Vol.2016-HPC-156 No.5
2016/9/15

密結合並列演算加速機構TCAにおける複数DMACの活用によるGPU対応GASNetの性能改善

佐藤 賢太[1,a)]　藤田 典久[3]　塙 敏博[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．この問題を解決し，TCA/PEACH2 を広く利用できるようにするために，我々は PGAS 言語を対象とした通信ライブラリである GASNet の GPU 対応版の実装を行っている．PEACH2 は 4 チャネルの DMAC（DMA Controller）が実装されているが，今までの実装ではそのうちの 1 チャネルしか利用されていなかった．そのため，本稿では複数チャネルの DMAC を活用した転送による性能改善についての検討および実装を行う．その結果，有効な転送サイズの範囲は限られるが，1 チャネルしか利用しない場合と比べて最大で 1.4 倍のバンド幅が得られた．

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，2016 年 6 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．

[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
[a)] ksato@hpcs.cs.tsukuba.ac.jp

以上の背景のもと，我々はTCA/PEACH2を広く利用できるようにするため，TCA/PEACH2によってGPU対応GASNetの実装を行っている [7], [8]．しかし，この実装では4チャネルあるPEACH2のDMAC（DMA Controller）のうち1チャネルしか利用していなかった．そこで，本稿ではDMACを複数チャネル利用することによる転送性能の改善を目指す．

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）[9] 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．ディスクリプタは一旦作成したものを何度も再利用することを前提にしており，ディスクリプタ作成そのものには時間がかかるが，転送を行う際はDMACに登録するだけで転送を開始できるという利点がある．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．

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 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 [10]．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．また，HA-PACS/TCAに搭載されているIB HCAはQDR Dual-portだが，QDR 1ポートとPEACH2のPCIe Gen2 x8がほぼ等しい性能となるため，本研究では1ポートのみ使用する．

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

2.5 DMACの転送性能

DMA通信は，DMACがホスト側のPCIeポートを経由してPCIeデバイスに対してRead Requestを送信し，その

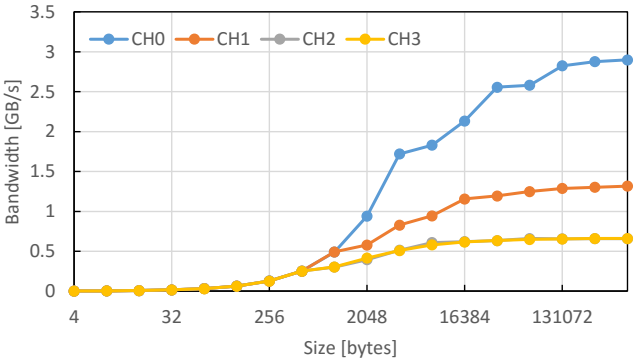

図 1　各 DMAC のバンド幅

応答として送られてくるデータを他のノードに中継することで実現されている．PCIe では，各 Read Request を識別するために TAG というものを利用するが [11]，PEACH2 の FPGA ではハード IP の制限により，TAG の個数が 64 個に制限されている．TAG の個数は，DMAC が同時に実行できる Read Request 数となるため，転送性能に直結する．PEACH2 では DMAC の独立同時動作を行うために，64 個の TAG を各 DMAC に対して，CH0 から順に 32 個，16 個，8 個，8 個を固定的に割り当てている．したがって，各 DMAC の性能の優劣は $CH0 > CH1 > CH2 = CH3$ となる．

図 1 に DMAC で GPU メモリを転送した際のバンド幅を示す．図より，DMAC の最大バンド幅は CH0 では 2.9 GB/s，CH1 では 1.3 GB/s，CH2 と CH3 では 0.7 GB/s となっている．CH0 に関しては若干性能が高いが，おおよそ DMAC に割り当てられた TAG の個数に比例した性能となっている．

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，開発元の LBNL（Lawrence Berkeley National Laboratory）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 2 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

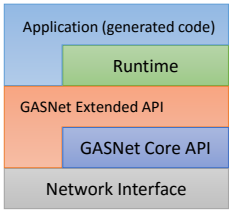

図 2　GASNet のソフトウェアスタック

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．

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．さらに，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[12] では，公式 API になる予定である．また，これらの RMA 通信の API には，ブロッキングのものとノンブロッキングのものがそれぞれが用意されている．

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．

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．既存の GASNet では Segment に CPU メモリを割り当てるため，Segment 外である GPU メモリには仕様上アクセスすることができない．そのため，GPU 対応 GASNet では Segment に GPU メモリを割り当てる．これによって，各実装での GPU メモリへの対応が必要ではあるが，仕様上はリモートノードの GPU メモリにアクセスすることが可能となる．

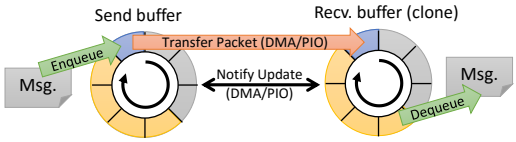

図 3　パケット通信機構

4. TCA/PEACH2 による実装

本節では，TCA/PEACH2 による GPU 対応 GASNet の実装について説明する．実装の詳細については [7], [8] を参照されたい．

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，現段階ではリファレンス実装を利用する．

4.2 パケット通信

本実装では，各種制御メッセージを送受信するために，PEACH2 の RDMA Write を用いたパケット通信を行う．図 3 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．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．

4.3 データ転送

2.3 節で説明したように，PEACH2 の DMA 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．

転送元と転送先が共に GPU メモリであれば，アラインメントの制限しかなく，一般的なアプリケーションではほとんどの場合満たされている．しかし，転送元や転送先が CPU メモリの場合は，そのメモリが専用の API で確保されている必要があり，一般的なアプリケーションがこの制限を満たしているとは考えにくい．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，アラインメントやメモリ種別に関わらず任意のデータを転送することができる（図 4(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，またはその両方が GPU メモリとなる．転送元と転送先の両方が GPU メモリの場合は，転送元から転送先まで DMA 通信によって直接転送を行い（図 4(b)），転送先が GPU メモリの場合は，送信バッファから転送先までを，逆に転送元が GPU メモリの場合は，転送元から受信バッファまでを DMA 通信によって直接転送できる（図 4(c),(d)）．このようにメモリコピーの回数を必要最低限に抑えた上で，メモリコピーと DMA 通信をオーバラップし，メモリコピーにかかる時間を隠蔽することで比較的高い性能が期待できる．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．本実装では，これら 4 つ転送モードから最適なモードを選択してデータの転送を行う．

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，転送パターンを減らし，必要なディスクリプタ数を削減するために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．

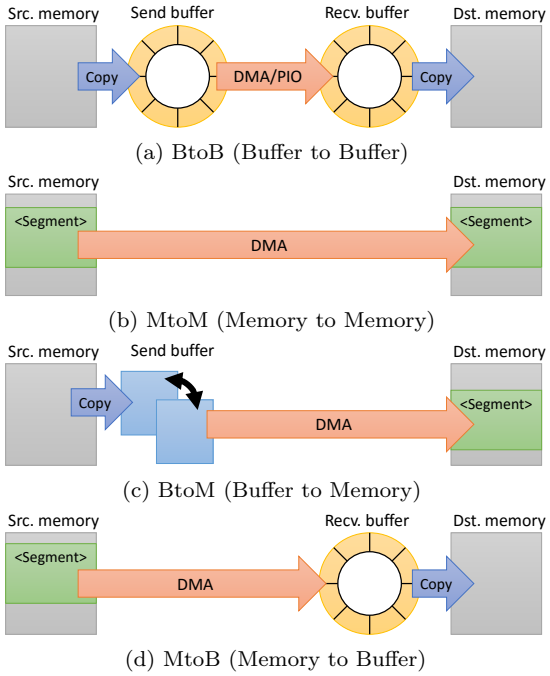

図 4　データ転送モード

ストライド転送やブロックストライド転送を実行する場合は，ブロックストライド転送でブロックサイズがある程度大きい場合は PEACH2 のブロックストライド転送機能を使用し，小さい場合やストライド転送の場合は Pack/Unpack を行い BtoB モードによって転送を行う．

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，TCA/PEACH2 ではディスクリプタは再利用することを前提としている．通常，ユーザが直接 TCA/PEAHC2 を使ってアプリケーションを記述する場合は，明示的にディスクリプタを作成し，それを DMAC にセットして転送を行うため，ディスクリプタの再利用は自然に行われる．本実装のように一般化された通信ライブラリを実装する場合は，ディスクリプタの作成と利用はライブラリ内で暗黙的に行われるため，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．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し，このようなアプリケーションでは，殆どのディスクリプタは再利用される．そこで，本実装では一旦作成したディスクリプタをキャッシュする機構を備えている．これによって，ユーザが直接 TCA/PEACH2 を使う場合と同程度にディスクリプタの再利用が行われると考えられる．

5.　複数 DMAC の利用

本節では，DMAC を複数チャネル利用することによる性能改善について述べる．

複数の通信チャネルを利用することによる性能改善手法として，(1) 一つの通信リクエストを分割して複数の通信チャネルで転送することで通信時間を短縮する方法と，(2) 各通信チャネルに別々の通信リクエストを割り当てることで，単位時間あたりに処理できる通信リクエスト数を増やす方法がある [13], [14]．

(1) の方法は，転送サイズがある程度大きい場合に限定されるが，任意の通信リクエストを内部で分割して処理することができる．高い性能を得るためには，分割した各転送が同時に完了する必要があり，各 DMAC の性能に応じた転送サイズに分割する必要がある．

(2) の方法は，転送サイズに依らず効果は得られるが，同時に複数の通信リクエストが実行される必要があるため，ノンブロッキング通信が複数実行される場合に限定される．そして，高い性能を得るためには，各 DMAC の性能と各通信リクエストの転送サイズを考慮した上で，通信リクエストを DMAC に割り当てる，といったスケジューリングが必要がある．しかし，このようなスケジューリングを行うためには，複数の通信リクエストが同時に開始される必要があるが，GASNet には MPI の永続通信（*MPI_Startall()*）のように一つの API で複数の通信リクエストを開始することはできないため，このようなスケジューリングは困難だと考えられる．

以上の要件から，本研究では比較的高い性能が期待できる (1) の方法を採用し，実装を行う．

5.1　実装

実装に関しては，4.3 節で述べたデータ転送のレイヤで通信の分割を行う．

MtoM モードや BtoM モードであれば，転送サイズと転送元・転送先のアドレスをずらした複数ディスクリプタを作成し，それらを各 DMAC で処理することで転送が分割される．この場合，ディスクリプタの作成や DMAC の起動コストは増えるが，比較的小さいオーバヘッドでの転送が可能だと考えられる．

MtoB モードや BtoB モードのようなパケット通信を利用する転送モードでは，パケットの転送自体は前述の MtoM モードや BtoM モードのように分割して行うことが可能である．しかし，パケット通信の場合は，パケットを転送した後にリングバッファの更新通知が必要となる．通常，リングバッファの更新通知はオーバヘッドを削減するために DMA Chaining を利用してパケットの転送に続けて行う．しかし，パケットを分割して転送する場合は，各 DMAC での転送が完了した後に改めて通知を転送する必要があり，DMA Chaining を利用する場合と比べてオーバヘッドが大きいと考えられる．また，リングバッファを DMAC 毎に用意して個別にパケット通信を行えば，リングバッファの更新通知は DMA Chaining で行うことができるが，リングバッファの管理やパケットの生成，送信・受

信処理を個別に行う必要があるため，オーバヘッドが大きいと考えられる．このような理由から，本研究では MtoB モードや BtoB モードでは転送の分割は行わないこととしている．

5.2 分割サイズの調整

前述したように，一つの通信を分割して複数の DMAC で転送を行う場合，高い性能を得るには各 DMAC の転送時間を揃える必要があり，分割サイズの調整が重要となる．

各 DMAC の合計バンド幅は，ホスト側の PCIe ポートの転送性能に律速されるため，サチレーションが発生すると各 DMAC のバンド幅が低下する．DMAC を複数チャネル利用するオーバヘッドなどを考慮すると，サチレーションが発生する領域では 1 チャネルしか利用しない方が高い性能が得られると考えられる．つまり，PEACH2 における複数 DMAC の利用は，本質的にあまり大きい転送サイズにおいては効果がない．また，5 節より転送サイズはある程度大きい必要がある．したがって，本研究ではこれら二つの制約を満たす，比較的狭い領域でのみ効果が得られると考えられる．そして，このような領域では通信時間はそれほど長くないため，効果を得るためには各 DMAC での転送時間を厳格に揃える必要がある．

転送時間を揃える手法として，レイテンシやピークバンド幅などの性能パラメータなどから転送時間を推定し，それをもとに分割サイズを調整するという方法が考えられるが，この手法によって厳格に転送時間を揃えるのは困難である．そこで，本研究では実際に転送を行い，その際の各チャネルの転送時間をもとに分割サイズを調整する．具体的には，各チャネルに重み付けを行い，転送毎に転送時間から重みを更新し，各チャネルに重みに比例したサイズを割り当てる．転送時間に関しては，DMAC 起動コストを含めるため，測定開始のタイミングは各チャネルで共通としている．重みを計算するアルゴリズムは [13] に従っており，転送時間と分割サイズから各チャネルのバンド幅を計算し，バンド幅に比例するように重みを再分配する．また，最適な重みは転送パターン毎に異なるため，重みは転送パターン毎に個別に保持・更新を行う．

この方法では，通信毎に重みが更新され分割サイズが変わるため，ディスクリプタを毎回作成する必要があり，ディスクリプタキャッシュとの相性が悪いと言える．また，それほど大きなオーバヘッドではないが，毎回転送時間を測定するのも望ましくはない．そのため，各転送パターンについて，分割サイズの調整は一定回数のみに制限し，それ以降は最も性能が高かった際の分割サイズを使うようにしている．

6. 性能評価

本節では，DMAC を複数利用することによって，性能がどの程度改善されるのかを評価する．

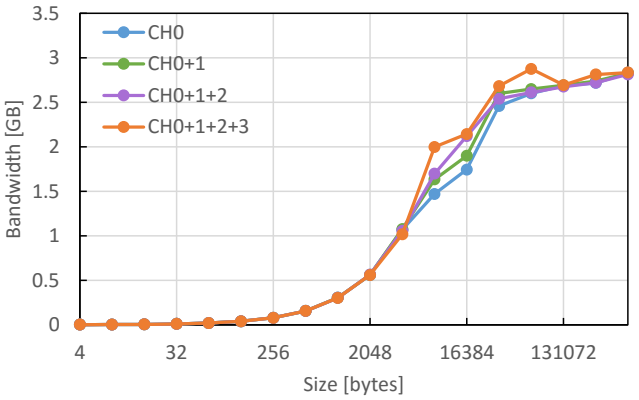

図 5 Ping-Pong 通信のバンド幅

6.1 Ping-Pong 通信の性能

本節では，利用するチャネル数を変えて，GPU to GPU の Ping-Pong 通信を行い，その際の性能について評価する．

図 5 にバンド幅を示す．図より，バンド幅が向上する範囲は 4 KB から 64 KB に限定されているが，基本的に利用するチャネル数が多いほど高いバンド幅が得れていることがわかる．4 チャネル（CH0+1+2+3）利用した場合に特に高いバンド幅が得られており，8 KB でのバンド幅は 1 チャネルしか利用しない場合と比べて 1.4 倍の性能が得られている．

6.2 袖領域通信の性能

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．ストライド転送に関しては，本実装では 4.3 節や 5.1 節より，DMAC を 1 チャネルしか利用しないため，本節での評価項目からは除外している．また，3 次元配列の各要素は倍精度浮動小数点数（8bytes）とする．

図 6 にバンド幅を示す．バンド幅が向上する範囲は $N = 32 \sim 64$ となっており，前節での評価同様に狭い範囲に限定されているが，GASNet/IB や MPI と比べても非常に高い性能が得られていることがわかる．また，$N = 16$ では若干性能が落ちているが，ブロックストライド転送においても 4 チャネル利用することで，1 チャネルしか利用しない場合よりも高い性能が得られていることがわかる．

6.3 姫野ベンチマークの性能

姫野ベンチマーク [15] 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

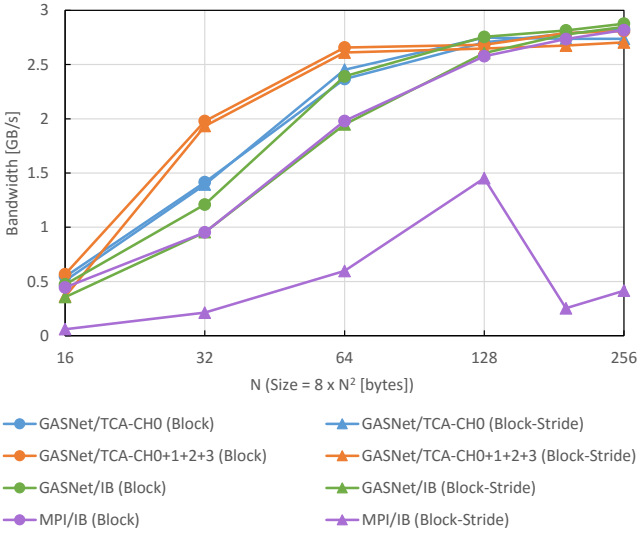

図 6　袖領域通信のバンド幅

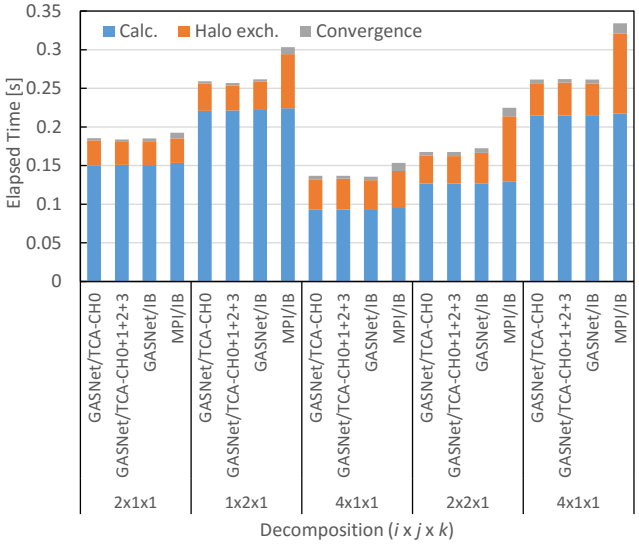

図 7　姫野ベンチマークの測定結果

である．本稿では，前節までの評価で得られた複数 DMAC をによる性能改善の有効範囲から，問題サイズ SMALL（64×64×128）を対象として評価を行う．なお，問題サイズ SMALL での袖領域通信の転送サイズは，2×2×1 の分割の場合は 16 KB，それ以外の分割では 32 KB である．また，評価に関してはヤコビ法の反復回数を 1000 回に固定した際の処理時間によって行う．

結果を図 7 に示す．図中の凡例における “Calc.” は計算にかかった時間，“Halo exch.” は袖領域通信にかかった時間，“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，DMAC を 4 チャネル利用した場合の性能は 1 チャネルしか利用していない場合とほぼ等しく，複数チャネルを利用することによるバンド幅向上の効果がほぼ得られていないことがわかる．袖領域通信に掛かった時間に着目すると，2×1×1 や 1×2×1 の分割では 8 % ～ 9 %程度短縮されており，この結果は最初に行った Ping-Pong 通信の評価における 32 KB での性能向上とほぼ等しく，妥当な結果ではあるが，全体の処理時間を大幅に短縮するほどの短縮ではない．また，4×1×1 や 1×4×1，2×2×1 の分割では，袖領域通信に掛かった時間が全く短縮されていない．特に，2×2×1 の分割に関しては，Ping-Pong 通信の評価における 16 KB での性能向上を考慮すると，袖領域通信に掛かる時間が 20 %程度短縮されるはずである．

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ず，間を空けずに連続して DMA 通信を実行することができないという問題がある．本実装では DMA 通信を連続して行う必要がある場合，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 の停止を検出してから次の DMA 通信通信を開始するようにしているが，オーバヘッドが大きい．2×1×1 や 1×2×1 の分割では，反復毎に 1 ノードを対象とした袖領域通信しか行わず，連続して DMA 通信を行わないためこの問題の影響は受けないが，これ以外の分割の場合では，2 ノードを対象とした袖領域通信を行い，連続して DMA 通信を行う必要があるため，この問題の影響を受けてしまう．特に，本研究では，DMAC を複数チャネル利用しているため，1 チャネルしか使わない場合と比べてオーバヘッド大きくなり，このような結果になったと考えられる．PEACH2 は FPGA による実装を行っているため，今後の改良でこの問題を修正し，性能を改善する余地はあると考えている．

7.　おわりに

本稿では，PEACH2 に実装されている複数の DMAC を活用した性能改善について検討を行い，我々が開発している TCA/PEACH による GPU 対応 GASNet の実装に対して適用した．そして，Ping-Pong 通信や，袖領域通信，姫野ベンチマークによって，性能改善についての評価を行った．

その結果，Ping-Pong 通信や袖領域通信では，性能改善が得られる範囲が広くはないものの，DMAC を 4 チャネル利用することで，1 チャネルしか利用しない場合と比べて最大で 1.4 倍のバンド幅が得られた．しかし，姫野ベンチマークに関しては，ハードウェア不具合への対処によるオーバヘッドが原因で，性能改善は殆ど得られなかった．

現在，64 個の TAG を 4 チャネルの DMAC に不均等に割り当てているため，DMAC 間での性能差が大きくなってしまっている．また，多くのチャネルを利用するというのは，DMAC の起動や完了待ちによるオーバヘッドを考えると効率が良いとは言えない．PEACH2 は FPGA で実装されているため，DMAC への TAG の割り当ても変更可能である．例えば，CH2 と CH3 を無効化し，CH0 と CH1

に TAG を 32 個ずつ割り当てれば，各 DMAC の性能は等しくなり，4 チャネル利用する場合よりも起動コストなどが少なく済むと考えられる．この点に関しては，今後の課題としたい．

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] 佐藤 賢太, 藤田 典久, 塙 敏博, 松本 和也, 朴 泰祐, Khaled Ibrahim: 密結合並列演算加速機構 TCA による GPU 対応 GASNet の実装, 情報処理学会第 153 回 HPC 研究会報告 (HPC153), volume 2016-HPC-153, 2016.
[8] 佐藤 賢太, 藤田 典久, 塙 敏博, 松本 和也, 朴 泰祐, Khaled Ibrahim: 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, ハイパフォーマンスコンピューティングと計算科学シンポジウム論文集 2016, pp.68-76, 2016.
[9] NVIDIA GPUDirect， https://developer.nvidia.com/gpudirect.
[10] HA-PACS プロジェクト， http://www.ccs.tsukuba.ac.jp/research/research_promotion/project/ha-pacs.
[11] PCI Express Base Specification, Rev. 3.0, PCI-SIG, Nov. 2010.
[12] GASNet-EX Collaboration, https://sites.google.com/a/lbl.gov/gasnet-ex-collaboration/
[13] J. Liu, A. Vishnu, D.K. Panda: Building Multirail InfiniBand Clusters: MPI-Level Design and Performance Evaluation, Supercomputing 2004, Nov. 2004.
[14] 畑中正行, 堀敦史, 石川裕: RDMA スケジューリングによる MPI 通信の高速化情報処理学会第 140 回 HPC 研究会報告 (HPC140), volume 2013-HPC-140, 2013.
[15] 姫野ベンチマーク，http://accc.riken.jp/2145.htm.