情報処理学会論文誌　コンピューティングシステム　Vol.8　No.4　36–49 (Nov. 2015)

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

松本 和也[1,a)]　塙 敏博[2]　児玉 祐悦[4]　藤井 久史[5]　朴 泰祐[1,3]

受付日 2015年4月21日, 採録日 2015年8月26日

概要：筑波大学計算科学研究センターでは，GPU クラスタにおけるノード間にまたがる GPU 間通信のレイテンシの改善を目的とした密結合並列演算加速機構 TCA（Tightly Coupled Accelerators）を独自開発している．本稿では，Broadcast，Scatter，Gather，Reduce，Allgather，Allreduce の 6 つの Collective 通信の TCA による実装と，その性能を TCA 実証環境の GPU クラスタである HA-PACS/TCA において評価した結果を述べる．TCA による実装は通信レイテンシが問題となる小さめなサイズの Collective 通信において，MPI による Collective 通信と比べて高速にその通信処理を行うことが可能であることを示す．また，実装した Collective 通信を利用した Conjugate Gradient 法（CG 法）の実装およびその性能について述べる．本研究で用いる CG 法の並列アルゴリズムは，Allgather と Allreduce をその通信部分に用いるものである．TCA による Collective 通信を用いた CG 法実装は，疎行列のサイズが比較的小さい場合（評価した範囲においては行数 36,000 の疎行列の場合まで）では MPI の Collective 通信を用いた実装よりも高い性能を達成できることを示す．

キーワード：Collective 通信，GPU クラスタ，GPU 間直接通信，相互結合網

Implementation and Performance Evaluation of Collective Communication with Proprietary Interconnect TCA for Direct Communication between GPUs

Kazuya Matsumoto[1,a)]　Toshihiro Hanawa[2]　Yuetsu Kodama[4]　Hisafumi Fujii[5]
Taisuke Boku[1,3]

Received: April 21, 2015, Accepted: August 26, 2015

Abstract: We have been developing a proprietary interconnect technology called Tightly Coupled Accelerators (TCA) architecture to improve communication latency between compute nodes on a GPU cluster. This paper presents the implementation and performance evaluation results of six different collective communication operations (broadcast, scatter, gather, reduce, allgather, and allreduce). The performance measurements are conducted on HA-PACS/TCA, which is a proof-of-concept GPU cluster based on the TCA architecture. The implementation using TCA is faster than an MPI collective communication implementation for small sizes where the communication latency decides most of its performance. This paper also describes an implementation of Conjugate Gradient (CG) method utilizing the implemented collective communication and presents the performance. We use the parallel algorithm of CG method that utilizes the allgather and allreduce in the data communication. The CG method implementation using TCA outperforms the implementation using MPI for sparse matrices whose matrix size is relatively small (up to 36,000 rows of matrix in this study).

Keywords: Collective communication, GPU cluster, direct communication between GPUs, interconnect network

1. はじめに

近年，高い演算性能とメモリバンド幅性能を持つ GPU を汎用的な計算に活用する GPGPU（General Purpose GPU）に関する研究がさかんに行われている．GPU は消費電力あたりの演算性能という点においても CPU を上回り，GPU を計算加速機構として搭載した GPU クラスタも高性能計算分野において普及している．しかし，GPU クラスタで並列処理計算を行うためには，複数ノードにまたがる GPU 間データ通信が必要であり，その通信性能は GPU の演算性能と比べて十分に高いとはいえない．計算ノードをまたがる GPU 間の通信はホストを経由して行う必要があるため，その通信レイテンシはホストメモリ間で通信を行う場合よりも大きくなる．また，小さなデータを頻繁に GPU 間でやりとりするようなアプリケーションにおいては，通信のバンド幅性能よりもレイテンシの小ささが重要になることがある．

そこで筑波大学計算科学研究センターでは，ノードをまたぐ通信に関わるレイテンシの改善を目指して密結合型並列演算加速機構 TCA （Tightly Coupled Accelerators）を独自開発している [1], [2]．TCA はインターコネクト・ネットワークに関する技術であり，ノードをまたぐ GPU などのアクセラレータ間の直接通信を可能にする．2013 年 10 月からは，GPU クラスタ HA-PACS [3] の拡張部として，TCA 実証環境である HA-PACS/TCA が稼働している．

TCA については Ping-pong 性能の評価 [2] をはじめとして，いくつかの性能評価が行われている [4], [5], [6], [7]．実際のアプリケーションに TCA を利用した場合の有効性については，姫野ベンチマーク [6]，格子 QCD のライブラリ QUDA [4]，Conjugate Gradient 法（CG 法）[5]，FFT [7] に対してこれまでに行ってきた．これらの性能評価により，TCA はノードをまたぐ GPU 間通信を低レイテンシで実現できることが確かめられ，通信時間が計算時間よりボトルネックとなる場合において，TCA は有効であることが示されている．

Collective 通信（集団通信）とは，複数ノード/プロセスが同時に関わる通信操作のことである．MPI 標準仕様においても各種の Collective 通信が定義されており，MPI を用いて並列計算を行うほとんどのプログラムに Collective 通信は現れる．Collective 通信についての研究は多様で，ネットワークトポロジを考慮した通信アルゴリズムの研究 [8] や通信ライブラリの自動チューニングの研究 [9] なども行われてきた．

本研究では，TCA を用いて各種の Collective 通信を実装しその性能を評価する．TCA による通信は Remote Write による片方向通信を基本としており，1 対 1 通信が基本の MPI により記述された並列プログラムを，単純に TCA を用いる形に移植できるわけではない．本稿においては，6 つの Collective 通信（Broadcast，Scatter，Gather，Reduce，Allgather，Allreduce）の TCA を用いた実装について述べる．そして，TCA を用いた実装と MPI の Collective 通信実装との性能比較を行い，TCA がどのような条件で有効であるのかを示す．

また，本稿では Collective 通信実装を CG 法の実装へ適用した結果についても記す．本研究で用いる CG 法の並列アルゴリズムは，Allgather と Allreduce をその通信部分に用いるものである．この CG 法の性能評価を行うことで，TCA による Collective 通信を使用することの効果について示す．

2. 密結合並列演算加速機構 TCA

2.1 TCA と PEACH2

密結合並列演算加速機構 TCA（Tightly Coupled Accelerators）は，アクセラレータ間（演算加速機構間）の直接結合に関する通信機構技術のことであり，その詳細は文献 [1], [2], [10] に詳しい．本節では，本稿を理解するために必要な TCA の概要情報を説明する．

PEACH2（PCI Express Adaptive Communication Hub version 2）は，TCA を実現するためのインタフェース・チップである [2]．PEACH2 は，PCI-Express（PCIe）をアクセラレータ間通信に利用する．PCIe は，シリアルバス・インタフェースであり GPU ボード，Eathernet ボード，InfiniBand（IB）ボードなどの外部機器をホストコンピュータに結合するために広く使われている．PEACH2 どうしを PCIe 外部ケーブルにより結合することにより，クラスタシステムを構築することができる．

PEACH2 はノードをまたぐ GPU 間の直接データ通信を GPUDirect Support for RDMA（GDR）技術 [11] を利用することで可能にする．現在，GDR 技術は NVIDIA の Kepler アーキテクチャの GPU ファミリにおいて利用できる．GDR を用いることで，PEACH2 や InfiniBand HCA のような通信アダプタは，GPU メモリへの直接読み書きが可能となる．GDR は不必要なシステムメモリへのデータコピーを削減し，CPU のオーバヘッドを小さくし，通

[1] 筑波大学計算科学研究センター
Center for Computational Sciences, University of Tsukuba, Tsukuba, Ibaraki 305–8577, Japan
[2] 東京大学情報基盤センター
Information Technology Center, The University of Tokyo, Kashiwa, Chiba 277–8589, Japan
[3] 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, Tsukuba, Ibaraki 305–8577, Japan
[4] 理化学研究所計算科学研究機構
RIKEN Advanced Institute for Computational Science, Kobe, Hyogo 650–0047, Japan
[5] 富士通ソフトウェアテクノロジーズ
FUJITSU Software Technologies Limited, Yokohama, Kanagawa 222–0033, Japan
[a)] kzmtmt@ccs.tsukuba.ac.jp

信レイテンシを短くする．InfiniBand HCA も GDR による通信をサポートするが，PEACH2 は InfiniBand に比べてさらにレイテンシが小さいという利点がある．それは，PEACH2 を介した通信は PCIe のプロトコルのままで行われるので，InfiniBand を介した通信では必要な PCIe と InfiniBand 間とのプロトコル変換にともなうオーバヘッドを削減できるためである．

PEACH2 ボードは，最大 4 GB/s のバンド幅でデータ通信を行う PCIe Gen2 x8 ポートを 4 つ持つ．1 ポートはホストとの接続に用い，残りの 3 ポートは他のノードの PEACH2 と接続するために用いられる．

PEACH2 は，PIO と DMA の 2 つの通信方式を備えている [2]．PIO 通信は，CPU の store 操作によりリモートノードへのデータ書き込みを行う．通信レイテンシがきわめて小さいため，小さなデータの通信に向いている．なお，PEACH2 は CPU 間の PIO 通信のみを提供する．それに対して DMA 通信機能は，PEACH2 チップに搭載されている DMA コントローラにより実現される．DMA 通信は，データの読み込み元と書き込み先の PCIe アドレスおよび書き込むデータのサイズを記述したディスクリプタに沿って行われる．PEACH2 は Chaining DMA 機能を備えており，複数のディスクリプタをポインタ連結しておけば，先頭のディスクリプタに対する通信開始の命令を送ることで連続した DMA 処理が可能である．DMA 通信のレイテンシは PIO 通信のレイテンシと比べて大きいが，DMA の実測バンド幅は PIO のバンド幅より大きく，大きなデータの通信では DMA を用いる方が高速な通信が可能である．

2.2　HA-PACS/TCA

HA-PACS（Highly Accelerated Parallel Advanced System for Computational Sciences）は，筑波大学計算科学研究センターで開発・運用されている，アクセラレータ技術に基づく大規模 GPU クラスタシステムである．HA-PACS は，2012 年 2 月に運用が開始されたベースクラスタ部と，2013 年 10 月に運用が開始された TCA 部（HA-PACS/TCA）からなる．HA-PACS ベースクラスタはコモディティ製品により構成されているのに対し，HA-PACS/TCA にはコモディティ製品に TCA を通信機構として加えた構成である．HA-PACS/TCA は TCA アーキテクチャの実証環境システムであり，PEACH2 ボードの実験環境でもある．本研究では HA-PACS/TCA のみを用いる．

表 1 に HA-PACS/TCA の構成仕様を記し，HA-PACS/TCA の計算ノードの構成を**図 1** に示す．それぞれの計算ノードは 2 ソケットの Intel Xeon E5-2680v2（IvyBridge-EP）CPU と 4 つの NVIDIA K20X（Kepler GK110）GPU を演算装置として持つ．HA-PACS/TCA は 64 ノードから構成されるが，その 64 ノードは 16 ノードずつ 4 つのサブクラスタに分けられる．16 ノードのサブクラスタは**図 2** に示すように 2 × 8 トーラスのトポロジで PEACH2 により接続されている．また，HA-PACS/TCA の全 64 ノードは，2 ポートの InfiniBand QDR によるフルバイセクションバンド幅の Fat Tree ネットワークによってもつながれている．

なお，HA-PACS/TCA においては，TCA は通信を行うデータ大きくなると InfiniBand に通信性能で逆転され

表 1　HA-PACS/TCA システムの構成仕様
Table 1　System Specification of HA-PACS/TCA.

ノード構成	
マザーボード	SuperMicro X9DRG-QF
CPU	Intel Xeon E5-2680 v2 2.8 GHz × 2 (IvyBridge 10 cores / CPU)
メモリ	DDR3 1866 MHz × 4 ch, 128 GB (=8 × 16 GB)
ピーク性能	224 GFlops / CPU
GPU	NVIDIA Tesla K20X 732 MHz × 4 (Kepler GK110 2688 cores / GPU)
メモリ	GDDR5 6 GB / GPU
ピーク性能	1.31 TFlops / GPU
インターコネクト	InfiniBand: Mellanox Connect-X3 Dual-port QDR TCA: PEACH2 board (Altera Stratix-IV GX 530 FPGA)
システム構成	
ノード数	64
インターコネクト	InfiniBand QDR 108 ports switch × 2 ch
ピーク性能	364 TFlops

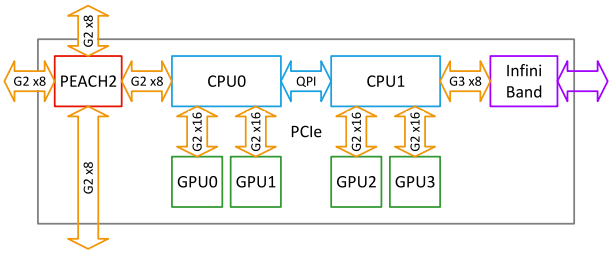


図 1　HA-PACS/TCA のノード構成
Fig. 1　Node configuration of HA-PACS/TCA.

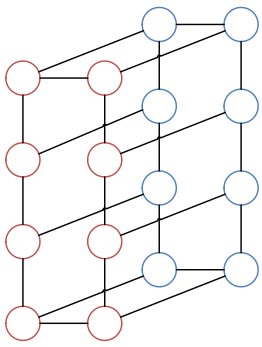


図 2　HA-PACS/TCA におけるサブクラスタの TCA トポロジ
Fig. 2　TCA topology of a sub-cluster on the HA-PACS/TCA.

表 3　プログラム実行時の設定
Table 3　Setting during program executions.

タイプ	環境変数設定
TCA/PEACH2	CUDA_VISIBLE_DEVICES=0,1 MV2_ENABLE_AFFINITY=0 MV2_USE_CUDA=1 numactl –cpunodebind=0 –localalloc
MPI/IB	CUDA_VISIBLE_DEVICES=2,3 MV2_ENABLE_AFFINITY=0 MV2_USE_CUDA=1 MV2_USE_GPUDIRECT=1 MV2_NUM_PORTS=2 MV2_GPUDIRECT_LIMIT=524288 numactl –cpunodebind=1 –localalloc
MPI/IB (GDRCopy 利用時)	CUDA_VISIBLE_DEVICES=2,3 MV2_ENABLE_AFFINITY=0 MV2_USE_CUDA=1 MV2_USE_GPUDIRECT=1 MV2_NUM_PORTS=2 MV2_GPUDIRECT_LIMIT=524288 MV2_USE_GPUDIRECT_GDRCOPY=1 MV2_USE_GPUDIRECT_GDRCOPY_LIMIT=16384 MV2_USE_GPUDIRECT_GDRCOPY_NAIVE_LIMIT=16384 numactl –cpunodebind=1 –localalloc

表 2　性能の測定条件
Table 2　Conditions of performance measurement.

OS	CentOS Linux 6.4 Linux 2.6.32-358.el6.x86_64
GPU プログラミング環境	CUDA 6.5
C コンパイラ	Intel Compiler 14.0.3 (Composer XE 2013 sp1.4.211)
MPI 環境	MVAPICH2 GDR 2.1a

る．TCA の PEACH2 は PCIe Gen2 x8 技術を利用しているので 4 GB/s が理論ピークバンド幅であるが，dual-rail InfiniBand QDR はその倍の理論ピーク性能で通信が可能だからである[*1]．

3. 予備評価

本章では，TCA/PEACH2 による基本的な通信の性能評価結果を記す．性能の測定条件を**表 2** に示す（本研究の性能測定は同様の条件で行う）．

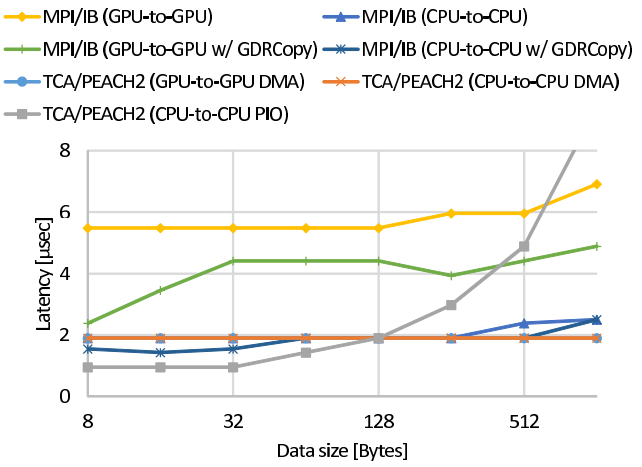
TCA/PEACH2 を用いた実装との比較のために，本稿では InfiniBand 向けの MPI 実装の 1 つである MVAPICH2-GDR 2.1a（以下 MV2GDR）[12] による性能も適宜示す．MV2GDR は，TCA と同様に GPU-Direct for RDMA（GDR）技術 [11] が使われている．GDR により GPU メモリと InfiniBand ボードとの間で直接アクセスが可能となり，InfiniBand を経由した小サイズデータ通信の際のレイテンシが通常の MVAPICH2 と比べて改善されている．また，MV2GDR は GDRCopy [13] ライブラリを利用することでさらに低レイテンシな通信が可能となる．GDRCopy はホスト CPU からの読み書きによって GPU メモリコピーを低レイテンシで行うためのライブラリである（GDRCopy も GDR 技術を使用している）．なお，MV2GDR を用いる際に通信プロトコル・スイッチするデータサイズを調節するためのパラメータをチューニングすることで性能を改善できる．本稿では**表 3** に示すように設定して測定を行った場合の性能を記す．図 1 に示すように計算ノードにおいて PEACH2 ボードは GPU0/GPU1 側に，InfiniBand ボードは GPU2/GPU3 側に搭載されている．本研究においては TCA/PEACH2 による通信の測定時には GPU0 を MPI/IB による通信の測定時には GPU2 を利用するように設定して性能を測定する．

[*1] Dual-rail InfiniBand QDR の理論ピーク性能は 8 GB/s であり，PCIe Gen3 x8 と同等の性能．

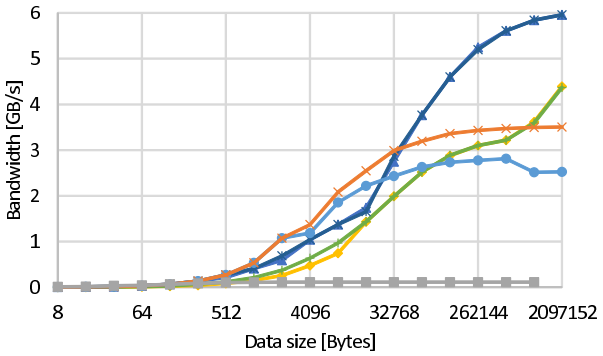
3.1 Ping-pong 通信の性能

図 3 (a) に Ping-pong 通信のレイテンシを示す．TCA/PEACH2 を用いるとき，そのレイテンシは 1.9 μsec である．この小さいデータサイズの測定において TCA による CPU-to-CPU 通信のレイテンシと GPU-to-GPU 通信のレイテンシには顕著な差がない．それに対して MPI/IB の GDRCopy 利用時の GPU-to-GPU 通信のレイテンシは 8 Bytes のとき 2.4 μsec であり 16 Bytes のとき 3.5 μsec であり，TCA の方がレイテンシが小さいことが確認できる．また，MPI/IB の CPU-to-CPU 通信のレイテンシは TCA と同程度である．図 3 (a) の結果から，MPI/IB においては GDRCopy を利用した方が高性能なため以降は GDRCopy を利用した場合のみの性能測定結果を述べる．PIO 通信のレイテンシは 1.0 μsec で最も小さいが，サイズが 128 Bytes より大きくなると DMA 通信より遅い．

図 3 (b) に Ping-pong 通信のバンド幅性能を示す．GPU-to-GPU の Ping-pong 通信の性能において，TCA/PEACH2 は小さいデータサイズの通信では MPI/IB を上回る性能を示している（128 KB 以上のサイズでは逆転される）．TCA/PEACH2 による GPU-to-GPU 通信の最大バンド幅は 2.8 GB/s であり CPU-to-CPU 通信の最大バンド幅は 3.5 GB/s である．GPU-to-GPU 通信の方が性能が低いのは，GPU からの READ 性能が CPU/GPU への WRITE 性能より低いからである．MPI/IB の最大バンド幅は GPU-to-GPU 通信が 4.4 GB/s であり CPU-to-CPU 通信が 6.0 GB/s である．なお，本研究では MPI/IB で GDRDirect 機能を用いる上限サイズを 512 KB と設定（MV2_GPUDIRECT_LIMIT=524288 と設定）しているが，



(a) レイテンシ



(b) バンド幅性能

図 3　Ping-pong 通信の性能比較（100 回測定の最短時間）．レイテンシの図の 8 Bytes から 256 Bytes のデータサイズ範囲においては，TCA/PEACH2 の CPU-to-CPU と GPU-to-GPU および MPI/IB の CPU-to-CPU の通信時間がほぼ重なっている

Fig. 3　Performance of ping-pong communication (shortest time of 100 times measurements). Communication time of TCA/PEACH2 CPU-to-CPU, TCA/PEACH2 GPU-to-GPU, and MPI/IB CPU-to-CPU is almost same.

これは図 4 に示すように本研究の測定環境において最適なサイズである．

3.2　同 GPU メモリ内データコピーの性能

本研究で実装する Collective 通信の中には同 GPU メモリ内でデータコピーを行う必要があるものもある．たとえば，Allgather の場合には，各プロセスが持つ自プロセスの割当て分の送信バッファの初期データを同 GPU メモリ内データコピーにより受信バッファにコピーする必要がある（受信バッファとして送信バッファを指定する場合は本稿では考えない）．このデータコピーを実現する単純な方法としては，同 GPU 間の cudaMemcpy 関数を用いる方法（`cudaMemcpyDeviceToDevice`）がある．しかし，この cudaMemcpy による方法は命令の発行レイテンシが小さくない．そこで，TCA/PEACH2 を用いてこのデータコピーを行う方法（`tcaMemcpyDeviceToDevice`）によりレイテンシの短縮を図る．

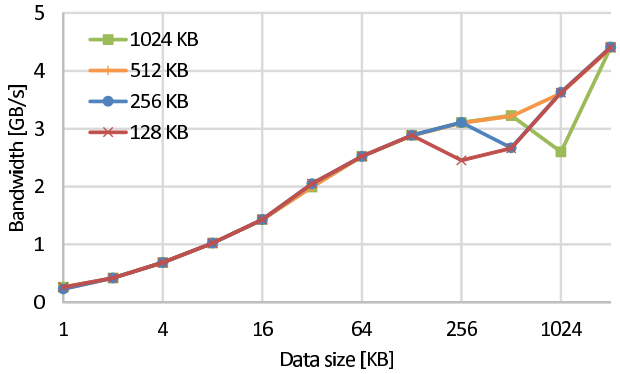


図 4　GDRDirect を用いる上限サイズを変更させたときの MPI/IB による Ping-pong 通信の性能比較（100 回測定の最短時間）

Fig. 4　Performance of MPI/IB ping-pong communication with different GDRDirect upper limit sizes (shortest time of 100 times measurements).

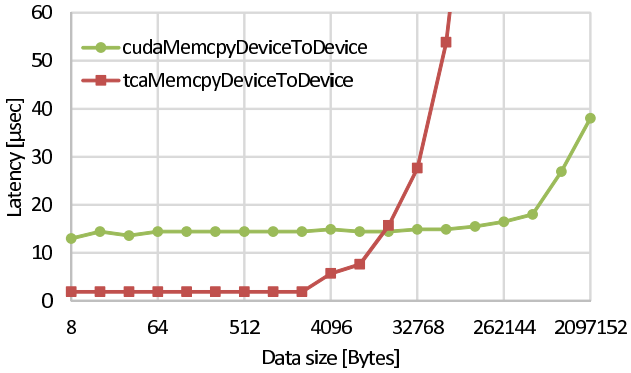


図 5　同 GPU 内データコピーの性能（100 測定の最短時間）

Fig. 5　Performance of data copy within same GPU memory (shortest time of 100 times measurements).

図 5 にその 2 種類の同 GPU 内データコピーの性能を示す．図 5 の結果から分かるように，小さいデータサイズ（8 KB 以下）の場合には TCA による方法の方がレイテンシが小さい．しかし，TCA による方法は，GPU メモリから PEACH2 にデータを読み込んだ後にそのデータを GPU メモリに書き込むためにその最大バンド幅性能は PCIe の性能に律速され，コピーするサイズが大きくなると `cudaMemcpyDeviceToDevice` より遅くなる．この結果から，TCA/PEACH2 による Collective 通信の実装で同 GPU メモリ内のデータコピーが必要な場合においては，小さいサイズでは TCA/PEACH2 による方法で行い，最適なサイズでその方法を cudaMemcpy によるものに切り替えるようにしている．なお，この `cudaMemcpyDeviceToDevice` の性能は，TCA による通信に利用することを想定しコピー対象の GPU のメモリ領域を TCA 用として登録したときのレイテンシである．この登録はレイテンシの増加を招き，登録した際の `cudaMemcpyDeviceToDevice` の最小レイテ

表 4　本稿で述べる Collective 通信（4 プロセス使用時の例）[14]
Table 4　Collective communications described in this paper.

Collective 通信名	通信前				通信後			
Broadcast	Rank 0	Rank 1	Rank 2	Rank 3	Rank 0	Rank 1	Rank 2	Rank 3
	x				x	x	x	x
Scatter	Rank 0	Rank 1	Rank 2	Rank 3	Rank 0	Rank 1	Rank 2	Rank 3
	x_0				x_0			
	x_1					x_1		
	x_2						x_2	
	x_3							x_3
Gather	Rank 0	Rank 1	Rank 2	Rank 3	Rank 0	Rank 1	Rank 2	Rank 3
	x_0				x_0			
		x_1			x_1			
			x_2		x_2			
				x_3	x_3			
Reduce	Rank 0	Rank 1	Rank 2	Rank 3	Rank 0	Rank 1	Rank 2	Rank 3
	$x^{(0)}$	$x^{(1)}$	$x^{(2)}$	$x^{(3)}$	$\sum_j x^{(j)}$			
Allgather	Rank 0	Rank 1	Rank 2	Rank 3	Rank 0	Rank 1	Rank 2	Rank 3
	x_0				x_0	x_0	x_0	x_0
		x_1			x_1	x_1	x_1	x_1
			x_2		x_2	x_2	x_2	x_2
				x_3	x_3	x_3	x_3	x_3
Allreduce	Rank 0	Rank 1	Rank 2	Rank 3	Rank 0	Rank 1	Rank 2	Rank 3
	$x^{(0)}$	$x^{(1)}$	$x^{(2)}$	$x^{(3)}$	$\sum_j x^{(j)}$	$\sum_j x^{(j)}$	$\sum_j x^{(j)}$	$\sum_j x^{(j)}$

ンシは 13.0 μsec であるが登録しない場合の最小レイテンシは 6.0 μsec である．このレイテンシの増加は GDR 技術を利用することの副作用と考えられ，MPI/IB で GDR を利用した際にも同様の現象がみられる．

4. Collective 通信

本章では，TCA による Collective 通信の実装およびその性能評価結果を述べる．性能の測定には HA-PACS/TCA の 1 サブクラスタ（最大 16 ノードまで）を用いる．本研究では，1 ノードあたり 1 GPU のみを用いており，これ以降の記述における利用プロセス数は利用ノード数と一致する．

本稿で述べる Collective 通信は，**表 4** のように通信前と通信後のデータの状態を指定できる [14]．表において，x はサイズ m のデータであり，Collective 通信でデータはプロセス数 p によりサイズ m/p の部分データ x_i $(i = 0, 1, \ldots, p-1)$ に分割される．表 4 の Reduce/Allreduce 通信における $x^{(j)}$ は各プロセスが通信前に各々持つ他プロセスと Reduction されるデータであり，$\sum_j x^{(j)}$ は Reduction 結果を表す（本稿では総和操作の結果を示す）．本研究では，使用するプロセス数 p は 2 のべき乗数（$p = 2, 4, 8,$ or 16）のみに限定する．また，通信に参加する全 p プロセスに対して $0, 1, \ldots, p-1$ と一意にプロセス番号（ランク）を振る．本稿においては，ランク 0 番のプロセスが Root プロセスの場合の実装を説明する．

表 5　Broadcast 実装の通信パターン（8 プロセス使用時）
Table 5　Communication pattern of broadcast implementation when using 8 processes.

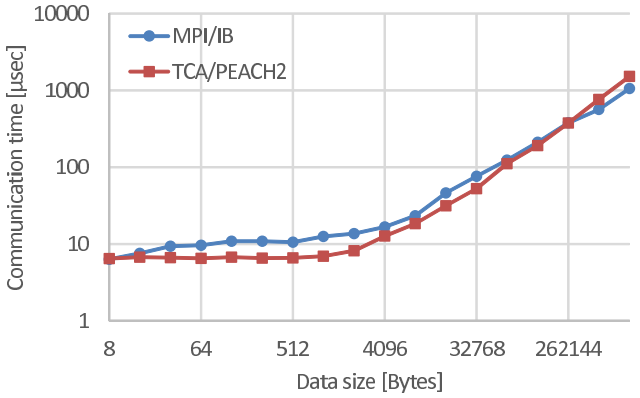
\ Rank	0	1	2	3	4	5	6	7
通信前	x							
Step 1	$x \to 4$							
Step 2	$x \to 2$				$x \to 6$			
Step 3	$x \to 1$		$x \to 3$		$x \to 5$		$x \to 7$	
通信後	x	x	x	x	x	x	x	x

4.1 Broadcast

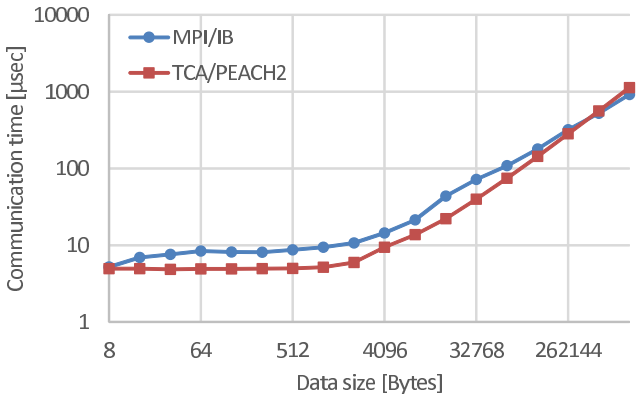
Broadcast（One-to-all broadcast とも呼ばれる）は，特定のプロセス（Root プロセス）のデータを他の全プロセスに送る Collective 通信である．Broadcast 通信により Root プロセスのサイズ m のデータが p プロセスすべてにコピーされる．

Broadcast に関しては，**表 5** のようにデータを送信するプロセス数をステップごとに倍にしていき $\log_2 p$ で通信を完了するアルゴリズム [15] を実装している．実装では TCA/PEACH2 の Chaining DMA 機能を利用し，送信が必要なプロセスにおいて DMA 通信の開始命令を 1 回だけ発行すれば済むようにしている．

図 6 はその Broadcast 実装の性能測定結果である．TCA/PEACH2 による実装は MPI/IB による実装より 256 KB のサイズまでは高い性能を示している．前述の



(a) #Nodes = 16



(b) #Nodes = 8

図 6　Broadcast 通信実装の性能（100 回測定の平均時間）

Fig. 6　Performance of broadcast communication (average time).

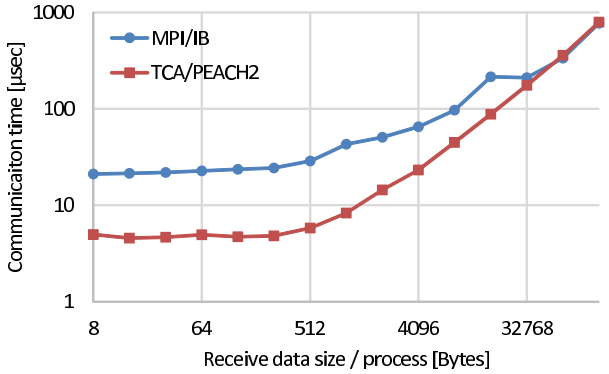


図 7　Scatter 通信実装の性能（16 ノード使用時の 100 回測定の平均時間）

Fig. 7　Performance of scatter communication (average time when using 16 nodes).

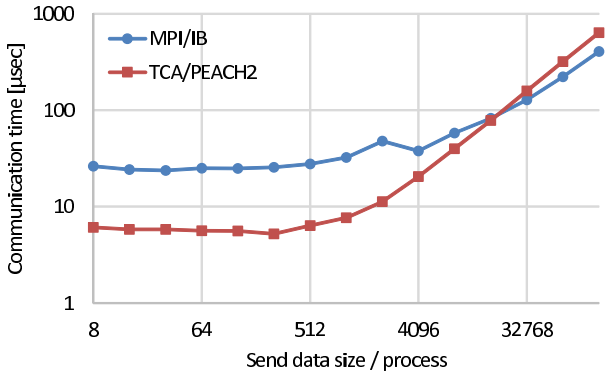


図 8　Gather 通信実装の性能（16 ノード使用時の 100 回測定の平均時間）

Fig. 8　Performance of gather communication (average time when using 16 nodes).

Ping-pong 通信より大きいサイズまで TCA/PEACH2 が高速な理由としては，複数の通信ステップ数が必要な Broadcast ではレイテンシの小さい TCA/PEACH2 が有利となる範囲が大きくなっていると考えられる．

4.2　Scatter

Scatter 操作は，Root プロセスが持つデータを他のプロセスへ送る．この操作は One-to-all personalized communication とも呼ばれる．Scatter 操作では，合計 m サイズのデータを送るが，プロセス・ランク 0 には始めの m/p のサイズのデータを送り，ランク 1 には次のサイズ m/p のデータを送るということを行い，最大ランクのランク $p-1$ には最後のサイズ m/p のデータを送る．

Scatter の実装としては，Root プロセスが各プロセスに対して順番にデータを送信するという単純なものを採用している．Scatter に関しては，Chaining DMA 機能を活用し，Root プロセスが 1 回の DMA 通信の開始命令を発行するだけで済むように実装している．

図 7 に性能を示す．TCA による Scatter 実装は，小さいサイズのときに `MPI_Scatter` 実装の 4 倍以上高速である．この実装の最大通信性能は，TCA の GPU 間通信のバンド幅により制限される．各プロセスごとに送られるデータサイズ（m/b）が 64 KB 以上になると性能が上がらなくなり，性能が 2.7 GB/s で飽和する．その性能が上がらなくなるサイズである 64 KB で `MPI_Scatter` に性能で逆転される．

4.3　Gather

Gather（Concatenation 操作とも呼ばれる）は Scatter と対になる Collective 通信で，Root プロセスへ他のプロセスからデータを集める．Gather に関しても単純な実装を採用しており，それぞれのプロセスが Root プロセスに入力データを送信（WRITE）することで実現している．

図 8 はその実装の性能測定結果である．性能傾向として Scatter と同様であるが，Gather 実装は性能が 2.8 GB/s まで出ており Scatter よりはわずかながら高い性能を示す．この性能差は PEACH2 は WRITE 性能の方が READ 性能より高いことに起因していると考えられる．

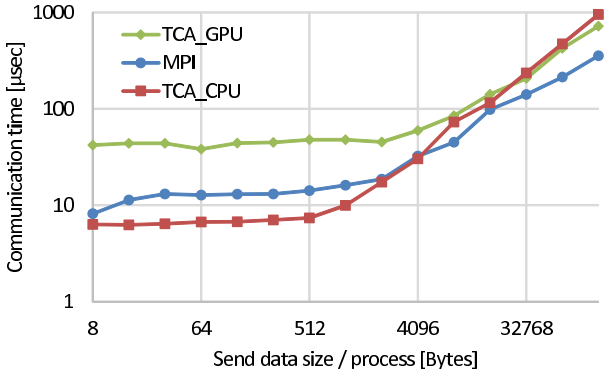


図 9　Reduce 通信実装の性能（16 ノード使用時の 100 回測定の平均時間）

Fig. 9　Performance of reduce communication (average time when using 16 nodes).

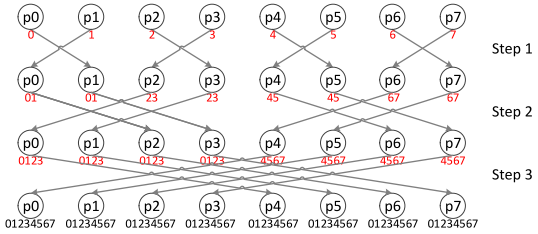
4.4　Reduce

Reduce（All-to-one reduction とも呼ばれる）は，各プロセスにあるデータどうしに加算や乗算などの結合則を満たす演算を施し，その演算結果を Root プロセスに集める Collective 通信である．Reduce 実装の 16 ノード使用時の性能を**図 9** に示す．TCA を用いる Reduce の実装に関しては 2 種類の方法による性能を述べる．図 9 の `TCA_GPU` は，GPU-to-GPU 通信のみを利用し GPU メモリにデータを Gather し，その後に CUDA カーネルを呼ぶことで GPU で Reduction を行う Reduce 実装の性能である．この方法はその CUDA カーネルを発行のためのレイテンシが 14 μsec と大きく，そのレイテンシがデータサイズが小さいときにはボトルネックとなる．

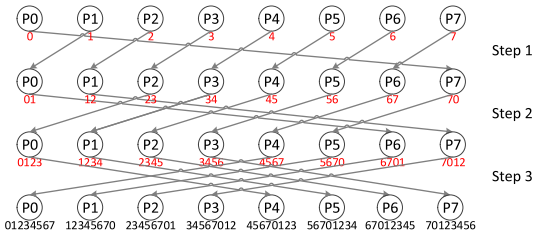
そこで，別の Reduce 実装として始めに TCA の GPU-to-CPU 通信を利用して CPU（ホスト）メモリにデータを Gather し，CPU で Reduction 処理を行い，その Reduction 結果を GPU へコピーする方法も実装した．図 9 の `TCA_CPU` は，その CPU で Reduction を行う実装の性能であり，この実装の方が `TCA_GPU` よりも 8 KB のサイズまでは高い性能を発揮する．倍精度スカラー（8 Bytes）データに対する Reduce の通信レイテンシは，`TCA` が 6.3 μsec，`TCA_CUDA` が 42.0 μsec，そして `MPI` が 8.2 μsec である．

4.5　Allgather

Allgather（All-to-all broadcast とも呼ばれる）は，全プロセスがいっせいに各プロセスへ Gather 操作を行う Collective 通信である．Allgather のアルゴリズムにも様々なものがある [15], [16], [17] が，以前に我々は Allgather アルゴリズムごとの性能の違いを調べた [18]．本稿ではその中で最も高い性能を示した Recursive Doubling 法 [16] による Allgather について述べる．**図 10** (a) に Recursive Doubling 法の通信パターンを示す．このアルゴリズムは自ノードとデータを交換する通信相手ノードのプロセス・ランクの差の絶対値を通信ステップごとに倍にしていく．



(a) Recursive Doubling 法



(b) Dissemination 法

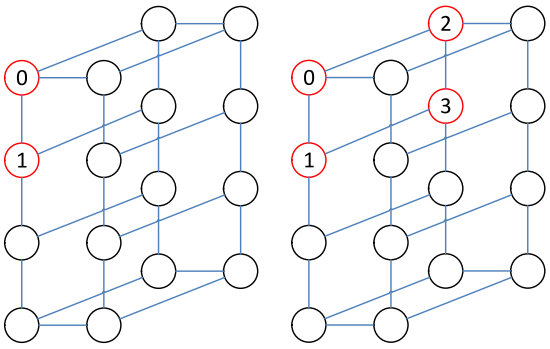
図 10　Allgather/Allreduce のアルゴリズムの通信パターン．図中の赤数字は，その通信ステップで送信するデータを表す

Fig. 10　Communication patterns on two allgather/allreduce algorithms.

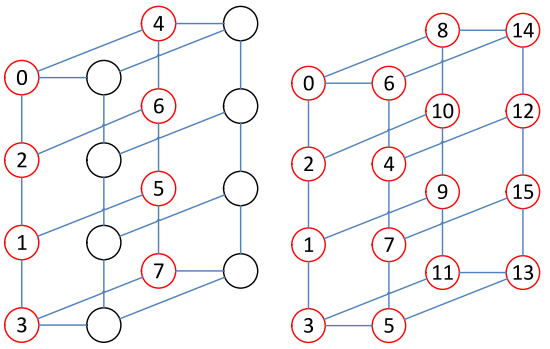
通信ステップ数は $\log_2 p$ で済むが毎回通信相手が異なり，TCA のような mesh/torus 系のトポロジでは通信経路において衝突を起こしてしまう．なお，現在の TCA/PEACH2 には動的に経路衝突を避けるようなルーティング機能は実装されておらず，固定のルーティング方式が採用されており，任意のノード間の通信においては通信ホップ数が最小となる経路を通るようにルーティングされる．Allgather の場合にはデータ通信量もステップごとに倍になる．そこで，通信の衝突の影響を最小化するために最適なノードマッピングを行い，衝突はメッセージサイズの小さい最初のステップでできるだけ起きるようにしている（**図 11** に使用する最適化したノードマッピングを示す）．また，この Allgather 実装に関しては，直前のステップで送られてきたデータを他のプロセスに送る必要があるため Chaining DMA は使えず，ステップ数の分だけ待ち合わせをし通信開始命令を発行しなければならない．

Allgather 実装の 16 ノード使用時と 8 ノード使用時の性能測定結果を**図 12** に示す．16 ノード使用時には TCA/PEACH2 を用いた Allgather 実装は 64 KB までは `MPI_Allgather` より高い性能を示している．使用ノード数が 8，4，2 のときにおいても以下のことがいえる．

- ノード数 8 のとき，TCA の Allgather 実装は 32 KB のサイズまでは `MPI_Allgather` より速い．
- ノード数 4 のとき，TCA の Allgather 実装は 128 KB のサイズまでは `MPI_Allgather` より速い．
- ノード数 2 のとき，TCA の Allgather 実装は 1 MB のサイズまでは `MPI_Allgather` より速い．



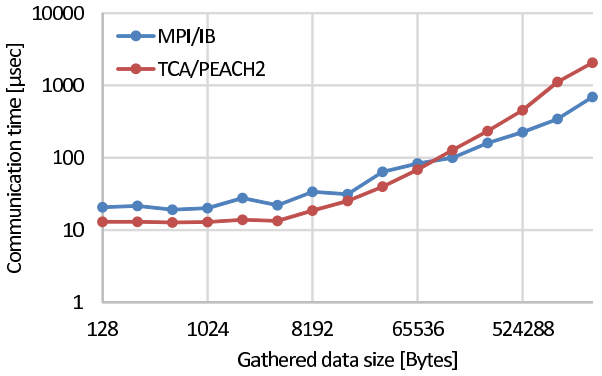
(a) #Nodes = 2　(b) #Nodes = 4



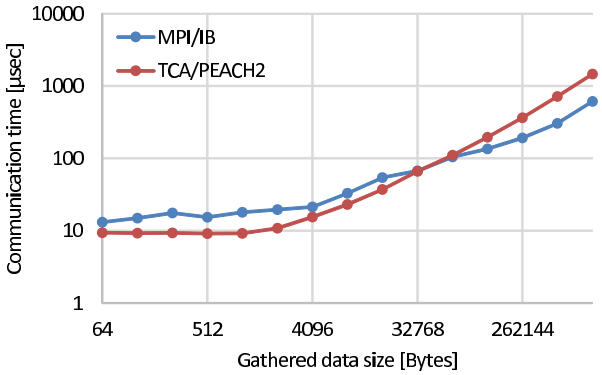
(c) #Nodes = 8　(d) #Nodes = 16

図 11　Recursive Doubling 法による allgather 実装に対して最適化したノードマッピング．丸の中の数字は自身のプロセスランクを表す

Fig. 11　Node mappings optimized for recursive doubling allgather. The number in each circle indicates own process rank.



(a) #Nodes = 16



(b) #Nodes = 8

図 12　Allgather 通信実装の性能（100 回測定の平均時間）

Fig. 12　Performance of reduce communication (average time).

表 6　倍精度スカラー値に対する Allreduce 通信実装の性能（100 回の平均時間で単位は μsec）

Table 6　Performance of allreduce communication for a double-precision value (average time in μsec).

ノード数	2	4	8	16
TCA/PEACH2	1.5	2.6	5.0	6.4
MPI/IB	5.2	8.1	10.4	14.2

4.6　Allreduce

Allreduce（All-to-all reduction とも呼ばれる）は，全プロセスがいっせいに各プロセスへ Reduce 操作を行う Collective 通信である．Allreduce のアルゴリズムとしては Dissemination 法 [17], [19] を用いている[*2]．Dissemination 法の通信パターンを図 10 (b) に示す．この Allreduce のアルゴリズムは，Recursive Doubling 法と同様に $\log_2 p$ 回の通信ステップが必要な方法で，通信先ノードとのプロセス・ランクの差の絶対値を通信ステップごとに倍にしていく方法である．ただし，Dissemination 法はデータを交換するのではなく全プロセスのプロセス・ランクの差が同じになるようにデータを送受信する．

表 6 に倍精度スカラー値（8 Bytes データ）に対する CPU 間の Allreduce 実装の性能を示す．このサイズでは通信のレイテンシにより性能が決まるため TCA/PEACH2 による Allgather 実装では PIO 通信を利用している．表 6 に示した結果からも分かるように，TCA/PEACH2 による実装は `MPI_Allreduce` の半分以下の通信時間で Allreduce を実現する．TCA/PEACH2 の CPU 間 PIO 通信のレイテンシの短さは，Allreduce 通信において有効に働いている．

[*2] Allreduce アルゴリズムごとの性能の違いについても以前に調べ，その中で Dissemination 法が最も良い性能を示している [18].

4.7　今後の改良案

本節では，TCA による GPU 間直接通信に関する今後の改良案について述べる．PEACH2 の DMA 通信のバンド幅性能は，図 3 で示したように GPU 間通信よりも CPU 間通信の方が高い．この CPU 間通信を効果的に利用して GPU 間の通信を実装することで，性能を改善できる可能性がある．それに加えて，Collective 通信に関してはノードマッピングやアルゴリズムのさらなる最適化により実装を改良できると思われる．

ハードウェアによる改良も可能であり，研究が進められている．PEACH2 は FPGA であり，その FPGA 上に演算モジュールを実装することで通常は CPU が行わざる

をえない処理をオフロードすることが可能である．現在，Reduction 演算を PEACH2 の演算モジュールに実装することでレイテンシを改善する試みが行われている [20]．また，PCIe Gen3 技術を利用して TCA を実現する PEACH3 と呼ばれる PEACH2 の改良チップの開発も進められている [21]．

5. CG 法実装への Collective 通信実装の適用

CG 法は，対称正定値行列を係数行列とする連立一次方程式 $Ax = b$ を解くための反復法である．ここで A は $N \times N$ の対称正定値行列であり，x および b は N 次元ベクトルである．本研究では，行列データの格納形式は，Compressed Row Storage（CRS）形式（CSR: Compressed Sparse Row 形式とも呼ばれる）[22] を用いる．浮動小数点演算は倍精度の実数に対して行う．なお，CG 法は前処理を行うことで収束性能を高められる可能性があるが，本研究の実装では前処理は行っていない．

本研究では，CG 法の並列化法として行列 A を単純に一次元分割する手法を用いる．CG 法を並列化するために，疎行列 A を行方向にほぼ均等にプロセス数でデータ分割し，かつベクトル x，b も同割合で均等に分割し，各プロセスに初期データとして持たせる．つまり，プロセス数を p と記述し $n = \lfloor N/p \rfloor$ とするとき，各プロセスは $n \times N$ の A の部分行列および n 次元の b と x の部分ベクトルを持つ（ただし最大ランクのプロセスは $(N-(p-1)n) \times N$ 行列および $(N-(p-1)n)$ 次元ベクトルを持つ）．このようにデータ分割を行うことにより，CG 法の並列アルゴリズムは**図 13** のように記述できる．

CG 法の主な演算は，疎行列ベクトル積計算（SpMV: Sparse Matrix-Vector multiply），内積計算（DOT product），ベクトル加算（AXPY）である．図 13 のアルゴリズムは毎反復において（$k \geq 2$），1 回の SpMV（図 13 の行 15），3 回の DOT（行 6，16，20[*3]），3 回の AXPY（行 12，18，19）を行う．これらの行列とベクトルに対する 3 つの演算は基本的な演算であり，CUDA による NVIDIA 社の数値計算ライブラリでも提供されている．SpMV は cuSPARSE ライブラリ [23] に，DOT と AXPY は cuBLAS ライブラリ [24] にそれぞれ cusparseDcsrmv，cublasDdot，cublasDaxpy ルーチンとして実装されている．本研究では，各 GPU 内の処理ではそれらの cuSPARSE と cuBLAS ルーチンを利用する．

各反復において，図 13 の並列アルゴリズムは，以下のようなデータ通信が必要である．

- SpMV 計算（図 13 の行 15）を行う前に，全プロセスが各プロセスに均等に分散されているベクトルデータ p_l を集める必要がある（Allgather）．

```
 1: x := Allgather(x_l)
 2: r_l := b_l − A_l x
 3: d_t := r_l^T r_l
 4: norm0 := sqrt(AllreduceSum(d_t))
 5: for k := 1, 2, ··· do
 6:    ρ_t := r_l^T r_l
 7:    ρ := AllreduceSum(ρ_t)
 8:    if k = 1 then
 9:       p_l := r_l
10:    else
11:       β := ρ/ρprev
12:       p_l := βp_l + r_l
13:    end if
14:    p := Allgather(p_l)
15:    q_l := A_l p
16:    α_t := ρ/(p_l^T q_l)
17:    α := AllreduceSum(α_t)
18:    x_l := αp_l + x_l
19:    r_l := −αq_l + r_l
20:    d_t := r_l^T r_l
21:    norm := sqrt(AllreduceSum(d_t))
22:    if norm/norm0 < ε then
23:       break
24:    end if
25:    ρprev := ρ
26: end for
```

図 13　CG 法の並列アルゴリズム．各変数の下付き文字 “l” および “t” は，各プロセスごとにローカルに持つ部分データおよび一時データであることをそれぞれ表す

Fig. 13　Parallel algorithm of CG method. The lowercase letters “l” and “t” in each variable indicate local partial data and temporary data which each process has, respectively.

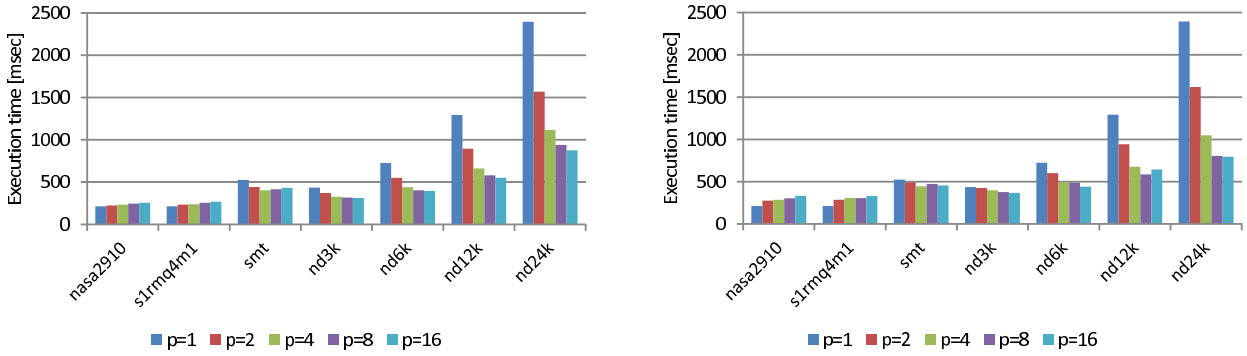
- 各プロセスごとの DOT 計算（行 6，16，20）の後に，そのローカルなベクトル内積の総和を計算し，全プロセスがその総和を持つ必要がある（Allreduce）．

この Collective 通信に TCA/PEACH2 により実装した Allgather と Allreduce を利用する．Allgather は各プロセスが持っているデータブロックを他のプロセスとやりとりし，Allreduce は倍精度の場合は 8 バイトという非常に少量のデータを他プロセスとやりとりする．以上の特徴から，Allgather は TCA の GPU 間 DMA 通信による実装を利用し，Allreduce は TCA の CPU 間 PIO 通信による実装を利用する[*4]．TCA による通信を行うためには，DMA 通信の場合は DMA ディスクリプタを作成する必要があり，PIO 通信の場合は PIO 領域の準備が必要である．一度通信準備をしたら同じ通信を何度でも行えるので，CG 法実装の Allgather と Allreduce 通信においては，初めて通信を行う前に一度だけ通信準備をするようにしている．

4.5 節において示したように，TCA/PEACH2 による通信の性能はノードマッピングの影響を受けるため，CG 法

[*3] ベクトルの 2-ノルムは内積計算を用いて計算する．

[*4] cublasDdot は計算したローカルな内積を CPU 側にも返すことができるので，それにより返されたスカラー値を CPU 間 Allreduce を利用して内積を計算する．



(a) TCA/PEACH2　　(b) MPI/IB

図 14　CG 法実装の実行時間

Fig. 14　Execution time of the CG method implementation.

表 7　性能評価に用いた疎行列の特性

Table 7　Characteristics of sparse matrices used for experiments.

行列名	行数（N）	非零要素数（nnz）	nnz/N
nasa2910	2,910	174,296	59.9
s1rmq4m1	5,489	281,111	51.2
smt	25,710	3,753,184	146.0
nd3k	9,000	3,279,690	364.4
nd6k	18,000	6,897,316	383.2
nd12k	36,000	14,220,946	395.0
nd24k	72,000	28,715,634	398.8

実装においてもどのようなマッピングを用いるかについて考慮する必要がある．Allgather と Allreduce という 2 種類の通信を含む CG 法実装において，それぞれの通信に対して別々の最適なマッピングを用いることが望ましいと考えられるが，本研究では図 11 に示す Allgather に最適化したマッピングを用いている．Allgather 実装に最適化したマッピングを用いる理由としては，CG 法実装においてスカラー値をやりとりするだけの Allreduc 実装はどのようなマッピングを用いても明らかな性能差がみられないが，Allgather 実装では数 KB から数 MB のデータを通信するためにマッピングが性能へ与える影響が大きいためである．

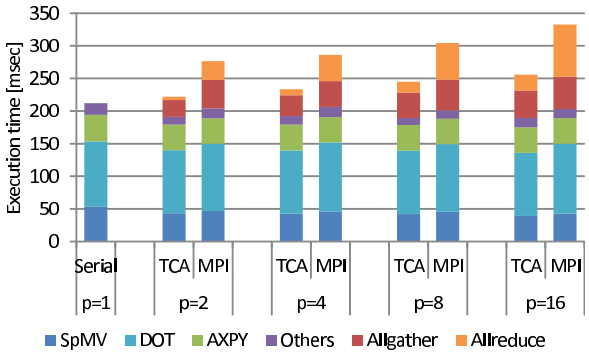
5.1　CG 法実装の性能

性能評価に用いる疎行列は，The University of Florida Sparse Matrix Collection [25] から取得した**表 7** に記す実数の対称正定値行列である．なお，実際の使用において CG 法は，解が収束するまで反復する必要があるが，本研究では性能評価のために反復回数を 1,000 回に固定している[*5]．

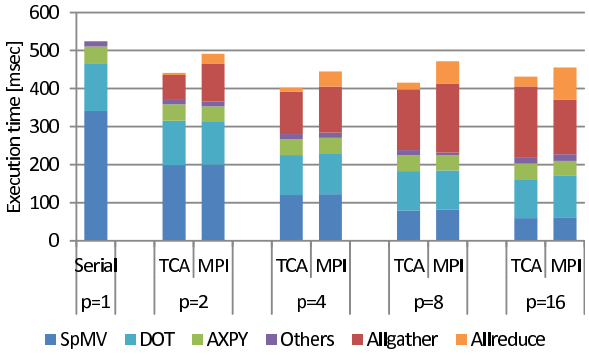
図 14 に，各疎行列に対する CG 法実装の実行時間を示す．この図では，プロセス数を 1，2，4，8，16 と増やしたときの実行時間を示している．TCA/PEACH2 を用いる実装は行列 nd3k，nd6k，nd12，nd24 に関しては，プロセス数を増やすことにより性能向上を達成できている．行列 smt に関しては，4 プロセス用いるときが最も高い性能を示す．残り 2 つの小さめの行列（nasa2910 と s1rmq4m1）に関しては，プロセス数を増やすと性能は悪化する．また，TCA/PEACH2 を用いることにより MPI/IB を用いる場合よりも nd24k 以外の行列に関しては実行時間を短縮することができている．

性能をさらに分析するために，各処理ごとの合計実行時間の内訳を見る．**図 15** に 3 種の異なるサイズの行列（nasa2910，smt，nd24k）に対する各処理の内訳を示す．この性能測定は各処理の前後に `gettimeofday` 関数を呼びシステム時間を取得することで行っている．測定環境では `gettimeofday` 関数の 1 回の呼び出しに平均 0.04 usec かかるが，CG 法実装の合計計算時間に対するこのシステム時間取得に関するオーバヘッドの割合は nasa2910 においても 0.5%以下であり，計測オーバヘッドの影響は多少はあるが大きくはないと思われる．この内訳はプロセス・ランク 0 の結果で，比較のために MPI/IB を用いる実装による結果も併記している．この図の結果からいえることは，nasa2910 のように行数 n（および行あたりの非零要素数）が小さすぎると並列化をしても計算処理（SpMV，DOT，AXPY）の実行時間がほぼ一定で変わらず，データ通信時間の分だけ遅くなってしまうということである．それに対して nd24k のように行列サイズが大きすぎると並列化により性能は向上するが，TCA による Allgather の通信時間が MPI のものより長くなってしまい，結果として TCA による実装の方が MPI による実装より性能が劣るものになってしまう．それらの中間の行列サイズの行列（smt を含む 18,000 行から 36,000 行のサイズの行列）に対しては，TCA を用いることで MPI を用いるよりも高い性能を実現できている．

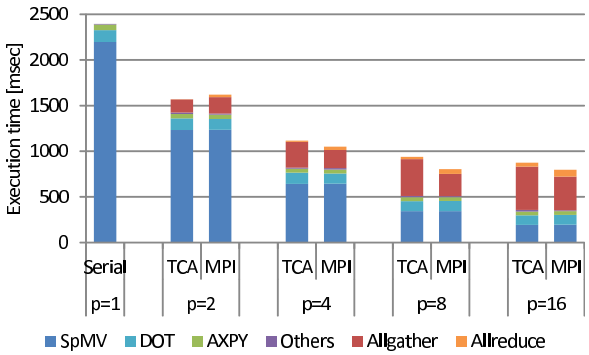
[*5] 本研究は CG 法における 1 反復あたりの処理時間の評価を目的としており，収束するか否かは問題としない．性能評価における反復あたりのバラつきによる誤差をなくすための十分な反復回数として 1,000 回を選んだ．



(a) nasa2910



(b) smt



(c) nd24k

図 15　CG 法実装の各処理実行時間の内訳（ランク 0 の内訳）
Fig. 15　Time breakdown on CG method implementation (breakdown of rank 0).

6. おわりに

本研究では，TCA による Collective 通信の実装を行い，HA-PACS/TCA GPU クラスタにおいてその実装の性能評価を行った．本稿では Broadcast，Scatter，Gather，Reduce，Allgather，Allreduce 通信の実装とその性能を述べた．TCA/PEACH2 によりノードをまたぐ通信が低レイテンシで実現され，それにより小さいサイズの Collective 通信については MPI/IB による Collective 通信と比べて高速にその通信処理を行うことが可能になる．

また，実装した Collective 通信を利用した CG 法の実装を行い，その性能について評価を行った．CG 法の並列アルゴリズムとしては，SpMV 計算に必要なデータを Allgather で集め，ベクトル内積を Allreduce を利用して実現するものを用いている．TCA を用いた実装は疎行列のサイズ（行数）が 36,000 までの場合においては MPI を用いた実装よりも高い性能を示した．

TCA はアクセラレータ間の直接通信機構が必要であるという考えのもとに開発されている．本研究の結果から TCA/PEACH2 によりノードをまたぐ GPU 間の Collective 通信を低レイテンシで実現できることが示せた．しかし，GPU を利用する際に必要な通信以外の処理のレイテンシ（CUDA カーネル発行のためのレイテンシなど）により，TCA を利用したとしても必ずしも性能を改善できるというわけではないことが分かった．

どのような通信と計算を含んだアプリケーションが TCA を用いるのに適しているのかを明らかにしていくことが今後の課題である．TCA 利用による効果を予測できるようにするために性能予測モデルの構築を進めている．

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，HA-PACS/TCA システムの利用に関しては筑波大学計算科学研究センターの学際共同利用プログラム研究課題「密結合演算加速機構アーキテクチャに向けた GPGPU アプリケーション」による．

参考文献

[1] 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，Vol.6, No.3, pp.14–25 (2013).

[2] Hanawa, T., Kodama, Y., Boku, T. and Sato, M.: Tightly Coupled Accelerators Architecture for Minimizing Communication Latency among Accelerators, *Proc. 27th IEEE International Parallel and Distributed Processing Symposium Workshops & PhD Forum* (*IPDPSW 2013*), pp.1030–1039, IEEE (2013).

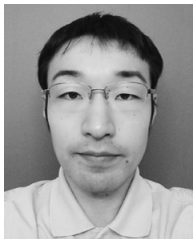
[3] 朴　泰祐，佐藤三久，塙　敏博，児玉祐悦，高橋大介，建部修見，多田野寛人，蔵増嘉伸，吉川耕司，庄司光男：演算加速装置に基づく超並列クラスタ HA-PACS による大規模計算科学，情報処理学会研究報告，Vol.2011-HPC-130, No.21, pp.1–7 (2011).

[4] 藤井久史，藤田典久，塙　敏博，児玉祐悦，朴　泰祐，佐藤三久，藏増嘉伸，Clark, M.：GPU 向け QCD ライブラリ QUDA の TCA アーキテクチャ実装の性能評価，情報処理学会研究報告，Vol.2014, No.43, pp.1–9 (2014).

[5] Matsumoto, K., Hanawa, T., Kodama, Y., Fujii, H. and Boku, T.: Implementation of CG Method on GPU Cluster with Proprietary Interconnect TCA for GPU Direct Communication, *Proc. 29th IEEE International Parallel and Distributed Processing Symposium Workshops* (*IPDPSW 2015*), pp.647–655, IEEE (2015).

[6] 藤井久史，塙　敏博，児玉祐悦，朴　泰祐，佐藤三久：TCA アーキテクチャによる並列 GPU アプリケーションの性

能評価，情報処理学会研究報告，Vol.HPC-140, No.37, pp.1–6 (2013).
[7] 藤井久史，塙　敏博，児玉祐悦，朴　泰祐：GPU 向け FFT コードの TCA アーキテクチャによる実装と性能評価，情報処理学会研究報告，Vol.HPC-148, No.12, pp.1–9 (2015).
[8] Sack, P. and Gropp, W.: Faster topology-aware collective algorithms through non-minimal communication, *ACM SIGPLAN Notices*, Vol.47, No.8, pp.45–55 (2012).
[9] Barnett, M., Shuler, L., van de Geijn, R., Gupta, S., Payne, D.G. and Watts, J.: Interprocessor collective communication library (InterCom), *Proc. IEEE Scalable High Performance Computing Conference 1994*, IEEE Computer Society, pp.357–364 (1994).
[10] Kodama, Y., Hanawa, T., Boku, T. and Sato, M.: PEACH2: An FPGA-based PCIe network device for Tightly Coupled Accelerators, *ACM SIGARCH Computer Architecture News*, Vol.42, No.4, pp.3–8 (2014).
[11] NVIDIA: NVIDIA GPUDirect, (online), available from ⟨https://developer.nvidia.com/gpudirect⟩ (accessed 2015-06-15).
[12] Panda, D.K.: MVAPICH2-GDR (MVAPICH2 with GPUDirect RDMA), The Ohio State University (online), available from ⟨http://mvapich.cse.ohio-state.edu/overview/⟩ (accessed 2015-06-15).
[13] NVIDIA Corporation: GDRCopy, GitHub (online), available from ⟨https://github.com/NVIDIA/gdrcopy⟩ (accessed 2015-06-14).
[14] Chan, E., Heimlich, M., Purkayastha, A. and van De Geijn, R.: Collective communication: theory, practice, and experience, *Concurrency and Computation: Practice and Experience*, Vol.19, No.13, pp.1749–1783 (2007).
[15] Grama, A., Karypis, G., Kumar, V. and Gupta, A.: *Introduction to Parallel Computing, 2nd edition*, Addison-Wesley (2003).
[16] Kogge, P.M. and Stone, H.S.: A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations, *IEEE Trans. Comput.*, Vol.C-22, No.8, pp.786–793 (1973).
[17] Bruck, J. and Ho, C.-T.: Efficient Algorithms for All-to-All Communications in Multiport Message-Passing Systems, *IEEE Trans. Parallel and Distributed Systems*, Vol.8, No.11, pp.1143–1156 (1997).
[18] 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 を用いた GPU 間直接通信による CG 法の実装と予備評価，情報処理学会研究報告，Vol.HPC-144, No.12, pp.1–9, 情報処理学会 (2014).
[19] Hensgen, D., Finkel, R. and Manber, U.: Two Algorithms for Barrier Synchronization, *International Journal of Parallel Programming*, Vol.17, No.1, pp.1–17 (1988).
[20] 久原拓也，宮島敬明，塙　敏博，天野英晴：PEACH2 への演算機構の実装とその性能評価，電子情報通信学会技術研究報告，Vol.114, No.223, RECONF2014-28, pp.63–68 (2014).
[21] Kuhara, T., Kaneda, T., Hanawa, T., Kodama, Y., Boku, T. and Amano, H.: A Preliminarily Evaluation of PEACH3: A Switching Hub for Tightly Coupled Accelerators, *Proc. 2nd International Symposium on Computing and Networking* (*CANDAR 2014*), pp.377–381 (2014).
[22] Saad, Y.: *Iterative Methods for Sparse Linear Systems, 2nd edition*, SIAM (2003).
[23] NVIDIA: cuSPARSE Library, (online), available from ⟨http://docs.nvidia.com/cuda/cusparse/index.html⟩ (accessed 2015-06-14).
[24] NVIDIA: cuBLAS Library, (online), available from ⟨http://docs.nvidia.com/cuda/cublas/index.html⟩ (accessed 2015-06-14).
[25] Davis, T.A. and Hu, Y.: The University of Florida Sparse Matrix Collection, *ACM Trans. Mathematical Software*, Vol.38, No.1, pp.1:1–1:25 (online), available from ⟨http://www.cise.ufl.edu/research/sparse/matrices⟩ (2011).



松本 和也　（正会員）

2008 年会津大学コンピュータ理工学部コンピュータソフトウェア学科卒業．2013 年会津大学大学院コンピュータ理工学研究科コンピュータ・情報システム学専攻博士後期課程修了．博士（コンピュータ理工学）．同年会津大学コンピュータ理工学部研究員を経て，筑波大学計算科学研究センター研究員．ハイパフォーマンスコンピューティングに関する研究に従事．高性能数値計算，高性能通信ライブラリ等に興味あり．IEEE CS 会員．

塙 敏博　（正会員）

1998 年慶應義塾大学大学院理工学研究科計算機科学専攻博士課程修了．博士（工学）．同年東京工科大学工学部情報工学科講師，2003 年東京工科大学コンピュータサイエンス学部講師，2007 年筑波大学計算科学研究センター研究員，2008 年筑波大学システム情報工学研究科准教授を経て，2013 年より東京大学情報基盤センター特任准教授．計算機アーキテクチャ，高性能相互結合網，ハイパフォーマンスコンピューティング，GPU コンピューティング等に関する研究に従事．2013 年度山下記念研究賞，1995 年度情報処理学会論文賞．IEEE CS 会員．

児玉 祐悦　（正会員）

1988 年東京大学大学院情報工学専門課程修士課程修了．同年通商産業省電子技術総合研究所入所．2001 年独立行政法人産業技術総合研究所に改組．2011 年 2 月筑波大学大学院システム情報工学研究科・計算科学研究センター教授．2015 年 4 月より国立研究開発法人理化学研究所計算科学研究機構上級研究員．データ駆動やマルチスレッド等の並列計算機システムの研究に従事．特にプロセッサアーキテクチャ，並列性制御，FPGA 応用，ネットワーク制御，アクセラレータ等に興味あり．博士（工学）．情報処理学会論文賞（1990 年度），市村学術賞（1995 年）等受賞．電子情報通信学会，IEEE CS 各会員．

藤井 久史

2013 年筑波大学情報学群情報科学類卒業．2015 年筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻修了．高性能計算，相互結合網等に興味あり．

朴 泰祐　（正会員）

1960 年生．1984 年慶應義塾大学工学部電気工学科卒業．1990 年慶應義塾大学大学院理工学研究科電気工学専攻後期博士課程修了．工学博士．1988 年慶應義塾大学理工学部物理学科助手．1992 年筑波大学電子・情報工学系講師，1995 年同助教授，2004 年同大学大学院システム情報工学研究科助教授，2005 年同教授，現在に至る．超並列計算機アーキテクチャ，ハイパフォーマンスコンピューティング，クラスタコンピューティング，GPU コンピューティングに関する研究に従事．筑波大学計算科学研究センターにおいて，超並列計算機 CP-PACS，PACS-CS，HA-PACS 等の研究開発を行う．2002 年および 2003 年情報処理学会論文賞，2011 年 ACM ゴードンベル賞，2012 年情報処理学会山下記念研究賞各受賞．IEEE CS，ACM 各会員．