて通信処理本体をプログレススレッドないし通信デバイスにオフロードすることで計算と通信のオーバラップが可能となる．

## 3. FX100 での永続型ブロードキャストの実装

本稿では，京コンピュータおよび FX100 でブロードキャストの通信アルゴリズムとして利用された Trinaryx3 アルゴリズム [2] を永続型集団通信関数の実装対象とした．このアルゴリズムは，ネットワークトポロジを考慮して通信競合を起こさないパイプラインパスを複数生成し，複数搭載されている通信デバイスの通信帯域幅を有効に利用する通信アルゴリズムとなっている．また，これは，複数の通信デバイスを持つ Blue Gene/L でも同様の通信アルゴリズム [3] が提案されており，適用範囲の広い通信アルゴリズムといえる．

### 3.1 オフロード機能を用いた永続型ブロードキャストの実装

Trinaryx3 アルゴリズムはブロードキャストで転送するデータを 3 個のフラグメントに分割，それぞれのデータを Trinary アルゴリズムを用いてパイプライン転送するものである．3 本の Trinary tree のパイプラインパスは，図 2 に示すように 3D トーラスにマッピングされる．各パイプラインパスのデータ転送は，Tofu2 の Tofu Network Intarface (TNI) によってオフロード実行する．

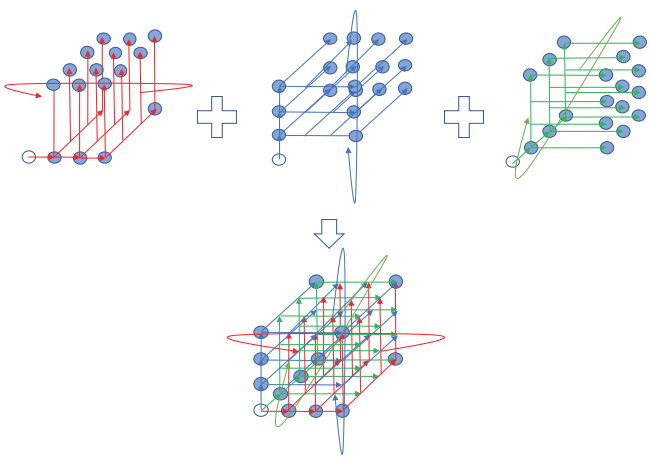

**図 2** Overview of Trinaryx3.

#### 3.1.1 Session Mode CQ

Tofu2 は Session Mode CQ (SMCQ) と呼ばれる通信を CPU の介在なしに通信デバイスだけで発行する機能を持つ．SMCQ は，図 3 に示すように FIFO の動作を基本とする Transmit Order Queue (TOQ) と 3 個のポインタからなる．ポインタはそれぞれリードポインタ，ライトポイン

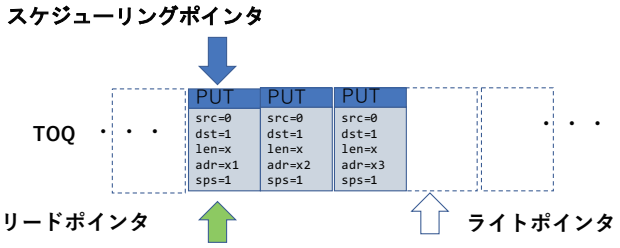


**図 3** Session mode CQ

タとスケジューリングポインタである．

TOQ には，TOQ ディスクリプタが投入される．この TOQ ディスクリプタを元に TNI が通信の発行を行う．この TOQ ディスクリプタは RDMA 通信に関する各種情報，例えば，RDMA の通信タイプ，宛先アドレス，メモリ領域のアドレス，データサイズ，Session Progress Step (SPS) などを保持する．SPS は，スケジューリングポインタの進行幅を指定する情報領域である．

TNI によりライトポインタは TOQ ディスクリプタが投入されるごとにキューの末尾を指すよう移動する．また，リードポインタも同様に先頭の TOQ ディクリプタを指すように移動する．SMCQ でないモードでは，TNI はリードポインタに従い，先頭から TOQ ディクリプタを元に通信コマンドを発行しリードポインタが末尾に至るまでコマンドを発行し続ける．

一方，SMCQ においては，リードポインタはスケジューリングポインタを追い越すことができない (図 4)．TNI は，スケジューリングポインタが指した要素より以前の要素であれば，リードポインタが指している通信コマンドを実行できる．そして，このスケジューリングポインタは，RDMA パケットを受け取った時にのみ SPS フィールドで指定された値と同一量前進させることができる．

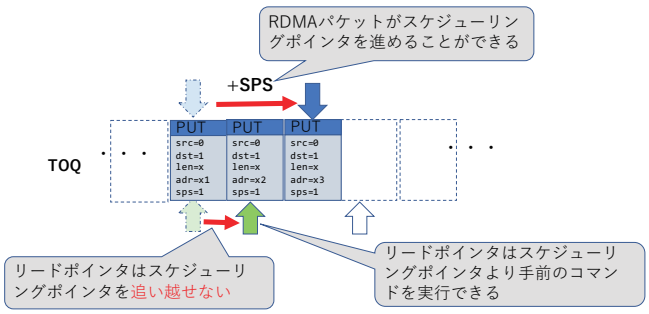


**図 4** SMCQ の動作

これらの機能により RDMA パケットが到着しない間は通信の発行を待つことができ，RDMA パケットが到着した時に通信コマンドをどこまで発行するかを決めることができる．このため，SMCQ を利用すれば CPU の介在なしにパケットの到着の起点として通信を進行させるオフロード機能を実現できる．

### 3.1.2 オフロード機能による永続型ブロードキャストの実装

Trinaryx3 アルゴリズムは 3 本の 3 分木のパイプラインパスを持つブロードキャストアルゴリズムである．単一パイプラインパスに対して枝分かれ処理と複数通信デバイス化を行ったものが，今回実装した永続型ブロードキャストとなる．ここでは，簡単のために単一パイプラインパスのブロードキャストをオフロード機能により実装した場合について述べる．

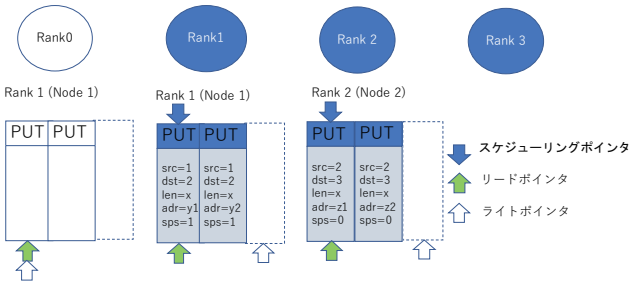


**図 5** 単一パイプラインパスのブロードキャスト:初期状態

以下，オフロード機能を用いた単一パイプラインパスのブロードキャストアルゴリズムの動作の説明を行う．ここでは，2 セグメントを 4 プロセスでパイプライン転送するブロードキャストを考える．

まず，SMCQ の動作が始まる直前の初期状態について述べる．はじめにルートランク以外の各ランクにおいて通信コマンド列を投入したあと，バリア同期を行い，全ランクが通信可能であることを確認する．TOQ は，図 5 に示すような状態となる．

簡単のためにバリア同期で，通信可能であることを確認する例を用いたが，我々の実装では，ルートランクも同様に SMCQ を用いており，他のランクが準備完了のフラグとしての RDMA パケットをルートランクへ送信することで処理を開始するよう設計している．

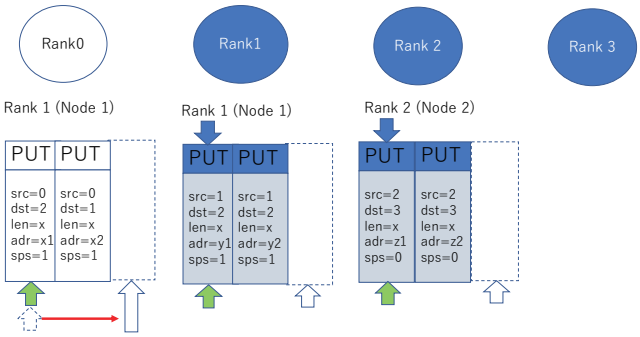


**図 6** 単一パイプラインパスのブロードキャスト:第 1 ステップ

次に，すべてのプロセスの準備が完了したことを確認したら，図 6 で示すようにルートランクは TCQ に対してコマンドを 2 個投入する．ルートランクで TOQ へのコマンドを投入されたことで集団通信の処理が開始される．

次に，図 7 に第 2 ステップを示す．ルートランクは，SMCQ を利用してないので直ちにリードポインタの示す通信コマンドの実行を行う．これによりルートランクはランク 1 へ第 1 セグメントの送信を行い，リードポインタを 1 前進させる．ランク 1 がこのセグメントを受け取るとルートランクで TOQ ディスクリプタの設定時に SPS を 1 と設定していることからスケジューリングポインタが 1 前進する．

同様に図 8 に第 3 ステップを示す．ルートランクは，ラ

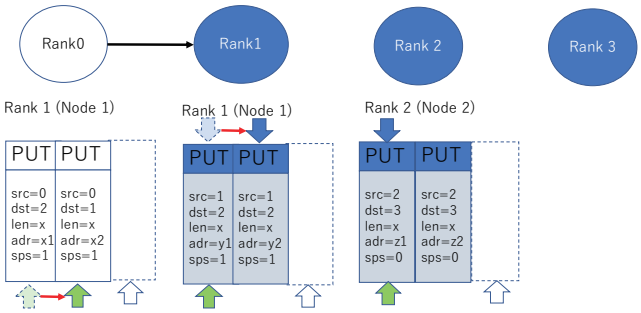


**図 7** 単一パイプラインパスのブロードキャスト:第 2 ステップ

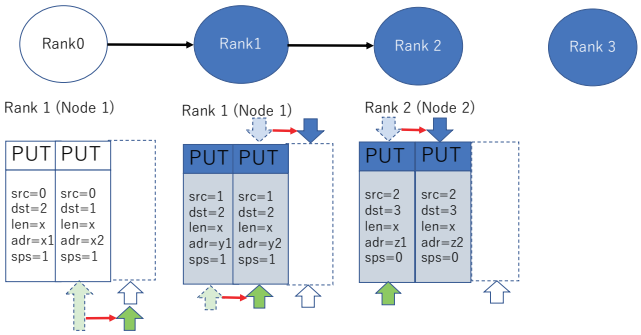


**図 8** 単一パイプラインパスのブロードキャスト:第 3 ステップ

ンク 1 へ第 2 セグメントの送信を行い，リードポインタを 1 前進させる．ランク 1 がこのセグメントを受信すると，スケジューリングポインタが第 1 ステップと同様に 1 増加する．ルートランクは処理を終える．この時，ランク 1 は最初のコマンドを実行できるため，ランク 2 へ第 1 セグメントの送信を実行する．ランク 1 のリードポインタはランク 2 の通信が終わると前進する．ランク 2 ではこのセグメントを受信すると，スケジューリングポインタを前進させる．これにより，ランク 2 は第 1 セグメントをランク 3 へ送付可能となる．

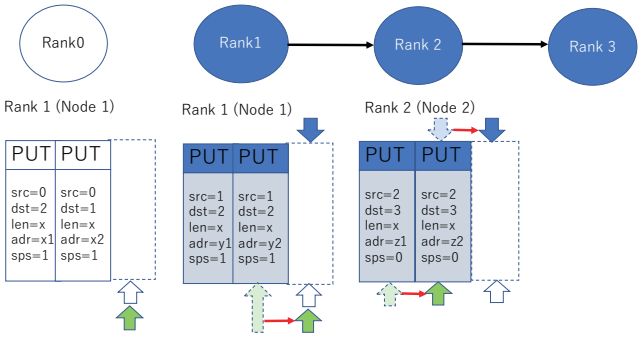


**図 9** 単一パイプラインパスのブロードキャスト:第 4 ステップ

同様に第 4 ステップ (図 9) ではランク 1 がランク 2 へ第 2 セグメントを送信したあとリードポインタを前進させて動作を完了する．ランク 2 ではこのセグメントを受信した時，スケジューリングポインタを 1 前進させる．これにより，ランク 2 は第 2 セグメントをランク 3 へ送信可能となる．

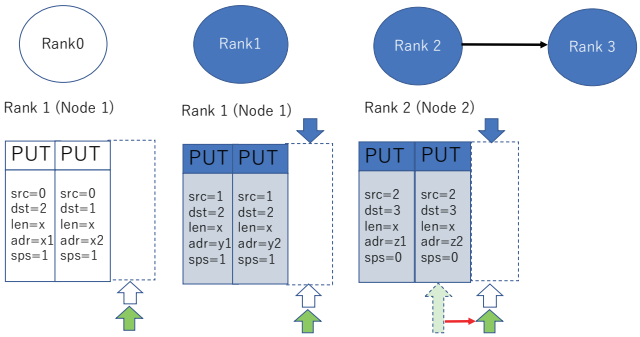


**図 10** 単一パイプラインパスのブロードキャスト:終了状態

最後に，図 10 のようにランク 3 へ残りのセグメントを送信を実行したら，リードポインタを 1 進める．ランク 3 へ通信の完了確認ができたら，集団通信を完了させる．

### 3.2 プログレススレッドによる永続型ブロードキャストの実装

ここでは，プログレススレッドを用いた永続型ブロードキャストについて述べる．

プログレススレッドが実行する集団通信は CPU が利用できることと実装による性能差を確認するなどの理由からプログレススレッドによる実装では SMCQ を利用しない形で実装する．このため，準備関数の実行には通信コマンド列の生成は行わずにプログレススレッドにて通信発行可能になったタイミングで通信コマンドを生成してその都度通信コマンドを投入する方式の実装とした．

永続型集団通信を非同期動作とするため，プログレススレッドを生成する．メインスレッドで実行される各関数はプログレススレッドに対して通信処理を依頼するフラグ設定のみを行う．

生成されたプログレススレッドでは，以下の処理を行う．通信開始フラグを監視し，通信開始フラグが設定されれば，集団通信を実行する．集団通信が終了したら，通信完了フラグを設定する．また，プログレススレッドの解放フラグを監視し，解放フラグが設定されていたら，プログレススレッドを終了する．

集団通信の準備関数において pthread_create() によりプログレススレッドを生成する．集団通信の開始関数では，通信開始フラグを設定し，集団通信の完了確認関数では，通信完了フラグを監視する．資源解放関数では，プログレススレッドの解放フラグを設定し，pthread_join() によりプログレススレッドの終了を待ち合わせる．

## 4. 性能評価実験

今回，性能評価実験では永続型集団通信に関して前節で紹介したプログレススレッドによる実装とオフロード機能を用いた実装を用意し，それぞれの通信性能および集団通信準備関数，集団通信開始関数のコストについて評価を

行った．今回は，ユーザ権限でのアシスタントコアの利用法が判明しなかったため，代替として演算コアでプログレススレッドを動作させる形で実験を行った．

### 4.1 実験環境

実験環境とした Fujitsu PRIMEHPC FX100 の仕様を以下に述べる．CPU は Fujitsu SPARC64$^{TM}$ XIfx (32 演算コア+2 アシスタントコア)，メモリは 32GB，メモリバンド幅は，480GB/s，計算ノード数は 36 基である．また，OS やコンパイラ，MPI ライブラリは富士通株式会社独自開発のものである．また，ネットワークは，Tofu2[4] を用いる．この通信帯域幅は，12.5GB/s で，ネットワークトポロジは 6 次元メッシュ/トーラスでユーザは仮想 3 次元トーラスとしてアクセスする．また，各計算ノードでは通信デバイスを 4 基搭載する．

### 4.2 実験結果

図 11, 図 12 に永続型ブロードキャストの実行時間，図 13 に実効通信帯域幅を示す．

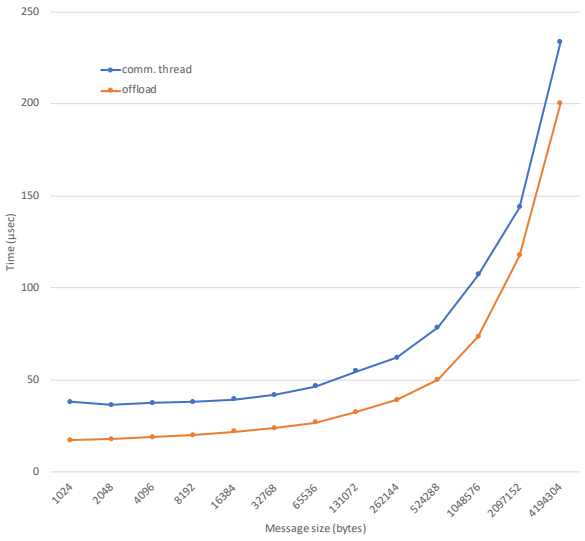


**図 11** 実行時間（≤ 4MiB）

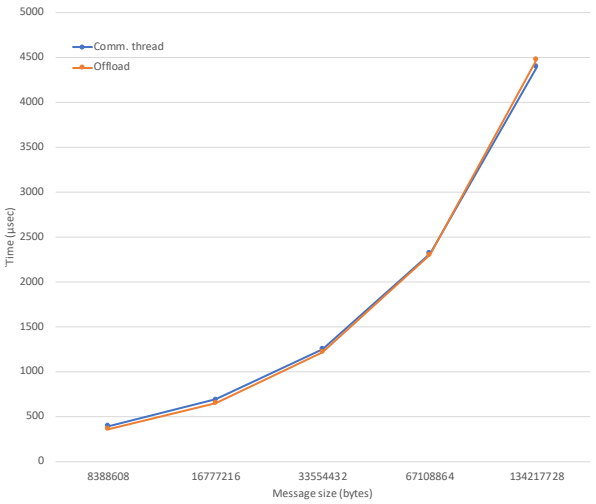


**図 12** 実効時間（≥ 8MiB）

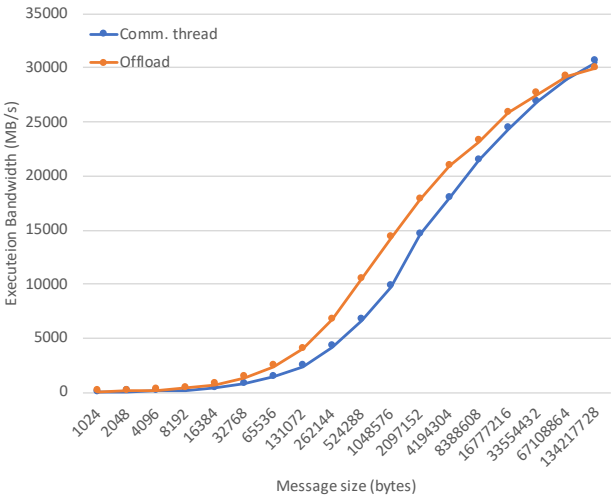


**図 13** 実効通信帯域幅

オフロード機能を用いた実装では，64MiB まですべてのメッセージサイズでのプログレススレッドによる実装に比べ性能が高速であった．とりわけ，1KB の時はもっとも高速でオフロード機能を用いた実装は 17.25$\mu$s でプログレススレッドによる実装は 37.92$\mu$s であった．メッセージサイズが小さい時の方がオフロード機能よる実装はより高速であった．これは，開始関数の実行時に事前に生成した通信コマンド列を一度に投入することで通信コマンド列の生成や個別投入によるコストが削減されたと考えられる．

また，大メッセージサイズでは，徐々に性能差がなくなり，128MiB では，プログレススレッドによる実装の方が 560MiB/sec ほど高速であった．これは，CPU が行っていた枝分かれする通信の起動をオフロード機能を用い場合は RDMA パケットとして別途発行する必要があるためだと考えられる．

次に，図 14 に集団通信開始関数の実行時間を示す．どちらの実装でも実行時間は非常に小さく集団通信開始関数の実行時間は，プログレススレッドによる実装が 1.1-1.4$\mu$s, オフロード機能による実装が 2.1-7.9$\mu$s となった．これはプログレススレッドによる実装では通信開始フラグを設定するのみで，その際に実行される pthread_mutex_lock()/pthread_mutex_unlock() が主要な処理となっており非常に短い実行時間となった．また，オフロード機能による実装では TOQ への通信コマンド列の投入によるメモリコピーのみとなるため，こちらも短い実行時間となった．オフロード機能による実装がメッセージサイズに比例して実行時間が増加しているのは，メッセージサイズに比例してセグメント分割数が増加して通信コマンド列が増加することが原因と考えられる．

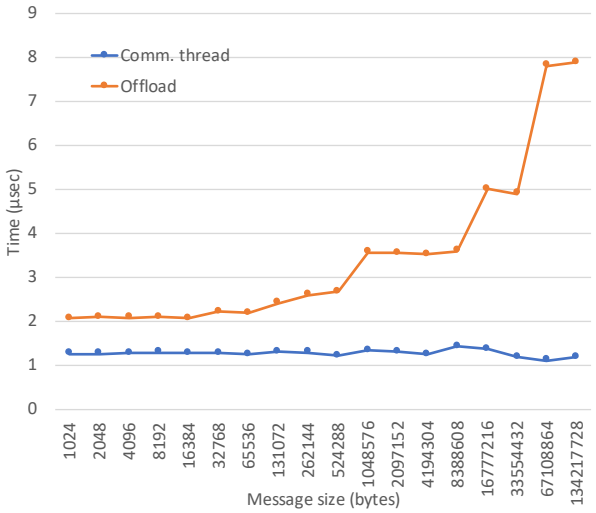


**図 14** 永続型集団通信の開始関数 (MPI_Start) の実行時間

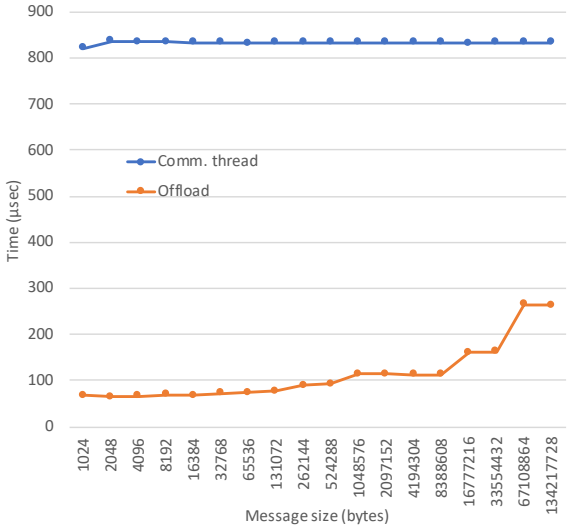


**図 15** 永続型集団通信の準備関数 (MPI_Bcast_init) の実行時間

最後に，図 15 に準備関数の実行時間を示す．その実行時間は，プログレススレッドによる実装が 821-835.9$\mu$s, オフロード機能による実装が 67.9-263.9$\mu$s となった．

オフロード機能による実装では，TOQ コマンド列を生成するコストが大部分となる．メッセージサイズが増加するにつれて実行時間が増大しているのは，集団通信開始関数と同様にメッセージイズに依存してセグメント分割数が増えるためである．

一方，今回のプログレススレッドによる実装に関しては準備関数実行の際にスレッド生成を行っているため，このような実行時間となった．これは，MPI_Init 時にプログレススレッドをプールしておくことで解消できる．

## 5. 関連研究

Ajima ら [4] は，Tofu2 のオフロード機能を利用する例として Bcast や Gather の非同期集団通信のアルゴリズムを報告している．しかしながら，これらは Tofu2 のネットワークトポロジを考慮したものではない．このため，実際の運用では性能の観点から実用的であるとは言えない．さらに，対象は非同期集団通信を想定しており，永続型集団通信として実装の考慮してものでもない．

また，Hatanaka ら [5] は，Tofu2 のオフロード機能を利用した永続型隣接集団通信の実装を行った．しかし，ここでは多段の依存関係を持つブロードキャストのような永続型集団通信の実装は行われていない．

筆者ら [6] は，Tofu2 のオフロード機能を用いてブロードキャストアルゴリズムの実装を行った．しかし，プログレススレッドを用いた際の永続型集団通信の実装は行っておらず，これらの性能を比較する形での評価も行われていない．

オフロード機能を持つ通信デバイスとして Mellanox ConnectX-2 や Bull eXascale Interconnect[7] などが挙げられる．これらを利用して非同期集団通信の実装 [8] などが行われているが，永続型集団通信の実装については報告はない．

南里ら [9] は，非同期集団通信の通信隠蔽効果の調査を行った．FX100 のアシスタントコアによる非同期集団通信や Mellanox 社のオフロード機能である SHArP 機能を用いた非同期集団通信の隠蔽効果を調査を行っている．しかし，永続型集団通信の調査は行っていない．

## 6. おわりに

本稿では，オフロード機能を用いた場合とプログレススレッドを用いた場合の 2 種類の永続型集団通信の実装を行い，それぞれの通信性能や各種関数のコストについて評価を行った．オフロード機能よる実装が 64MiB 以下ではプログレススレッドの実装に対して高速であることを確認した．また，大メッセージサイズでは徐々に性能差がなくなり，128MiB ではプログレススレッドによる実装の方が 560MiB/sec ほど高速であること確認した．

今後の課題としては，以下が挙げられる．まず，計算と通信のオーバラップを行った際の性能の評価を行う．また，今回はブロードキャストのみの実装となったが，他の永続型集団通信についても同様に評価実験を行う．また，計算負荷やメッセージサイズなどと性能の関係をモデル化し，計算機や対象アプリケーションに対して適切な永続型集団通信の設計が行えるような技術を確立する

**謝辞** 本論文の一部は，「特定先端運営費等補助金/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] MPI Forum. http://mpi-forum.org/.
[2] T. Adachi, N. Shida, K. Miura, S. Sumimoto, A. Uno, M. Kurokawa, F. Shoji, and M. Yokokawa. The design of

ultra scalable mpi collective communication on the k computer. *Computer Science - Research and Development*, Vol. 28, No. 2-3, pp. 147–155, may 2013.

[3] G. Almasi, P. Heidelberger, C. J. Archer, X. Martorell, C. C. Erway, J. E. Moreira, B. Steinmacher-Burow, and Y. Zheng. Optimization of mpi collective communication on bluegene/l systems. In *the 19th annual international conference on Supercomputing*, pp. 253–262, jun 2005.

[4] Y. Ajima, T. Inoue, S. Hiramoto, S. Ando, M. Maeda, T. Yoshikawa, K. Hosoe, and T. Shimizu. Tofu interconnect 2. In *2014 IEEE 22nd Annual Symposium on High-Performance Interconnects*, pp. 57–62, aug 2014.

[5] M. Hatanaka, M. Takagi, A Hori, and Y. Ishikawa. Offloaded mpi persistent collectives using persistent generalized request interface. In *EuroMPI/USA 2017*, sep 2017.

[6] Y. Morie, M. Hatanaka, M. Takagi, A Hori, and Y. Ishikawa. Prototyping of offloaded persistent broadcast on tofu2 interconnect. In *SC17*, nov 2017.

[7] S. Derradji, T. Palfer-Sollier, J. Panziera, A. Poudes, and F. W. Atos. The bxi interconnect architecture. In *IEEE 23rd Annual Symposium on High-Performance Interconnects (HOTI'15)*, pp. 18–25, aug 2015.

[8] S. Di Girolamo, P. Jolivet, K. D. Underwood, and T. Hoefler. Exploiting offload-enabled network interfaces. *IEEE Micro*, Vol. 36, No. 4, pp. 6–17, aug 2016.

[9] 南里豪志, 大島聡史, 小野 謙二. 非ブロッキング集団通信の通信隠蔽効果に関する調査. 情報処理学会研究会報告ハイパフォーマンスコンピューティング (HPC), 第 2017-HPC-162 巻, pp. 1–11, dec 2017.