密結合並列演算加速機構TCAによるGPU対応GASNetの実装と評価

佐藤 賢太[1,a)]　藤田 典久[1,†1]　塙 敏博[2]　松本 和也[3,†2]　朴 泰祐[3,1]　Khaled Ibrahim[4]

概要：近年，GPUのような演算加速装置を用いたクラスタがHPC分野で多く用いられるようになってきている．筑波大学計算科学研究センターでは，ノードを跨ぐ演算加速装置間での直接通信を実現するために，密結合並列演算加速機構TCA（Tightly Coupled Accelerators）を提唱している．このTCAの実装としてPEACH2（PCI Express Adaptive Communication Hub version 2）が開発されており，ノードを跨ぐGPU間での直接通信を行うことができる．しかしながら，TCA/PEACH2を利用するためには独自のAPIを用いる必要があり，プログラミングコストが高く，既存のアプリケーションの移植も容易ではないという問題がある．本研究では，PGAS言語を対象とした通信ライブラリであるGASNetに注目し，これをGPUを対象としたPEACH2に実装する．これによって，GASNetを介して各種のソフトウェアとの互換性が生じ，TCA/PEACH2が広く利用できるようになると考えられる．既存のGASNetではGPUメモリを対象とした通信しか想定されておらず，GPUを対象とした拡張は現在開発段階にあり，本論文では現在進行中のGASNetのGPU対応の拡張についても触れる．TCA/PEACH2によるGASNetのプロトタイプ実装において，ノードを跨ぐGPU間の通信性能はTCA/PEACH2を直接使用した場合の性能と比較して，最小レイテンシの増大は15 %程度に抑えられ，ソフトウェア支援によって最大バンド幅は1.2倍の性能向上を達成した．

キーワード：GPUクラスタ，GPU間直接通信，通信ライブラリ

Implementation and Evaluation of GPU-aware GASNet by Tightly Coupled Accelerators

Kenta Sato[1,a)]　Norihisa Fujita[1,†1]　Toshihiro Hanawa[2]　Kazuya Matsumoto[3,†2]
Taisuke Boku[3,1]　Khaled Ibrahim[4]

Abstract: Recently, PC clusters equipped with GPU as accelerators are widely spread and operated. We have been proposing Tightly Coupled Accelerators (TCA) architecture to realize inter-node direct communication among GPUs, and we developed PCI Express Adaptive Communication Hub version 2 (PEACH2) as a prototype of TCA implementation. However, currently non-standard unique API is required to use TCA/PEACH2, so that the programming cost is expensive and porting of existing applications is not easy. On the other hand, GASNet library developed by Lawrence Berkeley National Laboratory provides low-level communication layer for Partitioned Global Address Space (PGAS) languages such as Unified Parallel C (UPC), Co-Array-Fortran and XcalableMP (XMP), and so on. GASNet assumes only CPU memory as communication target, and extension for GPU-aware GASNet is work in progress now. Beside of GPU-aware GASNet development on commodity network such as InfiniBand, we implement it on TCA/PEACH2 to provide general programming and system software porting on this hardware in this paper. We also mention currently planned features of GPU-aware GASNet. In the case of inter-node GPU communication using GASNet prototype implementation by TCA/PEACH2, the minimum latency is increased only 15 % from the case with native API, and the maximum bandwidth is increased by 1.2 times of native API thanks to software support.

Keywords: GPU cluster, direct communication between GPUs, Communication library

1. はじめに

近年，GPU のような高い演算性能とメモリバンド幅を持つ演算加速装置を用いたクラスタが HPC 分野で広く用いられるようになってきている．Top500 List[1] においても，2015 年 11 月のリストで全体の 13 %のシステムが演算加速装置として GPU を利用している．一般に，GPU は PCIe（PCI Express）バスを介して CPU やノード内の他の GPU と接続されているが，PCIe の転送バンド幅は GPU のメモリバンド幅よりも低く，GPU アプリケーションにおいてボトルネックとなることも多い．また，ノードを跨ぐ GPU 間の通信には IB（InfiniBand）などのコモディティネットワークを介する必要があり，PCIe と IB 間でのプロトコル変換などが必要となるため，通信レイテンシが増大し，強スケーリングの達成が困難となる．

このような問題を解決するために，筑波大学計算科学研究センターでは，ノードを跨ぐ演算加速装置間での直接通信を実現する密結合並列演算加速機構 TCA（Tightly Coupled Accelerators）を提唱しており，その実装として，PEACH2（PCI Express Adaptive Communication Hub version 2）を開発している [2]．TCA は演算加速装置間を直接的通信網で結合するというコンセプトであり，PEACH2 は FPGA を用いて PCIe を対象として実装した TCA のプロトタイプである．以後，PEACH2 による TCA 実装を TCA/PEACH2 と記す．現在，TCA/PEACH2 を利用するためには独自の API を用いる必要があるため，MPI と比べてプログラミングコストが高く，既存のアプリケーションの移植が容易ではないという問題がある．

GASNet はプログラミング言語や通信インタフェースなどに依存しない低レベルな通信レイヤの提供を目指して開発されている通信ライブラリである [3]．GASNet の API は UPC（Unified Parallel C）[4] や Co-array Fortran[5]，XMP（XcalableMP）[6] などの PGAS（Partitioned Global Address Space）言語において，ランタイムやコンパイラが生成したコードなどが利用することを想定して設計されており，MPI と比べて機能は乏しいが軽量なライブラリとなっている．しかし，既存の GASNet は CPU メモリを対象とした通信しか想定されておらず，GPU メモリに対応していない．このため，現在，GPU メモリを通信対象とする GASNet の GPU 対応拡張が進められている．

以上の背景のもと，本論文では，GPU 対応 GASNet を TCA/PEACH2 によって実装する．この実装は，現在開発中の GPU 対応 GASNet を基準としており，今後の同システムをベースとするシステムへの適用を想定している．これによって，GASNet や上位レイヤの PGAS 言語などを介して各種ソフトウェアとの互換性が生じ，TCA/PEACH2 を広く利用できるようになると期待される．本稿では，GPU 対応 GASNet の基本的な通信性能について，TCA/PEACH2 と IB のそれぞれの実装で比較し，評価を行う．また，TCA/PEACH2 による実装では，姫野ベンチマークの性能について評価を行う．

[1] 筑波大学大学院 システム情報工学研究科
Graduate School of System and Information Engineering, University of Tsukuba
[2] 東京大学 情報基盤センター
Information Technology Center, The University of Tokyo
[3] 筑波大学 計算科学研究センター
Center for Computational Sciences, University of Tsukuba
[4] Lawrence Berkeley National Laboratory
[†1] 現在，筑波大学 計算科学研究センター
Presently with Center for Computational Sciences, University of Tsukuba
[†2] 現在，日本原子力研究開発機構
Presently with Japan Atomic Energy Agency
[a)] ksato@hpcs.cs.tsukuba.ac.jp

2. TCA/PEACH2

本節では，本研究を理解するための TCA および PEACH2 についての概要を説明する．これらの詳細については [2] を参照されたい．

2.1 TCA

筑波大学計算科学研究センターでは，ノードを跨ぐ演算加速装置間での直接通信を実現するために，密結合並列演算加速機構 TCA（Tightly Coupled Accelerators）を提唱している．TCA は，ノードを跨ぐ演算加速装置同士を密に結合することで，演算加速装置間での直接通信を可能とし，通信レイテンシを削減することで強スケーリングを達成するというものである．

2.2 PEACH2

TCA の実装として，PEACH2（PCI Express Adaptive Communication Hub version2）が開発されており，ノードを跨ぐ CPU 間および GPU 間での直接通信ができる．現状では，NVIDIA 社製 GPU に対応している．PEACH2 では，ノード間の接続に PCIe を用いており，PEACH2 が PCIe パケットのルーティングを行うことでノードを跨ぐ PCIe デバイス間での直接通信を実現している．

PEACH2 は 4 つの PCIe Gen2 x8 ポートを持っており，そのうち 1 ポートをホスト CPU との接続に使用し，残り 3 ポートを他のノードの PEACH2 との接続に使用する．ノード内の PCIe デバイスには，CPU 内部の PCIe スイッチを介して接続されている．PEACH2 による GPU メモリへのアクセスには，GDR（GPUDirect RDMA）[7] を利用している．

PEACH2 はリモートノードに対するアクセスとして，RDMA Write にのみ対応おり，RDMA Read などが必要な場合は，ソフトウェアで処理する必要がある．この点はアプリケーションやシステムソフトウェアの実装および性能上，大きな影響を及ぼすので注意が必要である．

2.3 DMA 通信

PEACH2 には，DMAC（DMA Controller）が 4 チャネル実装されている．これらの DMAC を利用した DMA 通信は，ホスト上であらかじめ転送元アドレスや転送先アドレス，転送サイズなどを指定したディスクリプタを作成しておき，使用する DMAC にこれを登録することで行われる．また，PEACH2 の DMAC は Chaining 機能を持っ

ており，複数のディスクリプタを連結することで，連続した DMA 処理を 1 回の DMA 要求で開始できる．ただし，ディスクリプタの作成には時間がかかるため，できる限り事前に作成しておき，以降は事前に作成しておいたディスクリプタを DMAC に登録するだけに留めるのが望ましい．また，PEACH2 では通常の連続アドレス上のデータのブロック転送の他，ブロックストライド転送も実行可能である．したがって，定型的なブロックストライド転送を DMA Chaining を用いなくても実行可能であり，さらに DMA Chaining との組み合わせも可能である．

ディスクリプタを格納する領域として，FPGA に内蔵されたメモリ（内蔵メモリモード）と，ホスト CPU のメモリ（ホストメモリモード）がある．内蔵メモリモードでは，ディスクリプタを読み出すコストが低いため，ホストメモリモードよりも通信レイテンシが低くなる．しかしながら，メモリ容量には限りがあるため，最大で 1024 個のディスクリプタしか格納できない．また，ディスクリプタを作成する度に，PEACH2 にアクセスする必要があるため，ディスクリプタ作成にかかる時間が大きい．一方，ホストメモリモードは通信レイテンシが若干増えるものの，ホストのメモリ容量の許す限りディスクリプタを格納することができ，またディスクリプタの作成はホスト内で完結するため，作成にかかる時間が小さい．このように，各モードで特性が異なるため，通信パターンなどに応じて適切に選択することで最適化が可能である．

PEACH2 の DMA 通信には，ハードウェアなどの制約から，転送元や転送先のアドレスのアラインメントなどに関する制限があり，任意のデータを対象として通信を行うことはできないので注意が必要である．また，CPU メモリを通信対象とする場合には，*tcaMalloc()* という専用の API で確保された領域のみ通信対象とすることができる．

2.4 PIO 通信

PIO（Programmed I/O）通信は CPU の Store 命令によってリモートノードのメモリに書き込みを行うというものである．DMAC を使用しないため，ディスクリプタの作成や DMAC の起動処理などが不要となり，非常に低いレイテンシで通信を行うことができるが，バンド幅は低いため，小さいデータの転送に適している．また，PIO 通信は転送先のメモリとして *tcaMalloc()* で確保された CPU メモリのみ対応しており，DMA 通信と同様にアラインメントに関する制限もあるので注意が必要である．

2.5 HA-PACS/TCA

筑波大学計算科学研究センターでは，TCA/PEACH2 の実験用システムとして，HA-PACS/TCA（Highly Accelerated Parallel Advanced System for Computational Sciences/TCA）が運用されている [8]．表 1 にノード構成を示す．本稿における今後の性能評価は，このクラスタを用いて行う．

表 1　HA-PACS/TCA のノード構成

ハードウェア	
CPU	Intel Xeon-E5 2680v2 2.8 GHz × 2 sockets
Memory	DDR3 1866 MHz × 4 ch，128 GB
GPU	NVIDIA Tesla K20X × 4
GPU Memory	GDDR5 2600 MHz，6 GB/GPU
InfiniBand	Mellanox ConnectX-3 QDR 4X Dual-port
PEACH2	Altera Stratix IV GX，EP4SGX530
ソフトウェア	
OS	CentOS 6.4
CUDA	CUDA 6.5
MPI	MVAPICH2-GDR 2.1a

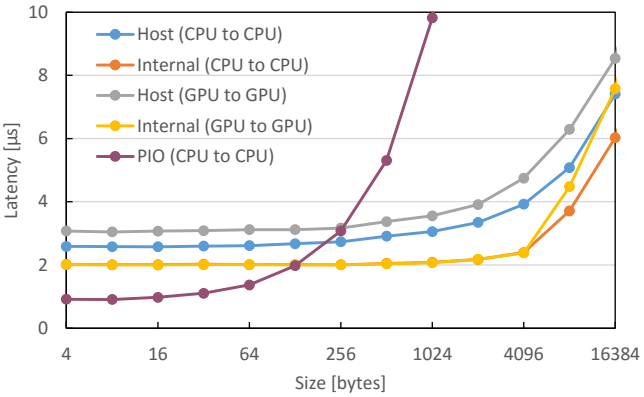

図 1　PEACH2 のレイテンシ

2.6 PEACH2 の基本性能

今後の実装や性能評価の基準となる PEACH2 の基本的な通信性能を図 1 および図 2 に示す．図中の凡例における “PIO” は PIO 通信を示しており，“Internal” および “Host” は DMA 通信における DMA ディスクリプタの配置の違いで，前者は内蔵メモリモード，後者はホストメモリモードをそれぞれ示している．また，括弧内は転送元と転送先のメモリを示している．

図 1 より，PIO 通信の最小レイテンシは 0.9 μs となっており，DMA 通信の最小レイテンシは内蔵メモリモードの場合はメモリの種別に依らず 2.0 μs，ホストメモリモードの場合は CPU メモリで 2.2 μs，GPU メモリで 3.0 μs である．また，図 2 より，PIO 通信の最大バンド幅は 0.1 GB/s，DMA 通信の最大バンド幅はモードに関わらず，CPU メモリの場合は 3.5 GB/s，GPU メモリの場合は 2.9 GB/s である．

3. GASNet

本節では，本研究を理解するために GASNet やその API についての概要と，現在 LBNL で進められている GPU 対応拡張について説明する．API の詳細については [3] を参照されたい．

GASNet は図 3 に示すように Core API と Extended API の 2 つの通信レイヤに分かれている．Extended API には，Core API のみを用いて実装されたリファレンス実装があり，Core API を実装することで GASNet の全ての API が利用可能となる．そして Extended API の特定の API のみを置き換えることが可能となっているため，ハードウェ

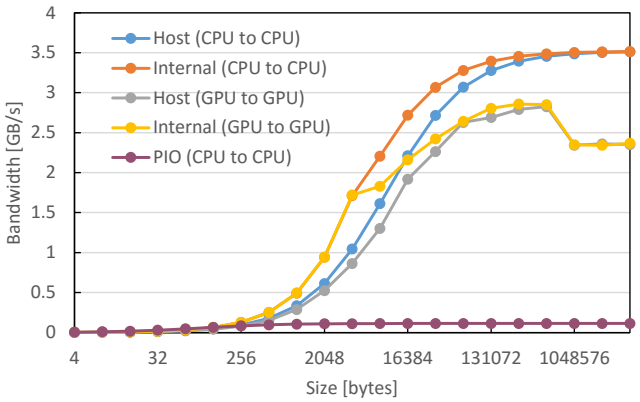

図 2　PEACH2 のバンド幅

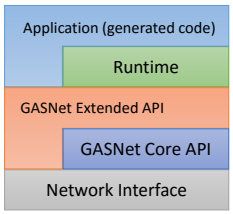

図 3　GASNet のソフトウェアスタック

アが対応している処理などは，リファレンス実装を使用せずにローカライズされた実装をすることで通信性能の最適化が可能である．

3.1　Core API

Core API は，GASNet の初期化や終了処理などの制御系の API と AM（Active Message）による通信 API で構成されている．AM は RPC（Remote Procedure Call）の一種で，あらかじめハンドラテーブルにハンドラ関数を登録しておき，リモートノードから，ハンドラや引数，データを指定し，ハンドラに対応する関数を呼び出すというものである．GASNet における AM は，データ転送の可否や転送先のメモリ領域などに応じて Short，Medium，Long の 3 種類が定義されている．以後，これらをそれぞれ *AM_Short()*，*AM_Medium()*，*AM_Long()* と記す．

AM_Short() では，データを転送はできず，ハンドラと引数のみを転送できる．そして，*AM_Medium()* と *AM_Long()* では *AM_Short()* に加えてデータも転送することができる．*AM_Medium()* におけるデータの転送先はシステムが管理するバッファとなるが，*AM_Long()* ではユーザが指定した領域を転送先として指定することができる．

3.2　Extended API

Extended API は *put()* や *get()* といった RMA（Remote Memory Access）通信やバリア同期などの API で構成されている．また，非公式 API として各種の Collective 通信や非連続なデータ転送のための API も存在し，非連続データの転送に関しては，Vector，Indexed，Strided の 3 種類があり，それぞれに *put()* と *get()* が用意されている．これらの API は現時点では非公式だが，現在開発が進められている GASNet-EX[9] では，公式 API になる予定である．

3.3　Segment

GASNet では，ライブラリ初期化時にユーザが指定したサイズのメモリを各ノードで確保する．このメモリ領域を Segment と呼び，リモートノードからアクセスされるメモリは，この領域に収まっている必要がある．つまり，*AM_Long()* や *put()* における転送先と，*get()* における転送元は Segment 内に限定される．

3.4　GPU 対応

前節で説明したように，GASNet ではリモートノードからアクセスされるメモリに関しては Segment 内である必要がある．そのため，Segment に GPU メモリを割り当てることで，リモートノードの GPU メモリへのアクセスが可能となる．ただし，Segment は単一の連続領域である必要があるため，Segment に GPU メモリを割り当てた場合，基本的にリモートノードの CPU メモリにアクセスすることはできなくなる．例外として，*AM_Medium()* に関しては，転送先はシステムが管理するバッファであり，Segment 内である必要はない．ここに CPU メモリを割り当てることで，転送サイズなどに制限はあるものの，リモートノードの CPU メモリにデータを転送することができる．通常，GPU を利用するアプリケーションでは，主として GPU to GPU の通信行うため，この問題による大きな影響はないと考えられる．また，GASNet-EX[9] では，各ノードが複数の Segment を持つことが可能となる予定であり，CPU メモリと GPU メモリをそれぞれ異なる Segment に割り当てることで，このような問題は生じないと考えられる．一方で，ローカルのメモリ領域に関しては特別な規定はなく，CPU メモリと GPU メモリの両方を扱うことができる．

4.　実装

本節では，本研究で開発した TCA/PEACH2 による GPU 対応 GASNet の実装について説明する．

4.1　概要

GASNet を実装するのに必要な機能として，各種の制御メッセージを送受信する機能と，任意のデータに対応したデータ転送機能が挙げられる．Core API で必要な AM は，必要に応じてデータの転送を行い，その後にハンドラや引数などの情報を含むメッセージをリモートノードに送信することで実装できる．Extended API の *put()* に関してはデータ転送機能そのものであり，*get()* に関してもメッセージの送受信やデータの転送の組み合わせで実装できる．また，非公式 API のストライド転送機能は，PEACH2 にブロックストライド転送機能を利用することで，高い性能が期待できる．Extended API の他の機能に関しては，PEACH2 との親和性が低く，直接実装しても高い性能が得られるとは考えにくいため，リファレンス実装を利用する．

4.2　パケット通信

図 4 にパケット通信機構の構造を示す．2.2 節で説明したように，PEACH2 ではリモートノードのメモリ領域に

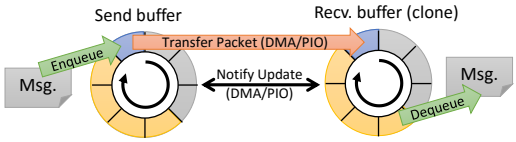

図 4　パケット通信機構

対する RDMA Write しか行うことができない．そのため，本実装では，各ノードペア間にリングバッファを用意し，送信バッファから受信バッファに対して RDMA Write を行うことでパケットを転送する．これによって，ローカルの送信バッファとリモートノードの受信バッファには全く同じデータが格納されている状態になる．その後，リモートノードに対してリングバッファの更新を通知することで，リモートノードはパケットの到着を検出することができる．リモートノードでは，到着したパケットの受信処理が完了すると，送信元に受信完了を通知する．これによって，リングバッファのエントリが再利用可能な状態となる．この機構では，送信側が受信バッファの空き状況を把握することができるため，受信バッファに空きがない場合は送信を停止することでフロー制御が可能となっている．

リングバッファに用いるメモリ領域は，ライブラリ初期化時に *tcaMalloc()* で確保し，固定長単位で利用する．ただし，実際に転送されるパケットのサイズは可変となっており，転送するメッセージ長に応じて変化する．2.6 節での測定結果より，転送サイズが小さい場合は PIO 通信の方が高速な転送が可能である．そのため，パケットの転送は，パケットサイズが小さい場合は PIO 通信を利用し，大きい場合は DMA 通信を利用する．

4.3　データ転送

2.3 節や 2.4 節で説明したように，PEACH2 の DMA 通信や PIO 通信にはいくつかの制限があり，任意のデータを転送できるわけではない．このような制限はユーザが TCA の API を用いて直接プログラムを開発する場合はそれほど問題にはならないが，本研究のように通信ライブラリを実装する場合は，上位レイヤのアプリケーションなどにハードウェアの制限を意識させないための一般化が必要であり，任意のデータを柔軟に転送できるようにする必要がある．

前節で説明したパケット通信によってデータを転送する場合，データは送信バッファと受信バッファを経由するため，任意のデータを転送することができる（図 5(a)）．しかし，この方法では 2 回のメモリコピーが行われるため高い通信性能は期待できない．他方，3.3 節で説明したように，GASNet ではリモートノードからアクセスされる領域は Segment 内である必要があり，転送元と転送先のどちらか一方は必ず GPU メモリとなる．具体的には，*AM_Long()* と *put()* の場合は転送先が Segment 内となり，*get()* の場合は転送元が Segment 内となり，必ず GPU メモリとなる．転送元と転送先の両方が Segment 内の場合は，転送元から転送先まで DMA 通信によって直接転送を行う（図 5(b)）．この場合，メモリコピーは発生しないため，高い通信性能が得られると考えられる．また，転送先が Segment 内の場合は，送信バッファから転送先までは DMA 通信によって直接転送でき（図 5(c)），逆に転送元が Segment 内の場合は，転送元から受信バッファまでは DMA 通信によって直接転送できる（図 5(d)）．この場合も，メモリコピーは 1 回で済むためパケット通信を利用する場合よりは高い通信性能が得られると考えられる．以後，パケット通信によるデータ転送を BtoB（Buffer to Buffer）モード，DMA 通信による直接転送を MtoM（Memory to Memory）モード，送信バッファから転送先への転送に DMA 通信を使うものを BtoM（Buffer to Memory）モード，転送元から受信バッファへの転送に DMA 通信を使うものを MtoB（Memory to Buffer）モードと記す．

パケット通信では，パケットサイズが小さい場合は PIO 通信を利用するため，DMA 通信を用いるモードよりも高速にデータを転送できる．そのため，転送するデータサイズが小さい場合は，その他のモードが利用可能な場合でも BtoB モードを利用して転送を行う．また，転送元や転送先のアラインメントがずれている場合は，転送するデータの先頭から数バイトのみを BtoB モードで転送してアラインメントの調整を行う．これによって，どのような場合であっても MtoM，BtoM，MtoB のいずれかの転送モードが利用可能となり，BtoB モードは基本的に DMA 通信よりも PIO 通信の方が高速な小さいデータの転送にのみ利用されることになる．

MtoM モード以外のモードではメモリコピーが必要となるが，メモリコピーと DMA 通信をオーバーラップすることで，メモリコピーにかかる時間を隠蔽することができる．また，MtoB モードで使用する受信バッファは受信側で転送先へのメモリコピーなどを行う必要があり，フロー制御が必要なため，BtoB モードのパケット通信で使用する受信バッファをそのまま使用する．BtoM モードで使用する送信バッファに関しても，BtoB モードで使用する送信バッファをそのまま使用することもできるが，敢えて専用の送信バッファを 2 つ用意し，ダブルバッファリングの要領で交互に使用する．この理由に関しては，次節で説明する．

4.4　ディスクリプタキャッシュ

2.3 節で説明したように，DMA 通信で必要なディスクリプタの作成には時間がかかるため，できる限り事前に作成したものを再利用するのが望ましい．しかし，通信ライブラリを実装する場合は，事前にどのような通信を行うかを予測して作成しておくのは困難である．一方で姫野ベンチマークを始め，HPC アプリケーションでは転送元や転送先の領域や転送サイズが変化しない固定パターンの通信を何度も繰り返すというアプリケーションが数多く存在する．このようなアプリケーションでは，事前に通信パターンを予測できなくても，一旦作成したディスクリプタをそのまま再利用できる可能性は高いと考えられる．もちろん，ユーザが直接 TCA/PEACH2 を使ってアプリケーションを記述する場合は，ディスクリプタの再利用は自然に行われるが，本研究では一般化された通信ライブラリを実装するため，ディスクリプタの再利用性をライブラリ内で管理する必要がある．そこで，本実装では一旦作成したディス

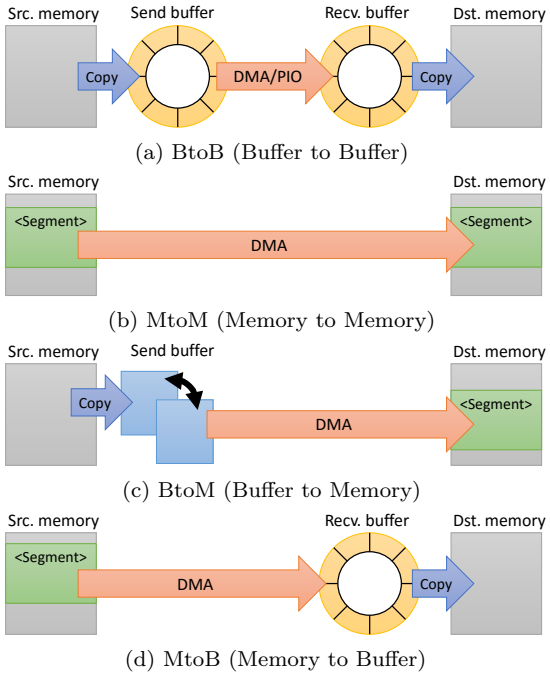

図 5 データ転送モード

クリプタをキャッシュする機構を備える．このようにディスクリプタをキャッシュすることによって，ディスクリプタ作成に時間がかかるという問題をある程度緩和できると考えられる．

キャッシュのアルゴリズムに関しては，対象とするアプリケーションの通信パターンによって最適なものが異なる．そこで，本実装では LRU と LFU のような一般的なキャッシュアルゴリズムの他，キャッシュエントリが埋まるまで先着順に割り当てるというアルゴリズムを実装しており，プログラム実行時に切り替えることで，アプリケーションの特性に応じた最適化を可能としている．また，2.6 節での測定結果より，ホストメモリモードのディスクリプタによる DMA 通信の最小レイテンシは内蔵メモリモードよりも大きいため，頻繁に発生する通信には内蔵メモリモードでディスクリプタを作成するのが望ましい．一方で，内蔵メモリモードは作成できるディスクリプタ数に限りがあり，ホストメモリモードよりも作成に時間がかかるため，低頻度な通信での利用や頻繁な置換は避けるべきである．一旦ホストメモリモードで作成した後，使用回数が一定回数以上となったディスクリプタを内蔵メモリモードで再作成すれば，特に頻繁に発生する通信のみを内蔵メモリモードで作成することができる．一般的に，HPC アプリケーションでは，同じデータ通信パターンが多数反復されることが多く，この機構は有効に動作すると考えられる．ただし，通信パターンが少なく，全てのディスクリプタを内蔵メモリモードで作成できる場合などに対応するため，この機構は実行時に無効化することも可能としている．また，MtoM モードでは通信パターン毎にディスクリプタを 1 つ，BtoM モードでは送信バッファが 2 つあるため，通信パターン毎にディスクリプタを 2 つ使用するが，BtoB モードや MtoB モードでは通信パターン毎にバッファのエントリ数分のディスクリプタを使用するため，キャッシュしたディスクリプタの利用効率が悪く，内蔵メモリを大量に消費してしまう．そのため，内蔵メモリモードの利用は MtoM モードと BtoM モードに限定している．また，BtoM モードで送信バッファに BtoB モードの送信バッファを使用せずに専用のバッファ領域を用意するのはこれが理由である．

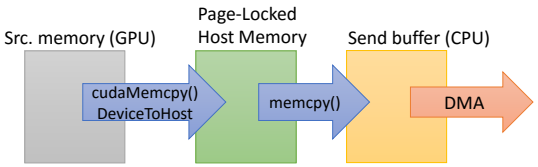

図 6 Page-Locked Host Memory を経由したデータ転送

4.5 gdrcopy

MtoM 以外のデータ転送モードでは，Segment である GPU メモリと送信バッファや受信バッファとの間でメモリコピーが必要となる．通常，GPU メモリのコピーには CUDA[10] の API である *cudaMemcpy()* を利用するが，*cudaMemcpy()* によるメモリコピーはレイテンシが大きいという問題がある．そこで，本実装では，GPU メモリのコピーに gdrcopy[11] を利用する．

gdrcopy は NVIDIA によって提供されている GDR を用いて GPU メモリのコピーを行うためのライブラリである．gdrcopy によるメモリコピーの特徴として，レイテンシが非常に小さく，CPU to GPU のバンド幅が高いというものがある．一方で，GPU to CPU のバンド幅は非常に低いため，サイズが小さい場合にしか利用できない．しかし，GPU to CPU の転送は基本的に MtoB モードで転送されるため，GPU メモリのコピーはアラインメントの調整でのみ行われ，それほど大きな影響はないと考えられる．

4.6 Page-Locked Host Memory

2.6 節での測定結果より，PEACH2 の DMA 通信で GPU メモリを転送した場合のバンド幅は 512 KB をピークとしてこれ以上の転送サイズでは低下している．この現象は GDR に対応した IB HCA を用いて GPU メモリを転送した場合でも同様に観測されるため，GDR を用いた GPU メモリの読み出し性能の限界だと考えられる．一方で，Page-Locked Host Memory[12] を転送先として，*cudaMemcpy()* を用いて GPU メモリのコピーを行うと，GDR を用いる場合よりも高いバンド幅を得ることできる．そこで，MtoM モードや MtoB モードで大きいデータの転送を行う際に，一旦 Page-Locked Host Memory を経由することで，GDR による読み出し性能の限界を超えたバンド幅での転送が可能になると考えられる．ただし，Page-Locked Host Memory は DMA 通信の通信対象とすることはできないため，図 6 に示すように，送信バッファを経由する必要がある．したがって，MtoM モードは実際には BtoM モードでの転送となり，MtoB モードは実際には BtoB モードでの転送となる．

5. 性能評価

本節では，本研究で実装した GASNet の基本的な通信性

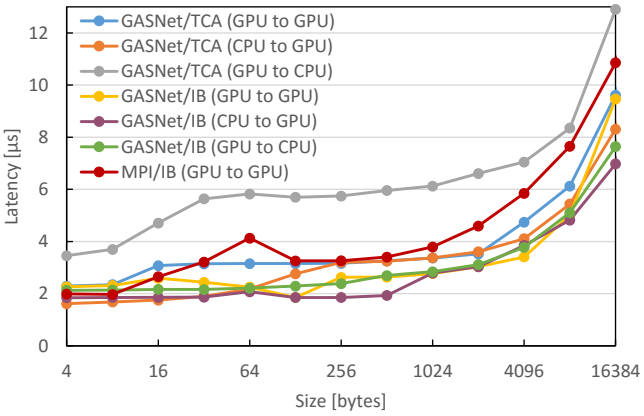

図 7　GPU メモリを転送する場合のレイテンシ

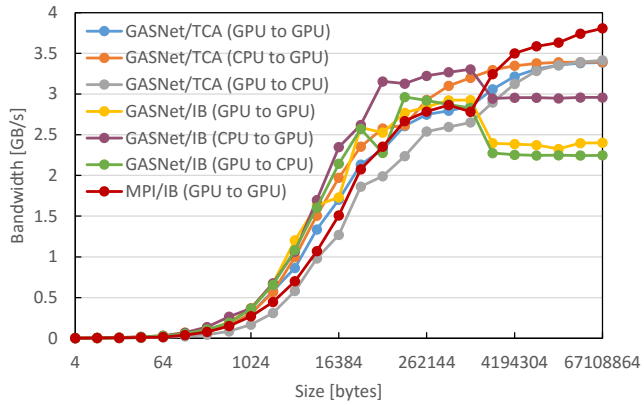

図 8　GPU メモリを転送する場合のバンド幅

能と袖領域通信の性能，GASNet を用いて姫野ベンチマークを実装したときの性能について評価を行う．また，HA-PACS/TCA に搭載されている IB HCA は QDR Dual-port だが，本研究では 1 ポートのみ使用する．

5.1　基本通信性能

通信性能の評価は，GASNet の仕様上利用可能な GPU to GPU，CPU to GPU，GPU to CPU の 3 種類の転送パターンについて行う．GPU to GPU と CPU to GPU に関しては，*put()* と *AM_Long()* で利用可能であるが，*put()* を利用する場合は，受信側で転送完了を検出するために GPU メモリのポーリングが必要となり，オーバヘッドが大きい．そのため，転送先が GPU メモリの場合は *AM_Long()* による Ping-Pong 通信によって評価を行う．GPU to CPU に関しては，*get()* の処理時間によって評価を行う．また，GPU to GPU に関しては，姫野ベンチマークで利用するため，MPI でも *MPI_Send()* と *MPI_Recv()* によって Ping-Pong 通信を行い GASNet と比較する．

図 7 にレイテンシを，図 8 にバンド幅を示す．凡例は，通信に使用した実装の種類を示しており，“GASNet/TCA” は TCA/PEACH2 による GASNet 実装，“GASNet/IB は IB” による GASNet 実装，“MPI/IB” は MPI による通信を示している．また，括弧内は転送元と転送先のメモリ種別を示している．

GPU to GPU の最小レイテンシは，GASNet/TCA の場合は 2.3 μs，GASNet/IB の場合は 1.9 μs，MPI/IB の場合は 2.1 μs となっており，GASNet/IB が最も低いことがわかる．GASNet/TCA と MPI/IB でレイテンシを比較すると，最小レイテンシは MPI/IB のほうが低いものの，32 B ～ 64 KB では GASNet/TCA の方が低いことがわかる．最大バンド幅に関しては，GASNet/TCA の場合は 3.4 GB/s，GASNet/IB の場合は 2.9 GB/s，MPI/IB の場合は 3.8 GB/s となっており，MPI が最も高いことがわかる．GASNet/IB のバンド幅は 1 MB をピークとして低下しているが，これは PEACH2 の DMA 通信と同様に GDR による GPU メモリの読み出し性能の限界だと考えられる．一方で，GASNet/TCA では，このようなバンド幅の低下は見られず，PEACH2 における DMA 通信の最大バンド幅の 1.2 倍の性能が得られている．これは，4.6 節で説明した Page-Locked Host Memory を経由した GPU メモリの転送が有効に機能しているためだと考えられる．また，MPI/IB でもバンド幅の低下は発生しておらず，本研究と同様の通信の最適化が行われていると考えられる．

CPU to GPU の最小レイテンシは，GASNet/TCA の場合は 1.6 μs，GASNet/IB の場合は 1.8 μs となっており，転送元が CPU メモリのため GPU to GPU よりも低いレイテンシで通信が行えていることがわかる．GASNet/TCA の最小レイテンシは PEACH2 の DMA 通信よりも低く，PIO 通信と比較したオーバヘッドは 0.7 μs である．最大バンド幅に関しては，GASNet/TCA の場合は 3.4 GB/s，GASNet/IB の場合は 3.3 GB/s となっている．

GPU to CPU の最小レイテンシは，GASNet/TCA の場合は 3.5 μs，GASNet/IB の場合は 2.1 μs となっており，*get()* を行っているため 3 種類の転送パターンの中で最もレイテンシが大きい．IB はハードウェアで *get()* を処理することができるためレイテンシの増加はそれほど大きくないが，TCA/PEACH2 ではソフトウェアで処理を行っているため最小レイテンシが大きく，さらに，MtoB モードによって転送を行っているため，内蔵メモリモードが利用できず，全体的にレイテンシが大きくなっていると考えられる．最大バンド幅に関しては，GASNet/TCA の場合は 3.4 GB/s，GASNet/IB の場合は 3.0 GB/s となっており，GASNet/IB では GPU to GPU と同様のバンド幅の低下が見られる．

5.2　袖領域通信の性能

多次元ステンシル計算などでは，連続領域のデータ転送だけでなく，ブロックストライドあるいはストライド配置のデータ転送性能も重要となる．TCA/PEACH2 では，このような需要を想定して，ハードウェア実装によるブロックストライド転送機能を持っている．GASNet の Extended API においても非公式ではあるが，ストライド転送の API が用意されており，本研究ではこれに TCA/PEACH2 実装を適用する最適化を行っている．

本節では，GPU メモリに確保した 3 次元配列の各面を対象とした Ping-Pong 通信によって袖領域通信の性能評価を行う．3 次元配列を対象とした袖領域通信では，転送する各面に応じて，ブロック転送，ブロックストライド転送，ストライド転送が行われる．GASNet ではストライド転送機能を利用してデータを転送した後，*AM_Short()* を利用し

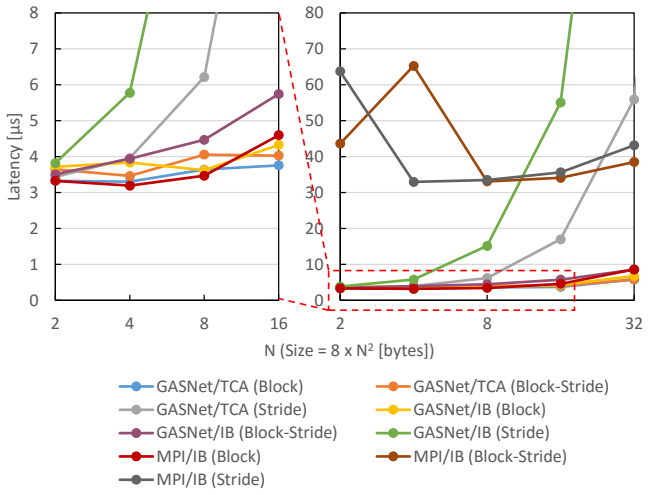

図 9 袖領域通信のレイテンシ

て転送完了を通知することで Ping-Pong 通信を行う．そして，MPI では *MPI_Type_vector()* を利用して，*MPI_Send()* と *MPI_Recv()* を行うことで Ping-Poing 通信を行う．また，配列の各要素は倍精度浮動小数点数（8bytes）とする．

図 9 にレイテンシを，図 10 バンド幅を示す．図より，GASNet/TCA のブロックストライド転送は，ブロック転送とほぼ等しい性能が得られており，全体的に MPI よりも大幅に高い性能が得られていることがわかる．GASNet/IB と比べた場合は，MPI ほどの性能差はないものの，$N = 4 \sim 64$ の範囲では GASNet/TCA の方が 10 %以上小さいレイテンシで転送できており，$N = 32$ では 30 %以上小さいレイテンシで転送できている．これらの結果より，小領域サイズにおける TCA/PEACH2 におけるブロックストライド転送のハードウェア実装の有効性がわかる．また，$N = 320$ 以上の大領域サイズにおいては，前節での測定同様に GASNet/IB ではバンド幅が大幅に低下してしまっているが，GASNet/TCA ではバンド幅の低下は見られず，本研究における通信の最適化がブロックストライド転送においても有効に機能していることがわかる．一方で，ストライド転送に関しては，GASNet/TCA の場合で $N = 16$ 以下，GASNet/IB の場合で $N = 8$ 以下の小領域サイズでは MPI よりも低いレイテンシで転送できているが，これ以上のサイズでは MPI よりも低い性能となっている．これは，GASNet/TCA や GASNet/IB では Pack/Unpack を行わずに細粒度の転送を行っているのに対して，MPI では非連続データを Pack/Unpack して連続データとして転送しているため，レイテンシは大きくなってしまうものの，ストライド転送でもブロックストライド転送と同程度の性能が得られていると考えられる．

5.3 姫野ベンチマークの性能

本節では，姫野ベンチマークを用いて TCA/PEACH2 による GPU 対応 GASNet の性能評価を行う．

姫野ベンチマークは非圧縮性流体解析コードの性能評価のために開発されたもので，3 次元ポアソン方程式をヤコビ反復法で解く場合の処理速度を測定するプログラムである．本稿では，表 2 に示す問題サイズを使用し，ヤコビ

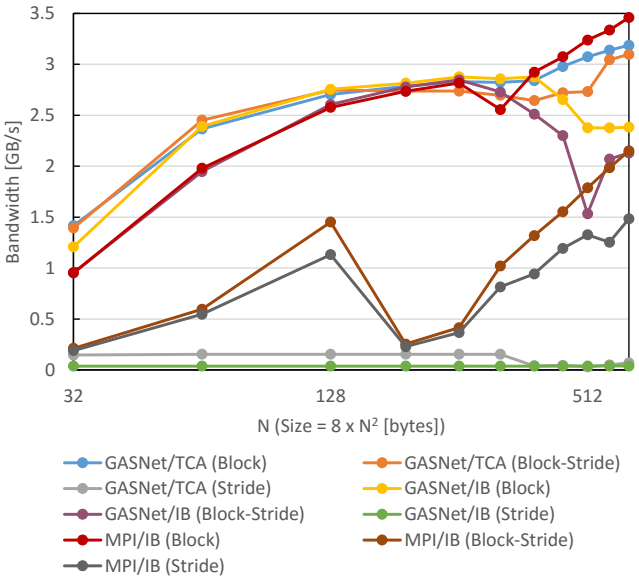

図 10 袖領域通信のバンド幅

法の反復回数を 1000 回に固定した際の処理時間によって性能を評価する．GASNet による姫野ベンチマークは，オリジナルの MPI 版 [13] を CUDA に移植したもの [14] に対して NVIDIA Kepler アーキテクチャ向けの最適化を施したもの [15] をベースとして実装している．姫野ベンチマークにおける通信は，袖領域の交換と収束判定のための Allreduce がある．袖領域の交換には GASNet のストライド転送機能を使用している．収束判定に関しては，GPU メモリに存在する誤差の値を gdrcopy を使用して，CPU メモリにコピーしてた後，Allreduce を行っている．Allreduce は *AM_Medium()* を使用して Dissemination 法 [16] によって実装している．

表 2 姫野ベンチマークにおける問題サイズ（$i \times j \times k$）

Small	Middle	Large
64 × 64 × 128	128 × 128 × 256	256 × 256 × 512

測定結果を図 11，12，13 にそれぞれ示す．図中の凡例における “Calc.” は計算にかかった時間，“Halo exch.” は袖領域の交換にかかった時間，“Convergence” は収束判定のための Allreduce にかかった時間をそれぞれ示している．図より，i 方向で分割を行った場合に関しては，全ての問題サイズと分割パターンにおいて，GASNet/TCA，GASNet/IB，MPI でほぼ等しい性能となっていることがわかる．一方で，j 方向で分割を行った場合に関しては，GASNet/TCA と GASNet/IB は MPI と比べて高い性能が得られており，GASNet のストライド転送機能の有効性がわかる．しかし，GASNet/TCA と GASNet/IB の間での性能差はほぼなく，前節での測定結果と一致していないが，これは現在生じている PEACH2 の不具合への対処に起因するオーバヘッドが原因である．具体的には，DMAC における転送完了通知の処理に不具合があり，DMA 通信が完了して DMAC が完全に停止したことを正しく検出できないという問題がある．現在は DMAC の制御レジスタにある Peformance Counter の値をポーリングすることで DMAC の停止を検出している．そのため，通常の完了通知

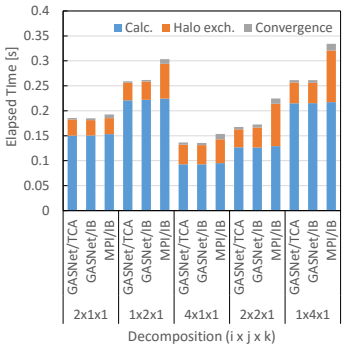

図 11 姫野ベンチマークの測定結果（Small）

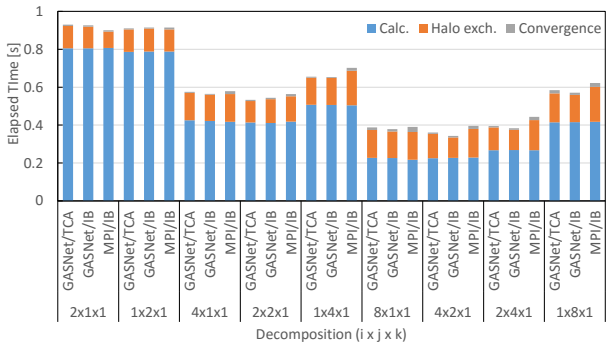

図 12 姫野ベンチマークの測定結果（Middle）

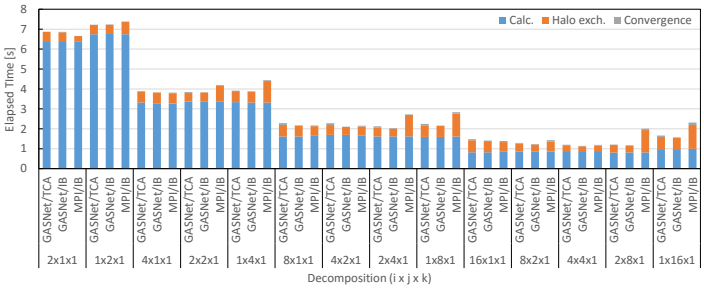

図 13 姫野ベンチマークの測定結果（Large）

を利用する方法よりも余分な時間がかかっており，姫野ベンチマークの性能が低下してしまっている．PEACH2 は FPGA による実装を行っているため，今後の改良でこの点を修正し，性能を改善する余地はあると考えている．

6. おわりに

本稿では，低レベル通信ライブラリである GASNet の GPU 対応拡張を TCA/PEACH2 を用いて実装し，基本的な通信性能や袖領域通信，姫野ベンチマークの性能に関しての評価を行った．その結果，GPU 間通信の最小レイテンシに関しては GASNet/IB や MPI よりも大きいが，比較的小さいオーバヘッドで通信を行うことができた．最大バンド幅に関しても，MPI よりは低いものの，ソフトウェア支援によって PEACH2 の DMA 通信の 1.2 倍のバンド幅が得られた．袖領域通信に関しては，ブロックストライド転送において，PEACH2 のハードウェア機能を利用することで，GASNet/IB や MPI と比べて高い性能が得られた．姫野ベンチマークに関しては，MPI よりは高い性能が得られたが，ハードウェア不具合への対処が原因で GASNet/IB と同程度の性能しか得られなかった．

今後の課題としては，ストライド転送の高速化が必要だと考えられる．ストライド転送は PEACH2 のブロックストライド転送でも実行は可能だが，高い性能は得らないため，MPI 同様に Pack/Unpack による転送に対応する必要があると考えられる．また，PEACH2 には DMAC が 4 チャネル実装されているが，現在 1 チャネルしか利用しおらず，利用されていない DMAC を活用することで，バンド幅の向上などが期待できる．

謝辞 本研究の一部は，JST-CREST 研究領域「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」，研究課題「ポストペタスケール時代に向けた演算加速機構・通信機構統合環境の研究開発」による．また，本研究における HA-PACS/TCA の利用は，筑波大学計算科学研究センター学際共同利用プログラム・平成 27 年度課題「密結合演算加速機構アーキテクチャに向けたアプリケーションの開発と性能評価」による．

参考文献

[1] Top500 Supercomputer Sites, http://www.top500.org/.
[2] 塙 敏博，児玉 祐悦，朴 泰祐，佐藤 三久：Tightly Coupled Accelerators アーキテクチャに基づく GPU クラスタの構築と性能予備評価，情報処理学会論文誌コンピューティングシステム（ACS），Vol.6，No.3，pp.14-25，2013.
[3] GASNet Communication System, http://gasnet.lbl.gov/.
[4] Unified Parallel C, http://upc.gwu.edu/.
[5] Co-array Fortran, http://www.co-array.org/.
[6] XcalableMP, http://www.xcalablemp.org/.
[7] NVIDIA GPUDirect, https://developer.nvidia.com/gpudirect.
[8] HA-PACS プロジェクト，http://www.ccs.tsukuba.ac.jp/research/research_promotion/project/ha-pacs.
[9] GASNet-EX Collaboration, https://sites.google.com/a/lbl.gov/gasnet-ex-collaboration/
[10] CUDA Toolkit, https://developer.nvidia.com/cuda-toolkit.
[11] NVIDIA gdrcopy, https://github.com/NVIDIA/gdrcopy.
[12] CUDA C Programming Guide, 3.4.2. Page-Locked Host Memory, http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#page-locked-host-memory
[13] 姫野ベンチマーク，http://accc.riken.jp/2145.htm.
[14] E. Phillips and M.fatica.：Implementing the Himeno benchmark with CUDA on GPU clusters, Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium，pp.1-10, Apr. 2010.
[15] Toshihiro Hanawa, Hisafumi Fujii, Norihisa Fujita, Tetsuya Odajima, Kazuya Matsumoto, Yuetsu Kodama, Taisuke Boku：Improving Strong-Scaling on GPU Cluster Based on Tightly Coupled Accelerators Architecture, IEEE Cluster 2015, Sep. 2015.
[16] 松本 和也，塙 敏博，児玉 祐悦，藤井 久史，朴 泰祐：密結合並列演算加速機構 TCA による GPU 間直接通信における Collective 通信の実装と性能評価，情報処理学会論文誌コンピューティングシステム（ACS），Vol.8，No.4，pp.36-49，2015.