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　61–77 (Nov. 2015)

# アクセラレータ向け並列言語XcalableACCにおけるTCA/InfiniBandハイブリッド通信

小田嶋 哲哉[1,a)]　朴 泰祐[1,2]　塙 敏博[3]　児玉 祐悦[4]
村井 均[4]　中尾 昌広[4]　田渕 晶大[1]　佐藤 三久[1,4]

受付日 2015年4月21日, 採録日 2015年7月21日

**概要**：近年，HPC の分野では GPU を搭載したクラスタが広く利用されている．しかし，ノードをまたぐ GPU 間通信は，1 度ホストメモリを経由して行うためレイテンシが増加し，アプリケーションの性能ボトルネックとなっている．筑波大学計算科学研究センターでは，ノード間および GPU 間を直接結合し，レイテンシ・バンド幅の改善を目的に密結合並列演算加速機構 TCA（Tightly Coupled Accelerators）を開発している．TCA は，ハードウェアの制限や実装効率の面で数十ノードを一組とするサブクラスタにとどまるが，今後，より大きな問題やサイズに対応するためにサブクラスタをまたいだ通信が必要である．我々は，TCA と InfiniBand によるハイブリッド通信を提案している．しかし，ハイブリッド通信は，TCA と MPI の通信を混在させてプログラムを記述する必要があり，プログラムが煩雑になりがちである．そこで，アクセラレータ向けの並列言語である XcalableACC のフレームワークの中にハイブリッド通信を組み込むことで，低いプログラミングコストで通信ネットワークを有効に利用することを目指す．本論文では，ラプラス方程式および姫野ベンチマークの袖領域交換に適用し，評価を行った結果，InfiniBand 通信に対して最大 40%の性能向上を達成した．これによって，ハイブリッド通信は TCA の適用範囲を拡大し，さらに，一般的な InfiniBand による通信に対して優位性があることが分かった．

**キーワード**：GPU クラスタ，アクセラレータ，PGAS プログラミング，相互結合網，Tightly Coupled Accelerators

# Hybrid Communication with TCA and InfiniBand on a Parallel Programming Language for Accelerators XcalableACC

Tetsuya Odajima[1,a)]　Taisuke Boku[1,2]　Toshihiro Hanawa[3]　Yuetsu Kodama[4]
Hitoshi Murai[4]　Masahiro Nakao[4]　Akihiro Tabuchi[1]　Mitsuhisa Sato[1,4]

Received: April 21, 2015, Accepted: July 21, 2015

***Abstract:*** Recently, GPU equipped clusters are widely used in HPC applications. However, inter-node communication among GPUs might easily be a performance bottleneck because it relies on a host-to-host message passing. We developed a proprietary interconnection network named TCA (Tightly Coupled Accelerators) architecture to improve inter-node communication among GPUs to solve this problem. Current implementation of TCA architecture, named PEACH2, introduces PCI Express (PCIe) external communication link to connect a number of PCIe. Since the number of GPUs which can be directly connected by PEACH2 network is limited to several tens of nodes, we need to combine it with a conventional interconnect such as InfiniBand for scalable systems. In this paper, we proposed a TCA and InfiniBand hybrid communication. For the user convenience on programming of large-scale parallel CPU computing, more sophisticated programming scheme rather than combining multiple communication paradigms is needed. We apply this hybrid communication framework to a parallel programming language for accelerators, XcalableACC, to utilize TCA system effectively to enhance the communication performance minimizing user effort. We implement the TCA and InfiniBand hybrid communication system embedded into the communication layer of XcalableACC compiler and, we achieved up to 40% of performance improvement compared with the InfiniBand only solution. This hybrid communication expands the scope of TCA as well as keeping the programming framework of XcalableACC.

***Keywords:*** GPU cluster, accelerator, PGAS language, interconnection networks, Tightly Coupled Accelerators

## 1. 序論

近年，GPU（Graphics Processing Unit）の持つ高い演算性能とメモリバンド幅に注目し，これを画像処理以外の汎用計算に用いる GPGPU（General-Purpose computation on GPU）が広く利用されている．TOP500 リストの上位には，GPU を搭載したいわゆる GPU クラスタが数多く出現するようになった [1]．しかし，その高い演算性能とメモリバンド幅に比べ，GPU を接続する PCI Express（以降「PCIe」と略す）の通信性能は非常に低く，特に GPU 間でのデータの交換を行う際に大きなボトルネックになる．これに加え，従来，ノード間をまたぐ GPU 間のデータの交換には，ホストメモリを経由して行う必要があり，特にメッセージサイズが小さいときにはレイテンシが大きな問題となり，性能低下の原因となっている．ステンシル計算の典型的な通信パターンとして，隣接ノード間での袖領域交換がある．このような GPU 間のデータ交換が頻繁に必要なアプリケーションで強スケーリングを求める場合，並列度を上げると通信データサイズが小さくなり，バンド幅よりもレイテンシがより性能に影響してくる．文献 [2] にもあるように，今後はアプリケーションの強スケーリング性能を向上させることが重要になる．

そこで，我々はノード間にまたがる GPU 間を直接結合し，レイテンシの改善を図るために密結合並列演算加速機構 TCA（Tightly Coupled Accelerators）を提案し，そのプロトタイプ実装として PCIe に基づく PEACH2（PCI Express Adaptive Communication Hub version 2）システムを開発している．TCA を用いたアプリケーションでは，低レイテンシ通信により性能が向上している [3], [4], [5], [6].

また，現状では，PEACH2 の適用範囲は PCIe などのハードウェア面の制約によりサブクラスタと呼ばれる数十ノードまでの直接結合にとどまるが，より大きな問題やサイズに対応するために，サブクラスタをまたぐ通信を検討する必要がある．そこで，本研究では，TCA と InfiniBand によるハイブリッド通信（以降これを単に「ハイブリッド通信」と呼ぶ）を実現し，より高いシステムスケーラビリティを得ることを目的とする．ハイブリッド通信は，単に通信路が増えるだけではなく，PEACH2+TCA の低レイテンシ通信，Block Stride 通信と，InfiniBand+MPI の高バンド幅通信を活かし，それぞれの利点が活かせるように通信方法を選択することで，PEACH2+TCA と InfiniBand+MPI だけでは得られない性能向上を期待する．

一方，ハイブリッド通信では，TCA 通信を制御する独自の API に加え，MPI の通信を混在させてプログラムを記述することや，通信パターンによって TCA と InfiniBand の最適な通信を設定する必要があり，プログラムが煩雑になりがちである．そこで，大規模分散メモリ環境における次世代の並列プログラミング言語として，理研 AICS が中心となって PGAS（Partitioned Global Address Space）並列言語 XcalableMP（以降「XMP」と略す）の開発が進められている [7], [8]．このアクセラレータ搭載のクラスタ向けの拡張として，XMP と OpenACC [9] を組み合わせた XcalableACC（以降「XACC」と略す）が提案されている [10], [11]. XACC のフレームワークの中にハイブリッド通信を組み込むことで，低いプログラミングコストで通信ネットワークを有効に利用することが期待される．

以上の背景に基づき，本論文では TCA+InfiniBand のハイブリッド通信システムを開発し，これを XACC 言語処理系の通信レイヤに組み込むことで，同言語のプログラム上でユーザ透過なハイブリッド通信の利用を実現する．本論文では，2 次元のラプラス方程式，3 次元の姫野ベンチマークに対して，XACC によるプログラミングを行い，袖領域交換にハイブリッド通信を用い，その有効性と性能について評価を行う．

## 2. TCA アーキテクチャと PEACH2

TCA アーキテクチャおよび PEACH2 は文献 [3], [12], [13], [14] に詳しいが，本章ではその概要を説明する．

### 2.1 TCA

密結合並列演算加速機構 TCA（Tightly Coupled Accelerators）は，ノード間のアクセラレータ（GPU）間を直接結合することで，アクセラレータ間通信のレイテンシを改善することを目的にしたコンセプトで，筑波大学計算科学研究センターが中心となって開発が進められている．現在の TCA では，PCIe をノード間通信に拡張することによって実現しており，PC クラスタにおける GPU，Intel MIC（Many Integrated Core processor）あるいは FPGA などは PCIe によってホスト CPU および並列ネットワークと接続されているため，この構成であらゆるアクセラレータを対象にできる．

これを実現する実装として，我々は PEACH2（PCI Express Adaptive Communication Hub version 2）の開発を行っている．この PEACH2 ボードどうしを PCIe 外部ケーブルで接続し，TCA システムを構成する．さらに TCA コンセプトの実証実験クラスタとして，筑波大学計

[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, Tsukuba, Ibaraki 305–8577, Japan
[2] 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
[3] 東京大学情報基盤センター
Information Technology Center, The University of Tokyo, Bunkyo, Tokyo 113–8658, 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
[a] odajima@hpcs.cs.tsukuba.ac.jp

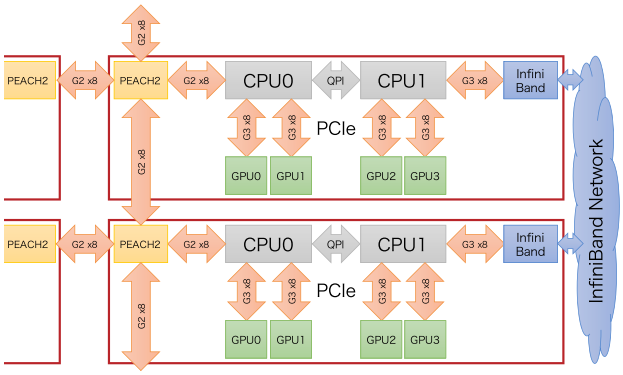


図 1　HA-PACS/TCA のノード構成（文献 [3] より引用）
**Fig. 1**　Node configuration of HA-PACS/TCA (Quote from Ref. [3]).

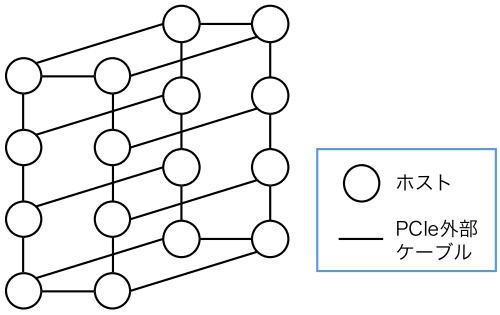


図 2　サブクラスタ：TCA ネットワーク構成
**Fig. 2**　Sub-cluster: TCA network configuration.

算科学研究センターの GPU クラスタ HA-PACS（Highly Accelerated Parallel Advanced system for Computational Sciences）[15] の拡張部として HA-PACS/TCA を構築し，運用している．

**図 1** に，HA-PACS/TCA のノード構成を示す．HA-PACS/TCA のノードは，2 ソケットの Intel Xeon E5-2680v2 CPU と 4 枚の NVIDIA K20X（Kepler アーキテクチャ）GPU を搭載し，CPU0 側には PEACH2 ボードが，CPU1 側には InfiniBand HCA（QDR x2 rails）が接続されている．図中，GPU には CPU が直接接続されているように見えるが，実際には CPU に内蔵されている PCIe スイッチを介して PEACH2 または InfiniBand HCA に接続されているため，実際の通信は CPU を介さず行われる．InfiniBand は HA-PACS/TCA のすべてのノードを単一スイッチでフラットに接続している．一方，TCA のみで大規模なクラスタを構成することは，外部接続ケーブル長の限界や，PCIe パケットのホップ数の増加にともなう性能面での制約により困難であるため，**図 2** のように 16 ノードを TCA で結合している．この集団を「サブクラスタ」と呼び，HA-PACS/TCA では，64 ノードが 4 つのサブクラスタに分かれている．しかし，同時に，64 ノードすべてが InfiniBand によっても結合されているため，あるノードから見ると，TCA で直接通信可能なノードは 15 台あり，それらを含めたすべてのノードとは InfiniBand 経由で通信ができる．

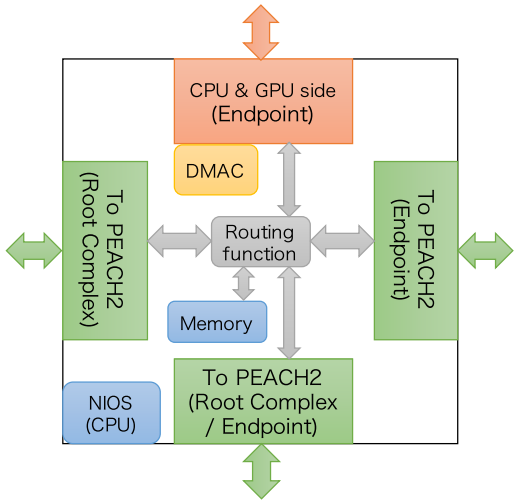


図 3　PEACH2 チップの構成（文献 [3] より引用）
**Fig. 3**　Configuration of PEACH2 chip (Quote from Ref. [3]).

### 2.2　PEACH2 チップ

各ノードの PEACH2 ボードには，我々が開発した PEACH2 チップが搭載されている．PEACH2 チップは，PCIe パケットの中継処理や DMA 転送などを行い，FPGA（Altera 社 Stratix IV GX [16]）で実装されている．**図 3** に PEACH2 チップの構成を示す．このチップは，4 つの PCIe Gen2 ×8 ポートを持ち，1 つはホスト CPU（CPU & GPU side）と接続し，残り 3 つのポート（To PEACH2）を隣接ノードの PEACH2 ボードとの接続に使用する．このように，PEACH2 はハードウェアの制約により，1 つのノードから外部へ延びるリンクが 3 つに限られる．一方で，PEACH2 のみで多くのノードを接続すると，ホップ数が増加し，低レイテンシ通信が有効に使えない．そこで我々は 16 ノードまでの PEACH2 による直接結合を考え，その中で最小のホップ数を実現できる 2 × 8 の 2 重リングトポロジとしてサブクラスタを構成する．PEACH2 には高度な DMA コントローラ（以降「DMAC」と略す）が 4 チャネル搭載されており，高速な DMA や Chained DMA などが可能である．パケットのバッファとして，FPGA 内蔵のメモリと PEACH2 ボード上の DDR3 SDRAM を用いる．

### 2.3　DMA 通信

PEACH2 では，PIO と DMA の 2 つの通信方式があるが，ここでは GPU 間通信の典型的手法である DMA 通信のみについて言及する．

PEACH2 において，リモートノードに対するアクセスは，基本的に RDMA Put プロトコルのみをサポートする．ホスト上であらかじめ読み込み元，書き込み先の PCIe アドレス，サイズを指定したディスクリプタを作成し，これらをアドレスポインタで連結しておく．DMA 通信を始めるときは，ディスクリプタの先頭のアドレスを指定するこ

とで，連続して DMA 処理を行うことが可能である．また，各 DMA のディスクリプタでは連続領域に対するデータ転送だけでなく，バースト長およびギャップを指定することで，Block Stride 転送も行うことができる．よって，あらかじめデータ通信パターン（通信相手と通信領域）が定められていれば，Chained DMA 機構により柔軟かつ高速な通信が実現できる．Block Stride 転送はステンシル計算などで必要となる隣接ノードとの袖領域交換において頻繁に用いられるが，従来の MPI では，データを Pack/Unpack して送る必要がある．PEACH2 では Chained DMA を使うことで，Pack/Unpack が必要なくなり，メッセージ長がある程度小さい場合において高いバンド幅が得られる．

PEACH2 の DMA 通信では，ディスクリプタの登録方法によって主に 2 つの通信モードを提供している．

**ホストメモリモード**

ホスト CPU のメモリ上に必要なサイズのディスクリプタを作成しておき，通信開始時に必要なディスクリプタをホストメモリから読み出し，通信を行う．ホストメモリを使って多数のディスクリプタを管理できるため，連結が長い場合に有効である．しかし，ホストメモリへの読み出しが必要なため，最低レイテンシは約 2.3 $\mu$sec にとどまる．

**内蔵メモリモード**

PEACH2 の FPGA に内蔵されているメモリにディスクリプタを登録する．ディスクリプタを格納するメモリに高速なアクセスが可能なため，通信のレイテンシが短縮される．特にデータサイズが小さいときに優位な性能を持ち，256 Btye までの通信で約 2.0 $\mu$sec の低レイテンシを実現する．しかし，内蔵メモリの容量には限りがあるため，最大で 1,024 個のディスクリプタまでしか登録することができない．

通信データサイズが小さい場合，内蔵メモリモードはホストメモリモードに比べより低レイテンシであるが，ある程度通信データサイズが大きくなるとその差はほとんどなくなる．本論文における袖領域交換では通信サイズが 256 B より大きくなるため，ホストメモリモードと内蔵メモリモードによる性能差が全体の性能に影響することはないと考えている．一方で，ホストメモリモードと内蔵メモリモードの切替えは flag を変更するだけで行えるため，より高速な通信が期待できる「内蔵メモリモード」を評価に使用する．

### 2.4 GPUDirect Support for RDMA

PEACH2 では GPU 間の直接通信を行うために，NVIDIA が提供する GPUDirect Support for RDMA 機能 [17] を用いる（以降「GDR」と略す）．GDR は，CUDA5.0 以降および Kepler アーキテクチャ世代以降の GPU を用いることで，GPU 上のデバイスメモリを PCIe アドレス空間にマッピングすることができる．同じ PCIe アドレス空間に属する GPU どうしでは，ホストメモリを経由することなく直接 PCIe 上でデータの転送が可能になる．TCA では，ノード間にまたがって PCIe アドレス空間を共有することができるため，ノード間の GPU 間直接通信を実現することができる．

前述のとおり，各ノード内の PEACH2，GPU 群，InfiniBand HCA は CPU 内の PCIe スイッチを介して接続されるが，2 つの CPU ソケットをまたいだアクセスにおいて重要な性能低下問題が発生する．Sandy Bridge 以降の世代の Intel Xeon CPU では，ソケット間をまたぐ QPI（QuickPath Interconnect）を介して双方に接続されている PCIe デバイス間通信が可能であるが，その際のバンド幅が著しく低下することが知られている [12]．図 1 において，各ノード内の GPU0，1 と GPU2，3 間の GDR を行う場合この問題が発生する．このため，現状では PEACH2 がアクセスする GPU は CPU0 側の GPU0，1 のみに限定している．

### 2.5 PEACH2 を用いたプログラミング

PEACH2 による GPU 並列プログラムは，NVIDIA 社が提供する CUDA 環境上で動作することが前提である．つまり，TCA を用いたプログラムでも，GPU の管理（メモリの確保，ホスト・デバイス間のデータ転送，カーネル関数の起動など）は，一般的な GPU プログラミングモデルと同様である．これに加え，PEACH2 による通信を行うためには PCIe アドレスを直接指定する必要があり，これは一般的な配列のポインタとは型が異なる．TCA を用いたプログラミングでは *tcaHandle* と呼ばれるメモリハンドルを定義し，PCIe アドレスの管理を容易にしている．RDMA Put プロトコルによる通信では，リモートノードの書き込み先アドレスが必要になる．そのため，送信先の *tcaHandle* が必要となり，TCP/IP や MPI を用いてノード間で交換を行う．これは，InfiniBand を用いた通信でも同様な制限があり，一般的な手法であるといえる．

DMA による通信を行うためには，TCA API である **tcaSetDMADesc_Memcpy()** を用いて送信先の *tcaHandle*，使用する DMA のチャネル番号，配列のオフセットなどを含むディスクリプタを作成し，ディスクリプタをポインタで連結する．**tcaStartDMADesc()** で，ディスクリプタを設定した DMA のチャネル番号を指定し，連結されたディスクリプタの通信が連続して開始される．この通信は非同期で行われ，**tcaWaitDMADesc()** によって通信の完了を待機する必要がある．

TCA による通信を何度も行う場合，あらかじめ通信の設定をしておくことで，DMA による通信を起動するだけで通信を開始することができる．このようにすることで，時間発展ループ内で何度も通信を行う場合に TCA の低レ

イテンシ通信を最大限活用することが可能になる．

## 3. XcalableACC

XMP は文献 [7], [8]，XACC は文献 [10], [11] に詳しいが，本章ではそれらの概要を説明する．

### 3.1 XMP と OpenACC

XMP は，分散メモリ型並列計算機上でプログラミングを行うための PGAS 並列言語である．逐次のプログラムに OpenMP に類似した指示文を挿入することで，データの分散や同期，並列計算を行うことができる．XMP の指示文は，「#pragma xmp」から始まる指示文を持つ．また，XMP は全プロセスが同じプログラムを実行する SPMD（Single Program Multiple Data）モデルである．通常，メモリアクセスはローカルメモリのデータに対する参照であるが，他のノードのデータを参照するには XMP の指示文を使い，ノード間通信をする必要がある．

一方，OpenACC [9] は，プログラムの一部をアクセラレータにオフロードするための指示文ベースのプログラミングモデルであり，「#pragma acc」から始まる指示文を持つ．OpenACC の指示文で指定されたプログラムの領域は，GPU などのアクセラレータ上で実行される．一般的に，GPU などのアクセラレータは CPU のメインメモリ（以降「ホストメモリ」と呼ぶ）と独立したメモリ（以降「デバイスメモリ」と呼ぶ）を持っている．OpenACC では，必要なデータの転送をホストメモリとデバイスメモリ間で暗黙的に行うことできる．しかし，OpenACC はノード内のアクセラレータに対するデータ転送やオフロードしか記述することができない．

XACC は，通常の XMP と OpenACC 指示文に加え，アクセラレータ間のデータ通信をするために XMP の指示文を拡張することで，アクセラレータ搭載の並列計算機におけるプログラムの生産性と性能を両立させることを可能にする．

### 3.2 XACC のプログラミングモデル

**図 4** に XACC のサンプルコードを示す．2〜5 行目は，分散配列 *a[]* を定義するための XMP の指示文であり，データの分散や並列実行主体への割当てを記述する．7 行目は OpenACC の **data** 指示文で，8〜14 行目の領域はデバイスメモリでのデータの確保，スコープに入るときに転送が行われ，スコープを抜けるときに，データがホストに書き戻される．9〜13 行目は XMP の **loop** 指示文で各ノードにデータを分散し，OpenACC の **parallel** 指示文が XMP の分散配列をアクセラレータ上で実行するようにスレッド分割を行う．

次に，**図 5** に XACC による袖領域交換の指示文とその通信イメージを示す．XMP および XACC では，袖領域の

```
1  double a[N];
2  #pragma xmp nodes p(4)
3  #pragma xmp template t(0:N−1)
4  #pragma xmp distribute t(block) onto p
5  #pragma xmp align a[i] with t(i)
6  ...
7  #pragma acc data copy (a)
8    {
9  #pragma xmp loop on t(i)
10 #pragma acc parallel loop
11     for (int i = 0; i < N; i++) {
12       a[i] += 1.0;
13     }
14   }
15 ...
```

**図 4**　XACC のサンプルコード
**Fig. 4**　Sample code of XACC.

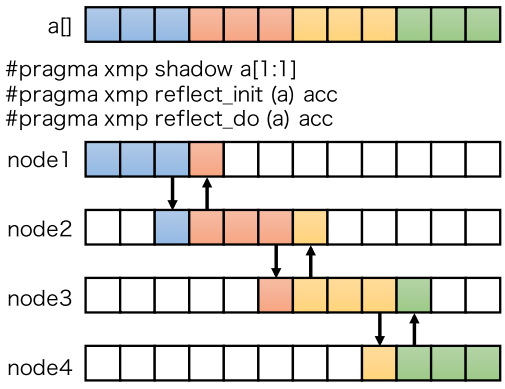


**図 5**　袖領域交換の定義と同期
**Fig. 5**　Definition and synchronization of halo exchange.

ことを「shadow」，その同期を「reflect」と定義している．図 5 の **shadow** 指示文は，袖領域を定義する分散配列を定義する．この例では，上側のノード，下側のノードそれぞれに対して袖幅 1 として定義されている．**reflect_init** 指示文は，隣接ノードの設定や転送データのオフセット計算などの内部処理を設定するものであり，これは通信領域が変わらない限り一度実行するだけでよい．**reflect_do** 指示文が宣言されたところで，**reflect_init** で設定された通信が実行される．

## 4. GPU 間通信の基礎評価

本章では，TCA と InfiniBand の通信性能について議論するために，隣接ノード間 GPU 通信の基礎評価を行う．ここでは特に，単純な連続データ通信だけでなく，多次元ステンシル計算で必要となる，多次元の袖領域（ステンシル計算で隣接領域間の境界となる部分）の通信に着目する．ハイブリッド通信において TCA が得意とする通信と InfiniBand 通信をどのように組み合わせるかが性

表 1　評価環境（HA-PACS/TCA）
**Table 1**　Performance evaluation environment (HA-PACS/TCA).

| | |
|---|---|
| CPU | Intel Xeon E5-2680v2 2.8 GHz ×2 sockets |
| GPU | NVIDIA Tesla K20X ×4 |
| Main Memory | DDR3 1,866 MHz 128 GB |
| GPU Memory | GDDR5 6 GB/GPU |
| Interconnection | InfiniBand: Mellanox Connect-X3 Dual-port QDR<br>TCA: PEACH2 Board |
| OFED | Mellanox OFED-2.2-1.0.1 |
| OS | CentOS release 6.4 |
| MPI | MVAPICH2-GDR 2.0<br>（MV2_USE_CUDA=1） |
| GPU Compiler | CUDA 6.0<br>NVIDIA Driver 340.32 |

能の鍵となるため，GPU の配置と利用するネットワークの組合せに関する最適化を視野に入れ，基礎評価を行う．評価環境として，**表 1** に示す HA-PACS/TCA の 2 ノードを用いる．TCA による通信でホップ数が増えないように，図 2 の TCA ネットワーク上で隣接するようにプロセスを配置する．InfiniBand 経由の GPU 間通信を利用するための MPI として，オハイオ州立大学が開発している MVAPICH2-GDR [18]（以降「MV2GDR」と略す）を用いる．MV2GDR は，TCA と同様に NVIDIA の GDR を用いて，InfiniBand を経由した GPU 間高速通信を実現している．HA-PACS/TCA の InfiniBand HCA は，PCIe Gen3 ×8 上に QDR ×4 クラスのインタフェースが 2 rails 実装されているが，本論文では，主に小メッセージにおけるレイテンシに着目し，TCA の 1 リンクに対して InfiniBand の 1 リンクとしたときの性能比較を行う．よって，性能評価では InfiniBand の 1 ポートのみを利用する．2.4 節で述べたように，QPI を経由した GPU 間直接通信は性能が極端に低下してしまうという問題があるため，図 1 の環境では，TCA の測定には GPU0 どうし，MPI の測定には GPU2 どうしの性能を測定する．一方，ハイブリッド通信では，TCA の性能を有効に利用するため，CPU0 側の GPU0, 1 を対象とする．しかし，MV2GDR が InfiniBand 経由で GPU 間通信を行う場合，QPI を経由する必要があり，QPI を経由しない MV2GDR よりも性能が低下してしまう．

**図 6** にノード間の GPU どうしで 1 次元配列の Ping-Pong 通信を行った場合の性能を示す．凡例の「TCA」は，TCA 経由による GPU0 どうしの通信，「MV2GDR」は InfiniBand 経由による GPU2 どうしの通信，「MV2GDR-QPI」は InfiniBand 経由による QPI をまたいだ GPU0 どうしの通信を示す．これより，TCA は MV2GDR に対して 512 KB まで優位な性能を示し，特に比較的小さいメッセージにおいて高い性能を示していることが分かる．しかし，MV2GDR-QPI の通信では 256 KB までのバンド幅が最高でも 290 MB/s 程度しか出ていない．デフォルトの MV2GDR は，以下のようにデータサイズによってプロトコルスイッチが行われる [19]．ここでメッセージサイズを $x$ とする．

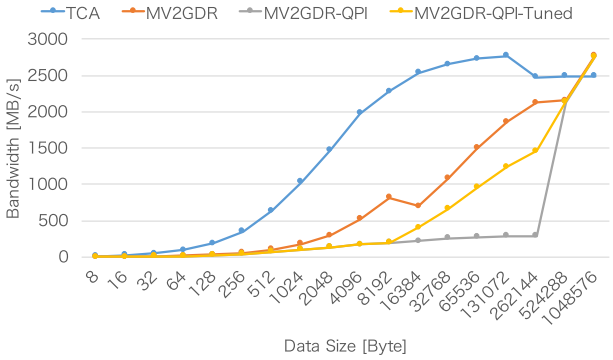


**図 6**　1 次元配列の Ping-Pong 通信性能
**Fig. 6**　Ping-Pong Communication performance of 1D array.

$x \leq$ **8 KB**
　GDR による GPU 間直接通信．

**8 KB** $< x \leq$ **256 KB**
　一度ホストにデータをコピーし，その後 GDR を用いてリモートの GPU へデータを書き込む．

**256 KB** $< x$
　ローカルホストへデータ転送，ホスト間のメッセージパッシング，リモートノードでのホストからデバイスへのデータ転送という 3 つの通信をパイプラインで処理する．これによって GDR による通信よりも通信効率が良くなる．

本来 QPI を経由する通信は性能が低下してしまうが，512 KB からはホストメモリを経由するため「MV2GDR-QPI」の性能は「MV2GDR」と同様になっていることが分かる．これは，QPI によって極端に性能が低下するのは，デバイス間での PCIe による通信の場合だけであり，QPI を超えたホストメモリのアクセスはわずかな性能低下にとどまるためである．MV2GDR は，実行時の環境変数でホスト経由の通信を行うようにするデータサイズを切り替えることができる．本論文における測定では，GDR のみで通信するデータサイズを 8 KB とするのが我々の調査によって最適であることが分かり，「MV2GDR-QPI-Tuned」がこれに該当する．つまり，8 KB までは GDR による GPU 間直接通信，それ以上ではホスト経由の通信に切り替わる選択が，MV2GDR の中で行われる．

次に，3 次元配列の袖領域通信について基礎評価を行う．**図 7** に 3 次元配列の袖領域アクセスパターン，**図 8** および**図 9** に各通信パターンの Ping-Pong 通信性能を示す．3 次元配列では，$jk$-平面はメモリアドレスが連続する Block 転送で，$ik$-平面は $N$ 要素の連続転送が $N \times N$ 周期で現れ

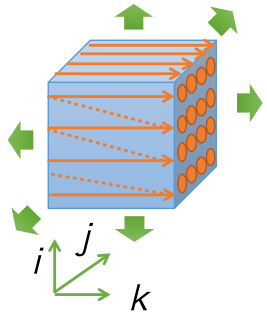


図 7　3 次元配列の袖領域交換の通信パターン
**Fig. 7**　3D communication pattern.

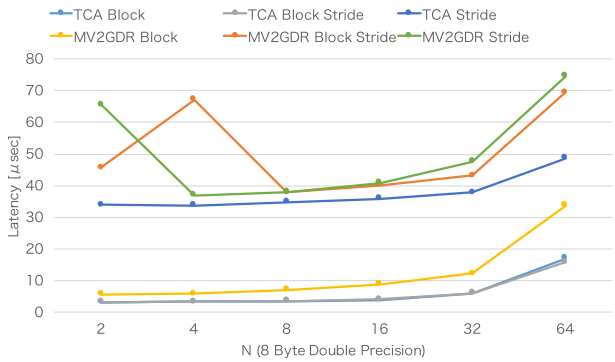


図 8　3 次元配列の Ping-Pong 通信によるレイテンシ
**Fig. 8**　Ping-Pong communication latency of 3D cube.

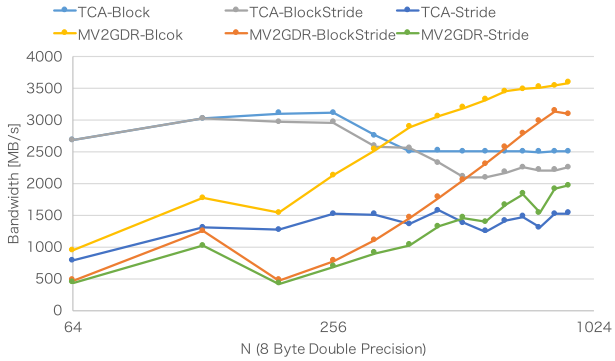


図 9　3 次元配列の Ping-Pong 通信によるバンド幅
**Fig. 9**　Ping-Pong communication bandwidth of 3D cube.

る Block Stride 転送である．$ij$-平面は，1 要素の転送が $N$ 周期で現れる Stride 転送になる．一般的に，Block Stride 通信や Stride 通信は，細粒度の通信を何度も行う必要があり，通信効率が悪い．そのため，InfiniBand 上の MPI などでは，バッファとして用いる配列にデータを Packing し，相手ノードに転送をした後，Unpacking をすることで通信の効率を上げている．TCA では，Block 転送および Block Stride 転送には Chained DMA を用い，Stride 転送には Pack/Unpack を行う．

図 8 および図 9 はそれぞれ，サイズ $N^3$ の 3 次元配列における各面のデータを隣接ノード間で通信する場合の Ping-Pong 通信のレイテンシとバンド幅を示している．Block 転送について，TCA はサイズが小さい場合においても非常に高いバンド幅を示している．TCA と MV2GDR の性能は，$N = 320 \sim 384$ で逆転する．図 8 では Latency の折れ線が重なっているように，TCA の Block Stride 転送は，Block 転送とほぼ同等の通信性能がある．これより，PEACH2 の DMAC に搭載しているハードウェア Block Stride 機能が非常に有用であることが分かる．一方，MV2GDR は Pack/Unpack を行う必要があるため，Block 転送に比べ性能が低く，$N = 576 \sim 640$ と大きなサイズまで TCA の性能が優位である．特に，図 8 のようにデータサイズが小さいときには Pack/Unpack が大きなレイテンシ増加の原因になっていることが分かる．Stride 転送では，ともに Pack/Unpack が必要であるため全体の性能差は他の通信方向よりも小さく，$N = 512$ で逆転する．

## 5. TCA/InfiniBand ハイブリッド通信

本章では，TCA と InfiniBand によるハイブリッド通信を提案する．

### 5.1 ハイブリッド通信の概要

TCA は，低レイテンシ通信を実現するが，PEACH2 実装を用いた適用範囲は PCIe などのハードウェアの制約や実装効率の面で数十ノードを一組とするサブクラスタを構成するにとどまっている．しかし，より大きな問題やサイズに対応するためにはサブクラスタをまたいだ通信が必要とされる．一方で，システム内のすべてのノードを InfiniBand によってフラットに接続することは自然であり，TCA は InfiniBand によるクラスタにおけるローカルな通信を加速するネットワークととらえることができる（そのローカルな高速通信が可能な単位がサブクラスタである）．そこで，高いバンド幅やスケーラビリティを持つ InfiniBand ネットワークに，局所的な通信に対して低レイテンシ通信を実現する TCA を加えることによって，スケーラビリティと InfiniBand のみでは得られない通信性能の向上を目的としたハイブリッド通信を実現する．このことにより，単に 2 つの通信路を束ねて全体のバンド幅を向上させるのではなく，それぞれの通信の特徴に応じたチャネルを選択することで，全体の通信性能を向上させることを目指す．このハイブリッド通信は，ステンシル計算における袖領域交換などにおいて，TCA と InfiniBand の通信がそれぞれを補完しあうことで全体の通信性能が向上するだけではなく，サブクラスタをまたいだ集団通信においても有効であると考えている．松本らは，TCA のサブクラスタ内における集団通信を実装し評価を行っている [20]．アプリケーションを強スケーリングさせる際には，集団通信の各メッセージサイズが小さくなるため，InfiniBand では大きなボトルネックとなってきた．TCA では，強スケーリングした際のメッセージサイズでも高い性能が得られる．そこで，TCA と InfiniBand によるサブクラスタ間通信を組み

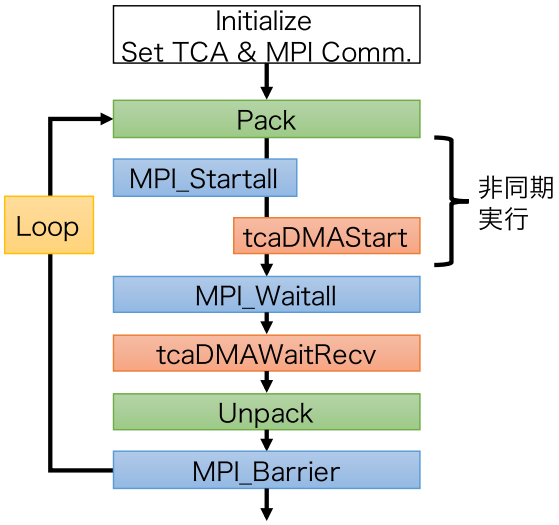


図 10　袖領域交換の流れ
**Fig. 10**　Flow of halo exchange.

合わせることで，各サブクラスタ内では TCA の高速な通信を活かし，最終的にサブクラスタ間の InfiniBand を経由して交換するデータ量を最小限にすることで，全体として低レイテンシ通信を行うことが期待できる．このような階層的な低レイテンシの集団通信は，単に InfiniBand のバンド幅を増やすだけでは実現することはできず，InfiniBand に TCA を加えることによるハイブリッド通信によって実現できる．

### 5.2　ハイブリッド通信による袖領域交換

4 章より，TCA と InfiniBand 経由の通信はデータサイズや通信パターンによって性能特性が異なることが分かった．そのため，単純に TCA 通信では通信できない相手に対して MV2GDR の通信を用いるだけでは，多くの場合において通信性能のバランスがとれず，全体の性能低下につながってしまう恐れがある．そこで，ハイブリッド通信による袖領域交換では，通信パターンによって TCA または MV2GDR の通信を適宜選択し，TCA と MPI の特徴を活かした通信を行うことを考える．2 次元配列の袖領域交換では，Block 通信（行方向）と Stride 通信（列方向）の通信パターンがあり，3 次元配列では，図 7 より，*jk*-平面は Block 通信，*ik*-平面は Block Stride 通信，*ij*-平面は Stride 通信となっている．

**図 10** に袖領域交換の流れを示す．隣接するノードに対して，Block Stride 通信または Stride 通信が発生する場合，Initialize フェイズであらかじめ Pack/Unpack に必要なバッファ配列をデバイスメモリ上に確保しておく．本論文では，実行中に袖領域のサイズが変わるようなことがないことを前提とし，TCA および MPI の通信を永続通信として設定する．この前提は，通常の Domain Decomposition によるステンシル計算で一般的に成り立つものである．そして，時間発展ループ中に通信が実行されたときに，非連続領域を Pack し，MPI と TCA の通信を開始する．これらの通信は非同期で実行され，Wait 関数でそれぞれの通信の完了を待つ．その後，Unpack を行うことで袖領域交換が完了する．

### 5.3　XACC におけるハイブリッド通信の実装

本節では，XACC による袖領域交換を行うための指示文 **reflect_init** と **reflect_do** に対して，提案するハイブリッド通信を適用した実装について述べる．

#### reflect_init 指示文

**reflect_init** 指示文は，同期を行う袖領域を設定する指示文である．この指示文は，図 10 の Initialize フェイズにあたる．ハイブリッド通信では，前述のとおり，通信方向の特性に応じて TCA または MPI の通信を割り当てる．そのために，XACC の言語処理系から reflect の対象とする配列の情報を手に入れ，各通信方向に対する行数，ブロック長，ギャップ長から通信パターンを判別する．行数が 1 であれば，その通信方向は連続領域である．一方，行数が 1 より大きく，ブロック長がスカラー 1 要素であれば Stride 領域，それ以外は Block Stride 領域だと分かる．Stride 領域は Pack/Unpack する必要があり，本実装では CUDA カーネルで実装されている．判別した通信の種類によって，TCA または MPI の API を用いて通信の設定を行う．PEACH2 による通信では，同じ通信方向に異なる PEACH2 からのパケットが流れるとき，または複数の PEACH2 から同時にパケットを受信したときにパケットの衝突が発生する．この衝突によって，通信路のバンド幅が低下することが知られている [6]．袖領域交換の通信パターンでは，後者の衝突が発生する可能性がある．そのため本実装では，一度に複数の PEACH2 からパケットを受信することがないように，通信するノードを pairwise して，ノードごとに DMA のディスクリプタを連結する順番を制御している．一方，MPI 通信ではつねに InfiniBand のスイッチを経由して通信が行われるため，このような衝突は発生しない．MPI 通信は，非同期 Send/Recv プロトコルによる通信を設定する．

#### reflect_do 指示文

**reflect_do** 指示文は，同期通信を実際に実行するための指示文である．この指示文は，図 10 の Pack から **MPI_Barrier()** までのフェイズである．まず，Stride 通信がある場合，通信の前後に Pack/Unpack が実行される．そして，**reflect_init** 指示文で設定した通信を，**MPI_Startall()** および **tcaDMAStart()** 関数で TCA の DMA 通信，MPI の Send/Recv 通信を開始する．これらは非同期通信であるため，**MPI_Waitall()** および **tca-**

```
 1 double u[XSIZE][YSIZE];
 2 #pragma xmp nodes p(YNODES, XNODES)
 3 #pragma xmp template t(0:YSIZE−1, 0:XSIZE−1)
 4 #pragma xmp distribute t(block, block) onto p
 5 #pragma xmp align u[i][j] with t(j, i)
 6 #pragma xmp shadow u[1:1][1:1]
 7 ...
 8
 9 int main(void) {
10 ...
11 #pragma acc data copy(u)
12   {
13 #pragma xmp reflect_init (u) acc // 袖領域交換の通信設定
14 while (...) {
15 #pragma xmp reflect_do (u) acc // 袖領域通信の実行
16 // 内点の計算
17 ...
18     } // while loop の終了
19   } // data 指示文の終了
20 ...
21 }
```

図 11　XACC による袖領域交換コードの記述
**Fig. 11**　Description of halo exchange code with XACC.

**DMAWaitRecv()** 関数でそれぞれの通信の完了を待つ．

### 5.4　XACC におけるハイブリッド通信による袖領域交換

本論文では，**reflect_init** 指示文と **reflect_do** 指示文に対して提案するハイブリッド通信の実装を適用した．本節では，袖領域交換の通信に対して従来の MPI+TCA によるハイブリッド通信と XACC によるハイブリッド通信のプログラムの記述を比較する．また，プログラム実行時のデータの分割と通信のマッピングについて述べる．

まず，MPI+TCA によるハイブリッド通信と XACC による袖領域交換のコードを比較する．**図 11** に XACC によるハイブリッド通信，**図 12** に MPI+TCA によるハイブリッド通信を用いた袖領域交換のプログラムを示す．ともに 2 次元配列 “u” に対しての袖領域交換を行うプログラムの例を示している．2 次元配列の袖領域は 4 つあり，図 12 では North，South，East，West とし，North および South は Block 通信の MPI，East および West は Stride 通信の TCA を割り当てる．図 11 では，XACC 内のランタイムで通信の種類を判断して，通信を割り当てる．

図 11 の XACC による袖領域交換のプログラムの 2〜6 行目は 3 章で述べたように，XMP の指示文を用いて通信するノードや配列の袖領域の設定を行う．11 行目でデバイスメモリ上に必要なデータを確保し，12〜19 行目のスコープでは，ホストとデバイス間への通信が発生し，スコープに入るときに必要データがデバイスメモリ上に転送，スコープから抜けるときにホストメモリへデータが書き戻される．13 行目の **reflect_init** 指示文は，5.3 節で述べたように袖領域交換を行うための通信設定である．この指示文では，袖領域交換に必要なバッファの確保，offset の計算，MPI や TCA 通信の設定などが行われる．一般的に，袖領域交換を用いるプログラムでは，14〜18 行目の while loop のように袖領域交換後に内点の計算をするルーチンを何度も実行することが多い．15 行目の **reflect_do** 指示文で，13 行目の **reflect_init** 指示文で設定した通信を実際に実行し，それらの通信の完了を待つ．以上が XACC による袖領域交換プログラムの記述である．

一方，図 12 の MPI+TCA のコードにおいて，図 11 の 2〜6 行目に相当するのが 1〜3 行目である．MPI によるプログラムでは，2 行目のようにプロセスマッピングに応じてどのノードと通信を行うかを計算し，それに応じて Rank 番号を割り当てる必要がある．ここでは North，South，East，West それぞれに対して通信相手がいればその Rank 番号，いなければ **MPI_PROC_NULL** を設定する．図 11 の 11〜19 行目のスコープに相当するのが，4〜7 行目および 39〜41 行目である．ここでは，TCA API である *tcaMalloc()* および *tcaFree()* 関数でデバイスメモリ上の配列 “u” の確保と開放を行う．その後，必要なデータを *cudaMemcpy()* 関数でホスト・デバイス間の通信を行う．次に，図 11 の 13 行目における袖領域交換の設定に相当するのが，8〜22 行目である．Stride 通信のための Pack/Unpack 用のバッファを 9 行目で確保し，10〜11 行目で TCA の Handle へ登録する．13〜15 行目および 16〜18 行目は，*MPI_Send_init()*，*MPI_Recv_init()* 関数で North，South 方向に通信する相手ノードが存在する場合 MPI の通信を設定する．19〜22 行目は *tcaSetDMADesc_Memcpy()* 関数を用いて East，West 方向に通信相手がいる場合 TCA の通信を設定する．TCA の通信では，これらの通信は Chained DMA によって行うため，22 行目の *tcaDescSet()* 関数で **tca_desc** ポインタに連結する．図 11 の 14〜18 行目の while loop に相当するのが，23〜38 行目でその中で図 11 の 15 行目の袖領域通信を実行する部分に相当するのが 24〜35 行目である．まず，24〜26 行目で Stride 領域の Packing を行い，これが完了次第 28 行目で TCA の通信，29 行目で MPI の通信を開始する．これらの通信は非同期で行われるため，30 行目で MPI の通信，31〜32 行目で TCA の通信を待機する．その後，33〜35 行目で Unpack を行い，Stride 領域にデータを書き込むことで袖領域交換が完了する．以上より，図 12 の MPI+TCA のプログラムは，配列の確保，通信の設定など非常に煩雑であったが，図 11 の XACC によるハイブリッド通信では，飛躍的にプログラムの記述が容易になり，かつ見通しの良いプログラミングが可能になることが分かった．

次に，データの分割と通信のマッピングについて述べる．XMP および XACC は最大 7 次元までのデータを扱うこと

```
1  double u[XSIZE/XNODES][YSIZE/YNODES], *d_u; // ‘‘u’’はホストメモリ上，‘‘d_u’’はデバイスメモリ上に確保される
2  setCommRank(my_rank, &north_rank, &south_rank, &east_rank, &west_rank); // 通信するノードの設定
3  ...
4  tcaMalloc((void**)&d_u, u_byte, tcaMemoryGPU); // 袖領域交換を行うMatrix ‘‘u’’の確保
5  tcaCreateHandle(&u_handle, d_u, ...); // TCA の Handle への登録
6  cudaMemcpy(d_u, u, u_byte, cudaMemcpyDefault); // Host to Device 通信
7  ...
8  // Stride 領域を Pack/Unpack するためのバッファの確保と TCA の Handle 登録
9  tcaMalloc((void**)&d_east_send_buffer, ...); tcaMalloc((void**)&d_east_recv_buffer, ...);
10 tcaCreateHandle(&west_send_handle, d_west_send_buffer, ...); tcaCreateHandle(&east_send_handle, d_east_send_buffer, ...);
11 tcaCreateHandle(&west_recv_handle, d_west_recv_buffer, ...); tcaCreateHandle(&east_recv_handle, d_east_recv_buffer, ...);
12 // MPI 通信の設定
13 if (north_rank != MPI_PROC_NULL) {
14   MPI_Send_init(d_north_send_ptr, ...); MPI_Recv_init(d_south_recv_ptr, ...);
15 }
16 if (south_rank != MPI_PROC_NULL) {
17   MPI_Send_init(d_south_send_ptr, ...); MPI_Recv_init(d_north_recv_ptr, ...);
18 }
19 // TCA 通信の設定と DMAC への通信登録
20 if (east_rank != MPI_PROC_NULL) tcaSetDMADesc_Memcpy(tca_desc, &west_recv_handle[east_rank], &east_send_handle, ...);
21 if (west_rank != MPI_PROC_NULL) tcaSetDMADesc_Memcpy(tca_desc, &east_recv_handle[east_rank], &west_send_handle, ...);
22 tcaDescSet(tca_desc, dmac_ch);
23 while (...) {
24   // Pack
25   if (west_rank != MPI_PROC_NULL) gpu_pack_vector_async(d_west_send_buffer, &d_u[west_offset], ...);
26   if (east_rank != MPI_PROC_NULL) gpu_pack_vector_async(d_east_send_buffer, &d_u[east_offset], ...);
27   // 通信開始と完了待ち
28   TCA_SAFE_CALL(tcaStartDMADesc(dma_ch));
29   MPI_SAFE_CALL(MPI_Startall(4, mpi_request));
30   MPI_SAFE_CALL(MPI_Waitall(4, mpi_request, MPI_STATUS_IGNORE));
31   if (east_rank != MPI_PROC_NULL) tcaWaitDMARecvDesc(&u_handle[east_rank], ...);
32   if (west_rank != MPI_PROC_NULL) tcaWaitDMARecvDesc(&u_handle[west_rank], ...);
33   // Unpack
34   if (west_rank != MPI_PROC_NULL) gpu_unpack_vector_async(&d_u[west_offset], d_west_recv_buffer, ...);
35   if (east_rank != MPI_PROC_NULL) gpu_unpack_vector_async(&d_u[east_offset], d_east_recv_buffer, ...);
36   // 内点の計算
37   ...
38 } // while loop の終了
39 cudaMemcpy(u, d_u, u_byte, cudaMemcpyDefault) // Device to Host 通信
40 tcaFree(d_u, tcaMemoryGPU);
41 ...
```

**図 12**　MPI+TCA ハイブリッド通信による袖領域交換コードの記述
**Fig. 12**　Description of halo exchange code with MPI+TCA hybrid communication.

ができるが，ハイブリッド通信を行うにあたって，データの分割サイズと通信パターンを議論するために 2 次元分割までを対象とする．この場合，2 次元配列は行方向・列方向に分割され，行方向に Block 通信となる MV2GDR，列方向に Stride 通信となる TCA を割り当てる．一方，3 次元配列の場合 3 通りの分割パターンが存在する．しかし，TCA および MPI の通信性能を最大限活かすためには，TCA には Block Stride 通信，MPI には Block 通信を割り当てるのが最適である．つまり，3 次元配列に対しては図 7 の $i$ 方向を MPI による Block 通信，$j$ 方向を TCA による Block Stride 通信を割り当てる．これによって，袖領域交換中に Pack/Unpack が必要なくなり，最小限のオーバヘッドで通

信が可能になる．

また，XMP における通信は，コンパイラが自動的に通信を起こすのではなく，ユーザが明示的に記述しない限り発生しない仕様である．これによって，意図しないタイミングでの通信を防ぐことができ，見通しの良いプログラミングが可能になることに加え，ユーザによる性能チューニングが容易になる．ハイブリッド通信は，データの分割方法とプロセスマッピングが性能に大きく影響する．現在の XACC では，プロセス数とデータ分割によって，明示的にノードへのプロセスマッピングをジョブスクリプトという形で記述している．これによって，つねに最適なマッピングが提供されるため，最大限の性能を引き出すことが可能になる．しかしながら，このような方法はユーザが通信パターンやプロセスマッピングを熟知する必要がある．そのため，XACC では経験が浅いユーザに対しても最適なマッピングを提供することを目的として，より明確かつ容易にプロセスマッピングを行うための指示文などの検討を行っている．この最適化機構は現在検討中のため，本論文ではユーザがノード配置と配列のトポロジを熟知し，最適化を行うことを前提として評価を行う．このため，現時点ではプログラムに性能可搬性がないことになるが，これについては今後の課題としたい．

## 6. 性能評価

本章では，XACC にハイブリッド通信を導入したことによるオーバヘッドを評価する．また，2 次元ラプラス方程式，3 次元姫野ベンチマークを用いて，隣接ノード間の袖領域交換を TCA のみ，MV2GDR（InfiniBand+MPI）のみ，ハイブリッド通信で行い，実行性能を比較する．2 次元ラプラス方程式および 3 次元姫野ベンチマークの XACC によるソースコードはそれぞれ文献 [10], [11] に示されている．OpenACC コンパイラとして，筑波大学の田渕らが開発している Omni OpenACC Compiler [21] を用いる．データの分割は 2 次元分割までとし，すべての通信が 1 ホップで完了する分割の組合せとする．また，評価には 4 章と同様に HA-PACS/TCA を用い，使用するノード数は最大 16 ノードとする．本論文におけるハイブリッド通信では，サブクラスタ内で仮想的にグループを分割することで，サブクラスタをまたいだ通信を再現する．評価はデータサイズを固定し，分割方法およびノード数を変化させた強スケーリングを行う．本来は複数のサブクラスタを利用し，本当に TCA 通信が分離された状況で実験すべきであるが，実験環境の制約により，ここではサブクラスタが「仮想的に」さらに小さいサブクラスタで構成されているという想定で，最大 16 ノードまでの実験を行う．

本評価では，ラプラス方程式および姫野ベンチマークはそれぞれ 2 次元分割までとするため，分割 $(i, j, k)$ の $k$ 次元方向は分割されず，つねに 1 となる．そのため，ラプラス方程式における分割（4, 2）と姫野ベンチマークにおける分割（4, 2, 1）では同じプロセスマッピングとサブクラスタの分割になる．**図 13** に分割（4, 2, 1）および分割（2, 8, 1）のプロセスマッピングと仮想サブクラスタの割当て例を示す．図中の数字が Rank 番号，それぞれの色が仮想的に分割したサブクラスタ，点線は各分割パターンにおいて使用されないパスを示している．分割 $(i, j, k)$ において，$j$ 次元が分割された場合 TCA の通信（ラプラス方程式では Stride 通信，姫野ベンチマークでは Block Stride 通信）が発生する．$i$ 次元が分割された場合 MV2GDR による Block 通信が発生し，この通信が仮想的なサブクラスタ間の通信となる．本論文の姫野ベンチマークでは $k$ 次元は分割しないため，Stride 通信は発生しない．たとえば，図 13 の分割（2, 8, 1）において，Rank 0〜7 と Rank 8〜15 はそれぞれ別の仮想サブクラスタに属し，この仮想サブクラスタ間では TCA 通信をいっさい行わない．分割（2, 8, 1）の Rank6 に着目すると Rank5 および 7 と TCA，Rank14 と MV2GDR による袖領域交換が行われる．また，計算に使用する GPU は 4 章の基礎評価と同様に，TCA およびハイブリッド通信では CPU0 側の GPU0，MV2GDR は CPU1 側の GPU2 とし，1 ノードにつき 1 GPU のみを計算に用いることとする．

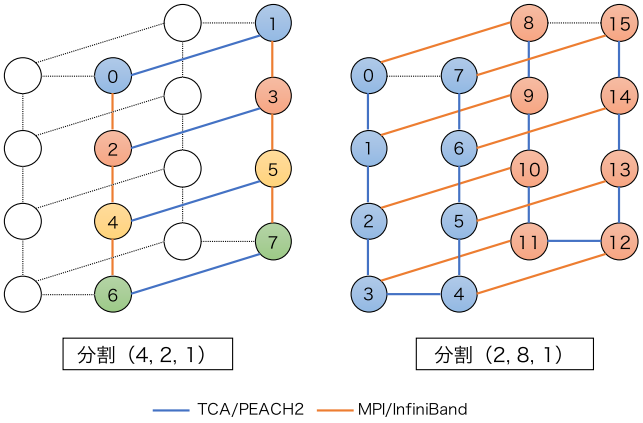


**図 13**　プロセスマッピングと仮想サブクラスタの分割
**Fig. 13**　Process mapping and division of virtual sub-cluster.

### 6.1 XACC によるオーバヘッドの評価

本評価では，8,192 × 8,192 の 2 次元配列に対して，MPI+TCA のハイブリッド通信を Hand-coding したプログラムと XACC で記述したプログラムを用いて，袖領域交換の設定（**reflect_init**）に要する時間と実際の通信時間（**reflect_do**）を比較する．**reflect_init** では，MPI の通信設定，TCA の通信設定およびディスクリプタの連結，Pack/Unpack 用のバッファの確保などを行う．また **reflect_do** では，**reflect_init** で設定した通信の起動および完了待ちを行う．袖領域交換の実行時間は 1,000 イテレーションの平均をとり，それぞれ 10 回の平均時間を**表 2**

表 2　袖領域交換の実行時間 [$\mu$sec]
**Table 2**　Execution time of halo exchange [$\mu$sec].

| | reflect_init | reflect_do |
|---|---|---|
| Hand-coding | 0.166 | 116.472 |
| XACC | 0.176 | 125.926 |

に示す．評価には HA-PACS/TCA を 16 ノード用いて，図 13 の分割（2, 8, 1）と同様なプロセスマッピングとサブクラスタの分割を行う．

これより，**reflect_init**，**reflect_do** ともに実行時間に大きな差はないが，わずかに XACC の性能が低いことが分かる．**reflect_init** の時間は全プログラム実行では誤差となるため，**reflect_do** に着目すると，XACC におけるこの部分の実行時間が，Hand-coding に比べ 8.1%増加している．これは，XACC では **reflect_do** が実行されるたびに，通信やその完了待ちをするかどうかの判定を各通信面に対して総当りで行う必要があるためだと推測される．しかし，全実行時間ではこれにさらに計算時間が追加されるため，全体としての性能低下はきわめて低いと考えられ，記述の簡単化による生産性の向上を考えると十分に有効といえる．

### 6.2　ラプラス方程式の評価

本評価の問題サイズとして Small（8,192 × 8,192），Large（16,384 × 16,384）の 2 種類を用いる．分割（$i, j$）は，（行方向，列方向）の分割を示し，たとえば分割 $2 \times 4$ の場合，分割されたデータサイズは Small で 4,096 × 2,048 となる．

**図 14** と**図 15** に 2 次元ラプラス方程式の実行性能を示す．グラフの縦軸は実行性能である GFLOPS 値，横軸は分割方法と使用したノード数である．2 つのグラフより，3 つの通信の性能差があまり大きくないことが分かる．これは，ノード間のデータ分割は 2 次元であるが，OpenACC によるスレッド分割が行方向の 1 次元しか行えない Omni OpenACC Compiler の制限による．ラプラス方程式の評価では，実行時間に対して通信時間は多くて 1 割程度にとどまっているが，今後 2 次元分割によりキャッシュが有効に使えることが期待できるため，計算時間が短くなり，実行時間全体に占める通信時間の割合が大きくなるため，通信性能の向上が全体の性能向上に大きく影響すると考えられる．

図 14 と図 15 より，ノード数が少ない場合は実行性能に顕著な差はないが，問題サイズ Small，Large ともに 16 ノード分割を行ったときに性能差が生まれていることが分かる．データサイズを固定し，ノード数を増やしていく強スケーリングでは，分割されたデータが徐々に小さくなっていき，それにともない袖領域交換が必要なデータサイズも減少する．16 ノードを用いた場合，1 ノードの通信データサイズは Small で 64 KB，Large で 128 KB となる．こ

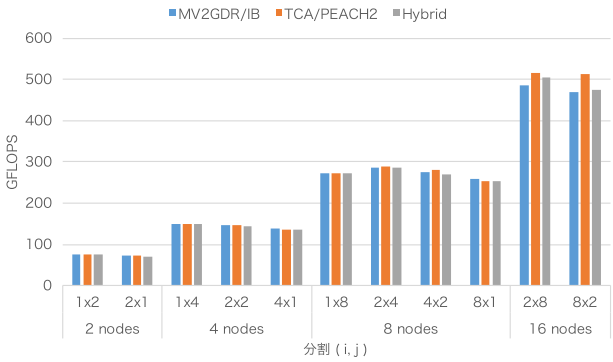


**図 14**　ラプラス方程式：データサイズ Small
**Fig. 14**　Laplace's equation: Small size.

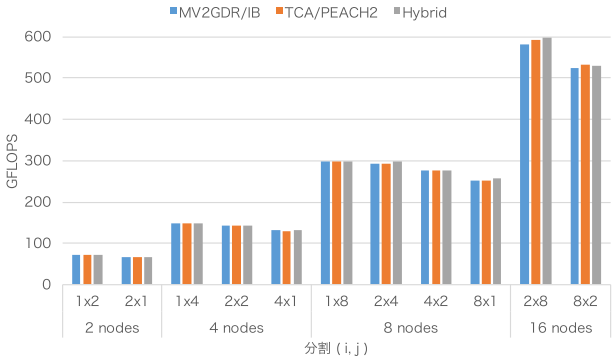


**図 15**　ラプラス方程式：データサイズ Large
**Fig. 15**　Laplace's equation: Large size.

のため，ある程度大きなデータを送るときに高いバンド幅を発揮する MPI 通信はプロトコル変換などのオーバヘッドにより十分な性能を出すことができない．一方，TCA 通信は PCIe パケットによる直接通信が可能なため，通信データサイズが小さい場合に効果的であることが分かる．同様に，ハイブリッド通信の MPI 通信に対して性能が向上していることが分かる．特に，図 14 の分割 $2 \times 8$ において，ハイブリッド通信が MPI に対して高速であることが分かる．行方向の分割数が 2 であるため，MV2GDR による通信は 1 方向のみで，その他の 2 方向に対しては TCA が用いられる．通信データサイズが小さいため，ハイブリッド通信においても TCA は有効に働いていることが分かる．

### 6.3　姫野ベンチマークの評価

本評価の問題サイズは Small（$64 \times 64 \times 128$），Middle（$128 \times 128 \times 256$），Large（$256 \times 256 \times 512$）とする．分割（$i, j, k$）は図 7 に対応している．本評価には，$i$ 方向と $j$ 方向を分割し，$k$ 方向は分割せずつねに 1 とする．5.2 節で述べたように，MPI と TCA の通信特性から $i$ 方向（Block アクセス）の通信を MV2GDR，$j$ 方向（Block Stride アクセス）の通信を TCA に割り当てる．また，姫野ベンチマークではイテレーションの最後にノード間でスカラーの Allreduce が必要であるが，本評価ではすべての通信パターンにおいて，一度ホストへデータをコピーし，

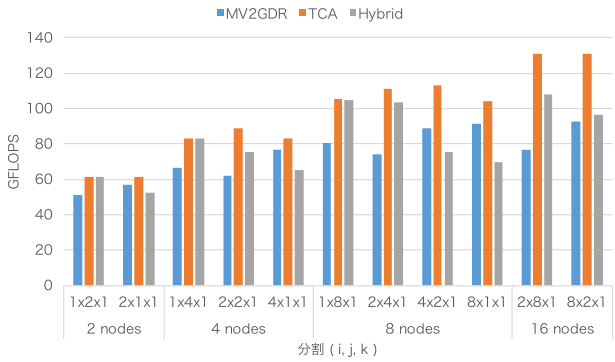


図 16　姫野ベンチマーク：データサイズ Small
**Fig. 16**　Himeno benchmark: Small size.

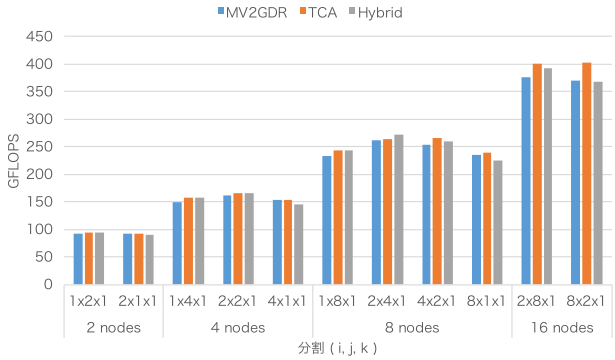


図 17　姫野ベンチマーク：データサイズ Middle
**Fig. 17**　Himeno benchmark: Middle size.

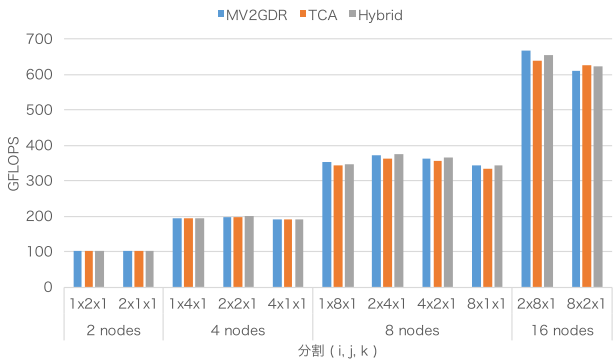


図 18　姫野ベンチマーク：データサイズ Large
**Fig. 18**　Himeno benchmark: Large size

**MPI_Allreduce()** を使ってデータを更新する．

**図 16**，**図 17**，**図 18** にそれぞれ問題サイズ Small，Middle，Large の実行性能を示す．ラプラス方程式の結果と同様に，縦軸に実行性能，横軸に分割方法とノード数を示す．3 次元配列の袖領域交換は，2 次元配列のそれと比べ通信データサイズが大きい．図 18 の問題サイズ Large では，通信データサイズが 512 KB を超えてしまっている．通信データサイズ 512 KB は，TCA と MV2GDR の通信性能分岐点であり，これを超えたデータサイズでは TCA の低レイテンシ通信のメリットを活かせない．図 17 の問題サイズ Middle では，通信データサイズは多くて 256 KB にとどまり，TCA が有効な範囲である．このため，TCA を用いたときの性能が MV2GDR に対して高いことが示されている．ハイブリッド通信では，分割（$2 \times 8 \times 1$）では MV2GDR よりも高速であり，分割（$2 \times 4 \times 1$）では MV2GDR だけでなく TCA のみの通信よりも高速な結果が得られた．この 2 つの分割方法に共通することは，全体の通信量のなかで Block Stride 通信が 8〜9 割を占めていることである．MV2GDR のみの通信に対して，Pack/Unpack が必要ないこと，Block Stride 通信が優位なデータサイズであることが影響し，優位な結果が得られたと考えられる．図 16 の問題サイズ Small は，通信サイズが最大で 64 KB であり，分割（$2 \times 8 \times 1$）において MV2GDR に対して TCA は 70%の性能向上が得られた．ハイブリッド通信は，問題サイズ Middle でもあったように Block Stride 通信の割合が多い分割方法を選択することで TCA のみの通信には及ばないものの，MV2GDR に対しては最大で 40%の性能向上が得られている．

一方，図 16 の各通信方法による性能差は図 17 および図 18 に比べて非常に大きい．これは，全体の実行時間に対する通信時間と計算時間の割合が影響していると考えている．一般的に，ステンシル計算のアプリケーションでは，問題サイズを固定しノード数を増やしていく強スケーリングをすると，計算時間は通信性能にかかわらず減少する．本論文における姫野ベンチマークにおいても同様に，計算時間は一定の割合で減少する．しかし，通信時間は通信の粒度が小さくなるため，計算時間ほどは小さくならず，全体の実行時間に対して通信時間が占める割合が大きくなる．そのため，問題サイズが小さいときに通信性能が全体の性能に大きく影響するため，問題サイズによる性能差の大小が生じていると考えられる．

また，TCA 通信に対して Hybrid 通信が劣るケースが見受けられる．ここで，**図 19**，**図 20** に問題サイズ Small および Middle における袖領域交換の実行時間を示す．凡例の「TCA」は TCA のみを通信に使った場合の袖領域交換の実行時間，「Hybrid TCA 部分」および「Hybrid MV2GDR 部分」はハイブリッド通信内でそれぞれ別々に袖領域交換の実行時間を測定したものである．ハイブリッド通信による袖領域交換の実行時間は，「Hybrid TCA 部分」または「Hybrid MV2GDR 部分」の実行時間の大きい方に律速される．たとえば，分割（$4 \times 2 \times 1$）は図 16 の問題サイズ Small および図 17 の Middle でともにハイブリッド通信の性能が TCA のみの通信に劣っている．同分割において，図 20 では「TCA」と「Hybrid MV2GDR 部分」の実行時間が近いことが分かる．このため，図 17 では TCA 通信とハイブリッド通信の性能差が小さい．図 19 では，「Hybrid MV2GDR 部分」の実行時間が「TCA」と比較して大きいため全体の性能差が大きいことが分かる．

今回の評価では，最大で 16 ノードまでの評価であったが，実際に複数のサブクラスタを用いて 16 ノードより大き

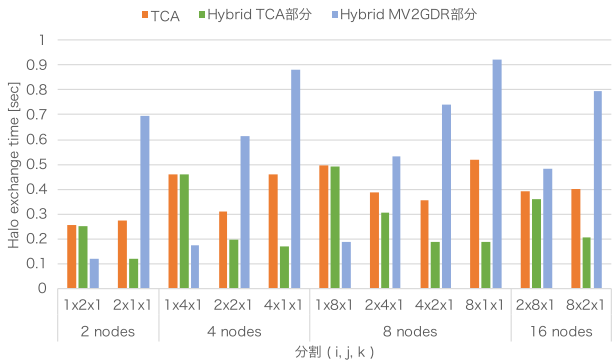


図 19　姫野ベンチマーク：データサイズ Small の通信時間
**Fig. 19**　Himeno benchmark: Communication time on small size.

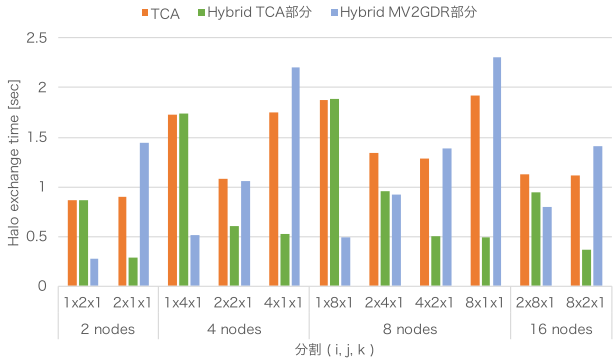


図 20　姫野ベンチマーク：データサイズ Middle の通信時間
**Fig. 20**　Himeno benchmark: Communication time on middle size.

いスケールで評価を行う際に，MV2GDR に対してハイブリッド通信は大きなアドバンテージがあると考えられる．

### 6.4　ハイブリッド通信の考察

本研究の端緒は，TCA を用いたクラスタにおいて，より大きな問題やシステムサイズに対応するためにサブクラスタをまたいだ通信が必要となることから，TCA と InfiniBand によるハイブリッド通信を提案し評価することであった．サブクラスタ間を接続する通信ネットワークは InfiniBand のみに頼らざるをえないが，ラプラス方程式と姫野ベンチマークの評価の結果，サブクラスタ内で InfiniBand に TCA を併用することにより，分割方法とデータサイズの適切な選択により，ハイブリッド通信が MV2GDR に対して性能上優位であることが分かった．特に，TCA 通信では Block Stride 通信が有効であり，ハイブリッド通信の性能に大きく影響している．すなわち，このハイブリッド通信では単に InfiniBand のバンド幅を増強するだけでなく，TCA に適した通信にそれを適用することで，その効果を高めることができる．

ハイブリッド通信は複数のサブクラスタを用いた環境を想定しているが，その一方で，TCA での通信が可能なサブクラスタ内においても，TCA のみを用いる場合に対し，ハイブリッド通信が優位な場合もある．6.3 節の姫野ベンチマークでは，問題サイズ Middle の分割（$2 \times 4 \times 1$）がそれに該当する．このとき，通信サイズは最大で Block 通信が 32 KB，Block Stride 通信が 256 KB（$= 128\,\text{KB} \times 2$）となり，Block Stride 通信が全体の通信量の 8 割を占めている．図 20 より，分割（$2 \times 4 \times 1$）ではハイブリッド通信の TCA 部分と MV2GDR 部分の実行時間がほぼ等しくなっていることが分かる．このように，TCA の Block Stride 通信と MV2GDR の Block 通信の実行時間が近ければ近いほど，TCA および InfiniBand の通信を最大限利用することができ，オーバヘッドが最小となるため，ハイブリッド通信の性能は向上し，TCA のみの通信に対しての優位性があると考えられる．これは，TCA のみの通信に対してだけでなく，16 ノードより多くのノードを使ったスケーラブルな環境において，MV2GDR のみの通信に対する優位性も示すことができると考えている．

また，本評価は XACC によるプログラミングを行ったことで，通常煩雑になりがちな袖領域交換のハイブリッド通信を指示文だけで記述できるようにした．このような典型的な通信パターンを容易に記述することができ，生産性と性能向上を両立させることが可能であることが分かった．

## 7.　関連研究

GPUDirect では，コモディティネットワークの InfiniBand と NVIDIA の Kepler アーキテクチャを搭載した GPU により，GPU 間直接通信を実現している [22]．これを MPI のインタフェースに適用したものとして，オハイオ州立大学の MVAPICH2-GDR がある [17]．PEACH2 でも同様に GDR を用いた通信をするが，InfiniBand の通信には HCA 間の通信には PCIe とは異なるプロトコルを用いている．一方，PEACH2 は PCIe のパケットをそのまま利用できるのでプロトコル変換のオーバヘッドがなく，InfiniBand の GDR に比べて低いレイテンシで通信が可能である．ハイブリッド通信でも InfiniBand を用いるが，すべての通信が必ずしも InfiniBand を経由することはないので，オーバヘッドは低減される．

NVIDIA 社が提供する NVLink [23] は，ノード内の GPU 間を直接結合する技術である．現在の実装は，PCIe Gen3 に基づいているが，今後専用の通信バスを提供することでさらに効率の良い GPU 間直接通信が可能になるといわれている．これは，我々が提案してきた TCA コンセプトであり，これまで HA-PACS/TCA で実証してきたことは意義があるものであるといえる．また，NVLink はノード内の GPU 間通信を加速するものであり，ノード間の通信には今までどおり，InfiniBand などのコモディティネットワークを併用することが考えられる．このことからも，我々が提案しているハイブリッド通信は意義のあるものであると考えられる．

本論文では，TCA 通信と InfiniBand 通信という 2 種類の通信系のハイブリッド通信として問題をとらえたが，たとえば 1 ノードに他ポートの PCIe スイッチを搭載し，多くの GPU を実装するようなシステムも存在する [24]．そのようなシステムでは，ノード内の GPU に問題をどう分割するかが重要であり，PCIe スイッチ上でのノード内通信は，本論文における TCA を用いた通信に共通するものがある．よって，本論文で考察した TCA 通信と InfiniBand 通信の，次元方向での使い分けはこのようなシステムにおける問題分割にも一部適用可能であると考えられる．

## 8. 結論

本研究は，TCA サブクラスタをまたいだ通信を実現するために，TCA と InfiniBand によるハイブリッド通信を提案している．ハイブリッド通信は，TCA と InfiniBand で異なる通信特性に着目し，最適な通信を選択することで，InfiniBand+MPI だけでは得られない性能向上を期待している．本論文では，2 次元ラプラス方程式と 3 次元姫野ベンチマークによる，隣接ノード間の袖領域交換の評価を行い，ハイブリッド通信は MV2GDR に対して最大 40%の性能向上が得られた．同時に，通常ハイブリッド通信では，TCA と MPI の通信を別々に記述する必要があるが，XACC のフレームワークを用いることで，典型的な通信パターンである袖領域交換を指示文によって行うことができる．これによって，ユーザのプログラミングコストは大幅に減り，同時に性能面でも有意な結果が得られた．

本論文では，袖領域交換の通信に対するハイブリッド通信を実装，評価を行ったが，今後は松本らの TCA による集団通信ライブラリ [20] をもとに，ハイブリッド化を進めていき，それぞれの評価とともにアプリケーションへ適用させる．また，今回は 16 ノードによる評価であったが，実際にサブクラスタを複数用いた，よりスケーラブルな環境での評価を行いたいと考えている．

**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，筑波大学計算科学研究センターの学際共同利用プロジェクト研究課題「メニーコアおよび演算加速機構を持つクラスタシステム向け並列プログラミング言語の開発と評価」および研究課題「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] TOP500 Supercomputer Sites, available from ⟨http://top500.org/⟩.
[2] HPCI 技術ロードマップ白書 2012 年 3 月，入手先 ⟨http://open-supercomputer.org/wp-content/uploads/2012/03/hpci-roadmap.pdf⟩.
[3] 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.4, pp.14–25 (2013).
[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，Mike Clark：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（ハイパフォーマンスコンピューティング），Vol.2014-HPC-145, No.43, pp.1–9 (2014).
[5] Fujita, N., Fujii, H., Hanawa, T., Kodama, Y., Boku, T., Kuramashi, Y. and Clark, M.: QCD Library for GPU Cluster with Proprietary Interconnect for GPU Direct Communication, *12th International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms* (*HeteroPar2014*) *in conjunction with Euro-Par2014* (Aug. 2014).
[6] 藤井久史，塙　敏博，児玉祐悦，朴　泰祐，佐藤三久：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（ハイパフォーマンスコンピューティング），Vol.2015-HPC-148, No.12, pp.1–9 (2015).
[7] Lee, J. and Sato, M.: Implementation and Performance Evaluation of XcalableMP: A Parallel Programming Language for Distributed Memory Systems, *3rd International Workshop on Parallel Programming Models and Systems Software for High-End Computing* (*P2S2*), pp.413–420 (Sep. 2010).
[8] Nakao, M., Lee, J., Boku, T. and Sato, M.: Productivity and Performance of Global-View Programming with XcalableMP PGAS Language, *Proc. 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (*CCGrid 2012*), *CCGRID '12*, pp.402–409 (2012).
[9] OpenACC, available from ⟨http://www.openacc-standard.org/⟩.
[10] 田渕晶大，村井　均，朴　泰祐，佐藤三久：XcalableMP と OpenACC の統合による GPU クラスタ向け並列プログラミングモデル，情報処理学会研究報告（ハイパフォーマンスコンピューティング），Vol.2014-HPC-145, No.39, pp.1–7 (2014).
[11] 中尾昌広，村井　均，下坂健則，田渕晶大，塙　敏博，児玉祐悦，朴　泰祐，佐藤三久：XcalableACC: OpenACC を用いたアクセラレータクラスタのための PGAS 言語 XcalableMP の拡張，情報処理学会研究報告（ハイパフォーマンスコンピューティング），Vol.2014-HPC-146, No.7, pp.1–11 (2014).
[12] 塙　敏博，児玉祐悦，藤井久史，朴　泰祐，佐藤三久：HA-PACS/TCA システムにおけるマルチノード GPU 間通信性能評価，情報処理学会研究報告（計算機アーキテクチャ），Vol.2014-ARC-208, No.20, pp.1–8 (2014).
[13] Hanawa, T., Kodama, Y., Boku, T. and Sato, M.: Interconnection Network for Tightly Coupled Accelerators Architecture, *IEEE 21st Annual Symposium on High-Performance Interconnects* (*HOT Interconnects 21*), pp.79–82 (Aug. 2013).
[14] Hanawa, T., Kodama, Y., Boku, T. and Sato, M.: Tightly Coupled Accelerators Architecture for Minimizing Communication Latency among Accelerators, *3rd International Workshop on Accelerators and Hybrid Exascale Systems* (*AsHES*) *in conjunction with IEEE 27th International Parallel and Distributed Processing Symposium* (*IPDPS*), pp.1030–1039 (May 2013).
[15] 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，藏増嘉伸，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（ハイパフォーマンスコンピューティング），Vol.2011-HPC-130, No.21, pp.1–7 (2011).
[16] Altera Corporation: Stratix IV Device Handbook, available from ⟨http://www.altera.co.jp/literature/lit-stratix-iv.jsp⟩.
[17] NVIDIA Corporation: NVIDA GPUDirect, available from ⟨https://developer.nvidia.com/gpudirect⟩.
[18] MVAPICH2: A High Performance MPI Library for NVIDIA GPU Clusters with InfiniBand, available from ⟨http://on-demand.gputechconf.com/gtc/2013/presentations/S3316-MVAPICH2-High-Performance-MPI-Library.pdf⟩.
[19] MVAPICH2-GDR 2.0 README, available from ⟨http://mvapich.cse.ohio-state.edu/static/media/mvapich/MV2-GDR-README.txt⟩.
[20] 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 Collective 通信の実装と予備評価，情報処理学会研究報告（ハイパフォーマンスコンピューティング），Vol.2014-HPC-147, No.23, pp.1–10 (2014).
[21] Tabuchi, A., Nakao, M. and Sato, M.: A Source-to-Source OpenACC Compiler for CUDA, *11th International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms* (*HeteroPar2013*) *in conjunction with Euro-Par2013*, Vol.8374, pp.178–187, 2013.
[22] Mellanox Technologies: Mellanox OFED GPUDirect, available from ⟨http://www.mellanox.com/page/products_dyn?product_family=116⟩.
[23] NVIDIA NVLINK HIGH-SPEED INTERCONNECT, available from ⟨http://www.nvidia.com/object/nvlink.html⟩.
[24] 濱田　剛：DEGIMA における LINPACK 電力性能評価，情報処理学会研究報告（計算機アーキテクチャ研究会報告），Vol.2012-ARC-199, No.12, pp.1–24 (2012).

**小田嶋 哲哉**　（学生会員）

昭和 63 年生．平成 23 年筑波大学情報学群情報科学類卒業．平成 25 年同大学大学院システム情報工学研究科博士前期課程修了．修士（工学）．同年 4 月より同大学院システム情報工学科博士後期課程在籍．計算機アーキテクチャ，並列プログラミング言語，アクセラレータ等に興味あり．

**朴 泰祐**　（正会員）

昭和 35 年生．昭和 59 年慶應義塾大学工学部電気工学科卒業．平成 2 年同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．昭和 63 年慶應義塾大学理工学部物理学科助手．平成 4 年筑波大学電子・情報工学系講師，平成 7 年同助教授，平成 16 年同大学大学院システム情報工学研究科助教授，平成 17 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．平成 14 年および平成 15 年度情報処理学会論文賞，平成 23 年 ACM ゴードンベル賞，平成 24 年度情報処理学会山下記念研究賞各受賞．IEEECS，ACM 各会員．

**塙 敏博**　（正会員）

平成 10 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，平成 15 年東京工科大学コンピュータサイエンス学部講師，平成 19 年筑波大学計算科学研究センター研究員，平成 20 年筑波大学大学院システム情報工学研究科准教授を経て，平成 25 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．平成 25 年度山下記念研究賞，平成 7 年度情報処理学会論文賞．IEEE CS 会員．

**児玉 祐悦**　（正会員）

昭和 63 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．平成 13 年独立行政法人産業技術総合研究所に改組．平成 23 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．平成 27 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（平成 2 年度），市村学術賞（平成 7 年）等受賞．電子情報通信学会，IEEE CS 各会員．

**村井 均**　（正会員）

平成 8 年 3 月京都大学大学院工学研究科情報工学専攻修士課程修了．平成 8 年 4 月～平成 22 年 3 月 NEC．平成 22 年 3 月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博士課程修了．平成 22 年 4 月より理化学研究所．現在，同計算科学研究機構・プログラミング環境研究チームにて HPC 向けプログラミング環境の研究に従事するとともに，同エクサスケールコンピューティング開発プロジェクト・アーキテクチャ開発チームにてポスト「京」の開発に従事．

**中尾 昌広**　（正会員）

平成 17 年同志社大学大学院工学研究科博士前期課程修了．同年 NTT アドバンステクノロジ株式会社入社．平成 22 年同志社大学大学院工学研究科博士後期課程修了．筑波大学計算科学研究センター研究員，理化学研究所計算科学研究機構特別研究員を経て，平成 27 年から同機構研究員．ハイパフォーマンスコンピューティングの研究に従事．平成 27 年度情報処理学会山下記念研究賞受賞．

**田渕 晶大**　（学生会員）

平成 3 年生．平成 25 年筑波大学情報学群情報科学類卒業．平成 27 年同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博士前期課程修了．修士（工学）．同年 4 月より同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博士後期課程在籍．並列プログラミング言語，アクセラレータ等に興味あり．

**佐藤 三久**　（正会員）

昭和 34 年生．昭和 57 年東京大学理学部情報科学科卒業．昭和 61 年同大学大学院理学系研究科博士課程中退．同年新技術事業団後藤磁束量子情報プロジェクトに参加．平成 3 年通商産業省電子技術総合研究所入所．平成 8 年新情報処理開発機構並列分散システムパフォーマンス研究室室長．平成 13 年から平成 27 年まで筑波大学システム情報系教授．平成 19 年度より平成 24 年度まで同大学計算科学研究センターセンタ長．平成 22 年より理化学研究所計算科学研究機構プログラミング環境研究チームリーダ．平成 26 年より同機構エクサスケールコンピューティング開発プロジェクト副プロジェクトリーダ．理学博士．並列処理アーキテクチャ，プログラミングモデルと言語およびコンパイラ，計算機性能評価技術等の研究に従事．IEEE，日本応用数理学会各会員．