Vol.2015-HPC-150 No.43
2015/8/6

アクセラレータ向け並列プログラミング言語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]　村井 均[4]　中尾 昌広[4]　田渕 晶大[1]　佐藤 三久[4,2]

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．しかしながら，ノードを跨ぐ GPU 間の通信は，ホストメモリを経由して行うためレイテンシが増加し，特にアプリケーションの強スケーリングおける性能の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，ノード間の 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）を提案し，そのプロトタイプ実装として PCI Express に基づく PEACH2（PCI Express Adaptive Communication Hub version 2）システムを開発している．PEACH2 に基づく TCA システムでは，ハードウェア制限や実装効率の面で数十ノードを一組とするクラスタが限界であり，より大きな問題やサイズに対応するためにはこれらをさらに階層的に結合したシステムを構築し，それらを跨ぐ通信を行う必要がある．そこで我々は，PEACH2 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，プログラミングが煩雑になる．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．本稿では，姫野ベンチマークの袖領域交換の通信にこのシステムを適用し，評価を行う．その結果，PEACH2 と InfiniBand によるハイブリッド通信を用いた場合，InfiniBand のみの場合に比べ，最大 40%の性能向上が達成できた．同様に，集団通信である Broadcast，Allgather にハイブリッド通信を適用した結果，それぞれ最大で 21%，46%の速度向上が達成できた．これによって，ハイブリッド通信は TCA の適用範囲を拡大するだけでなく，TCA の低レイテンシ通信と InfiniBand の高バンド幅通信を組み合わせることによって，一般的な InfiniBand による通信に対して優位性があることが示された．

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 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て，GPU を接続する PCI Express（以降「PCIe」と略す）の通信性能は相対的に低く，特にノード内の 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，ノード間を跨ぐ GPU 間のデータの交換は，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時にはレイテンシが増加し，通信性能が低下する原因となっている．例えば，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性を求める場合，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．さらに，Allgather のような集団通信においても，強スケーリングの場合，通信データサイズが小さくなり，性能のボトルネックとなる．また一般的に，Broadcast，Allreduce 等の集団通信でも GPU クラスタでの通信ボトルネックは重要な問題である．

そこで，我々はノード間に跨る 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

[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] 筑波大学 計算科学研究センター
Center for Computational Sciences, University of Tsukuba
[3] 東京大学 情報基盤センター
Information Technology Center, The University of Tokyo
[4] 理化学研究所 計算科学研究機構
RIKEN Advanced Institute for Computational Science
[a)] odajima@hpcs.cs.tsukuba.ac.jp

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（以下，本稿では単 PEACH2 に基づく TCA 実装を TCA/PEACH2 と記す）．TCA/PEACH2 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することが確認されている [3], [4], [5], [6].

また，現状では，PEACH2 の適用範囲は PCIe 等のハードウェア面の制約により，結合可能ノード数が数十ノードにとどまっている．本稿ではこれをサブクラスタと呼ぶ．より大きなシステムを構築するためには，サブクラスタを跨ぐ通信を実現する必要がある．一つの典型的な実装として考えられるのは，大規模システムの全ノードが InfiniBand で結合され，その中に TCA/PEACH2 でサブクラスタを構築する手法である．この場合，サブクラスタ内のノードは TCA/PEACH2 と InfiniBand の両方で結合され，サブクラスタ外のノードとは InfiniBand だけで結合される．

本研究では，TCA/PEACH2 の低レイテンシ通信や高い Block Stride 転送性能と，InfiniBand の高バンド幅通信に着目し，すべてのノードが InfiniBand で接続されているネットワークの局所的な通信に対して TCA/PEACH2 を加えることによって，システムのスケーラビリティと，InfiniBand ネットワークのみの場合に比べ，通信性能を向上させることを目的として，TCA+InfiniBand 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）を実現する．

また，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 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．一方，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 [7], [8]，さらに XMP のアクセラレータ搭載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，本稿では TCA+InfiniBand のハイブリッド通信システムを開発し，袖領域交換の通信を 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．本稿では，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て，その有効性と性能について評価を行う．同様に，集団通信である Broadcast と Allgather に対してもハイブリッド通信を適用し，評価を行う．

2. TCA と PEACH2

TCA 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を進めている．このコンセプトの実証システムとして，PEACH2 チップが開発され，TCA/PEACH2 として性能評価が進められている．TCA/PEACH2 では，PCIe をノード間通信に拡張することによって TCA を実現しているため，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 GPU や Intel MIC（Many Integrated Core processor）等に適用可能である．

PEACH2（PCI Express Adaptive Communication Hub version 2）は，PCIe リンクをノード間通信に拡張することにより，GPU 等のアクセラレータの通信範囲をノード間に広げるもので，FPGA（Altera 社 Stratix Ⅳ GX[15]）によって実装されている．この PEACH2 チップを搭載したボード同士を PCIe 外部ケーブルで接続し，TCA/PEACH2 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）[16] の拡張部として HA-PACS/TCA が構築され，運用されている．

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（Ivy Bridge アーキテクチャ）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 ×2 rails）が接続されている．図中，GPU は CPU に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 ボードまたは InfiniBand HCA に接続されているため，実際の通信は CPU 本体を介さずに行われる．

2.2 PEACH2

PEACH2 チップは，PCIe パケットの中継処理や DMA 転送などを行う，一種のインテリジェントな PCIe スイッチと見なせる．PEACH2 は高度な DMA コントローラ（以降「DMAC」と略す）を 4 チャネル搭載しており，高速な DMA や Chained DMA などが可能である．PEACH2 チップは，PCIe Gen2 ×8 ポートを 4 つ持ち，1 つはホスト CPU と接続し，残り 3 つのポートを隣接ノードの PEACH2 ボー

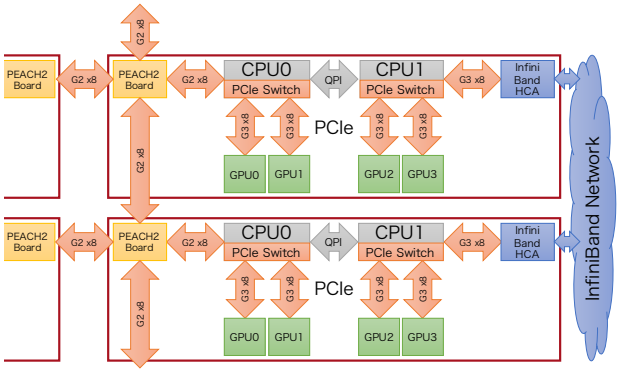

図 1 HA-PACS/TCA のノード構成

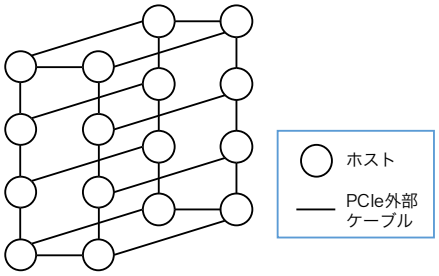

図 2 サブクラスタ：TCA ネットワーク構成

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伸びるリンクが 3 つに限られる．PEACH2 は PCIe パケットのルーティング機能を持つため，原理的には多数のノードを PCIe リンクでリング結合することにより，TCA/PEACH2 システム内のノード数を増やすことは可能である．しかしその場合，通信ホップ数が増加し，PEACH2 の持つ低レイテンシ通信を有効に活用できない．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することができるトポロジとして**図 2** のように 2×8 の 2 重リングトポロジを構成している．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．一方で，InfiniBand は HA-PACS/TCA のすべてノードを単一スイッチでフラットに接続している．

2.3 DMA 通信

PEACH2 には，PIO と DMA の 2 つの通信方式が提供されているが，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 では，リモートノードに対するアクセス方法として，基本的に RDMA Write プロトコルのみがサポートされている．ホスト上で，読み込み元・書き込み先の PCIe アドレス・通信データサイズ等を指定したディスクリプタを作成し，そのアドレスを指定することで通信を起動できる．また，複数のディスクリプタをポインタで連結しておくことで，連続した DMA 転送を行うことも可能である（Chained DMA）．よって，あらかじめ通信のパターン（通信相手と通信領域）が定められていれば，一度作成した DMA ディスクリプタを再利用することで，Chained DMA 機構により柔軟かつ高速な通信を実現することができる．また，各 DMA ディスクリプタには，連続領域に対するデータ転送だけでなく，Block Stride 転送のためのバースト長およびギャップを指定することも可能である．Block Stride 転送は多次元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（ステンシル計算で隣接領域間の境界となる部分）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 Block Stride 転送あるいは Chained DMA 機構を使うことで，Pack/Unpack が必要なくなり，メッセージ長が小さい場合においても高いバンド幅が得られる．

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 では，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 同士では，CPU の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/PEACH2 は，ノード間に跨って 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．

2.2 節に示した通り，HA-PACS/TCA の各ノード内の PEACH2 ボード，GPU 群，InfiniBand HCA は CPU 内の PCIe スイッチを介して接続されているが，2 つの CPU ソケットを跨いだアクセスにおいて重大な性能低下問題が発生する．Sandy Bridge 世代以降の Intel Xeon CPU は，ソケット間を跨ぐ QPI（QuickPath Interconnect）を介して双方に接続されている PCIe デバイス間の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/PEACH2 を用いたプログラムにおいても，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．また，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

があるが，これは一般的な配列のポインタとは型が異なる．そこで，TCA/PEACH2 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 Write 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であり，TCP/IP や MPI を用いて予めノード間で交換を行う．現在の TCA/PEACH2 の API では，このアドレス情報の交換には MPI を用いている．よって，厳密には TCA/PEACH2 のみで閉じたシステムにはなっていないが，通常のクラスタでは Ethernet 等で全系が結合されていることは普通であるため，このことにより一般性が失われるわけではない．TCA/PEACH2 によるプログラミングの流れを以下に示す．

(1) **tcaSetDMADesc_Memcpy()** を用いて，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，これらをポインタで連結しておく
(2) **tcaDescSet()** を用いて連結した DMA 通信を DMAC に登録する
(3) **tcaStartDMADesc()** を用いて DMAC を指定し，DMA 通信を発行する
(4) **tcaWaitDMADesc()** を用いて非同期通信の完了を待つ

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 を 1 度発行するだけで通信を開始することができる．これによって，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に活用することが可能になる．

3. XcalableACC

XMP については文献 [7], [8]，XACC については文献 [10], [11] にそれぞれ詳しいが，本章では本稿を理解するための概要のみを記す．

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（Partitioned Global Address Space）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 はすべての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．XMP のプログラムは，“#pragma xmp” から始まる指示文を持つ．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 の指示文を用いて，ノード間通信を行う必要がある．

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 は，ホストメモリとデバイスメモリ間のデータの転送を暗黙的または明示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処理を記述することしかできず，複数ノードに跨がる並列処理を意識していない．そこで XACC は，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することを可能にする．

```
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  #pragma xmp shadow a[1:1]
7  ...
8  #pragma acc data copy (a)
9    {
10 #pragma xmp reflect_init (a) acc // 袖領域交換の設定
11 #pragma xmp reflect_do (a) acc // 袖領域交換の実行
12 ...
13 #pragma xmp loop on t(i)
14 #pragma acc parallel loop
15     for (int i = 0; i < N; i++)
16       a[i] += a[i−1] + a[i+1];
17   }
18 ...
```

図 3 XACC のサンプルコード

3.1 XACC のプログラミングモデル

図 3 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り当てを記述する．6 行目は袖領域を定義する分散配列を指定する．8 行目は OpenACC の **data** 指示文で，9 ∼ 17 行目の領域はデバイスメモリでのデータの確保，スコープに入る時に転送が行われ，スコープを抜ける時に，データがホストメモリに書き戻される．10 行目は袖領域交換の通信を定義する指示文で，隣接ノードの設定や転送データに必要なオフセットの計算などの内部処理を設定する．これは通信領域が変わらない限り一度実行するだけで良い．11 行目は **reflect_init**

表 1 評価環境（HA-PACS/TCA）

CPU	Intel Xeon E5-2680v2 2.8GHz × 2 sockets
GPU	NVIDIA Tesla K20X × 4
Main Memory	DDR3 1866MHz 128GB
GPU Memory	GDDR5 6GB / GPU
Interconnection	InfiniBand: Mellanox Connect-X3 Dual-port QDR TCA: PEACH2 Board
OFED	Mellanox OFED-2.2-1.0.1
OS	CentOS release 6.4
MPI	MVAPICH2-GDR 2.0 (MV2_USE_CUDA=1, MV2_NUM_PORTS=1)
GPU Compiler	CUDA 6.5 NVIDIA Driver 340.32

で設定された通信が実行される．13 ～ 16 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．

4. GPU 間通信の基礎性能評価

本章では，TCA と InfiniBand の通信性能について議論するために，隣接ノード間の GPU 間通信の基礎性能評価を行う．ここでは，単純な連続データ通信だけではなく，多次元ステンシル計算で必要となる袖領域交換の通信にも着目する．ハイブリッド通信では，TCA と InfiniBand の通信をどのように組み合わせるかが性能の鍵となるため，GPU の配置と利用するネットワークの組み合わせに関する最適化を視野に入れ，基礎性能評価を行う．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 本実装されているが，本稿では，主に小メッセージにおける通信性能に着目するために，TCA の 1 リンクに対して InfiniBand の 1 リンクを用いた時の性能比較を行う．よって，性能評価では InfiniBand の 1 ポートのみを利用する．また，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問題があるため，図 1 の環境では，TCA の測定には GPU0 同士，MV2GDR の測定には GPU2 同士の性能を測定する．一方，ハイブリッド通信では，TCA の低レイテンシ通信を有効に活用するために，CPU0 側の GPU0，1 を対象とする．しかしながら，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．この影響について，併せて評価する．

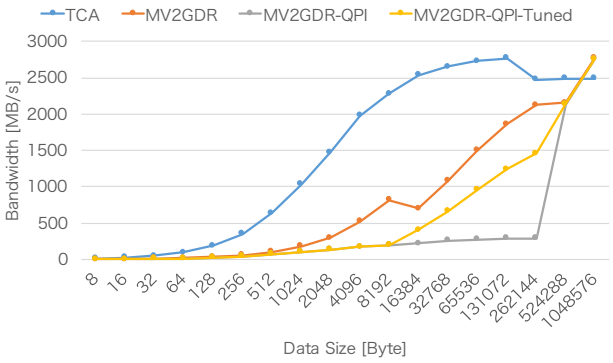

図 4 1 次元配列における Ping-Pong 通信の性能

図 4 にノード間の GPU 間における 1 次元配列の Ping-Pong 通信の性能を示す．凡例の「TCA」は，TCA 経由による GPU0 同士の通信，「MV2GDR」は InfiniBand 経由による GPU2 同士の通信，「MV2GDR-QPI」は InfiniBand 経由による QPI を跨いだ GPU0 同士の通信を示す．これより，TCA は MV2GDR に対して 512KB 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わかる．しかし，MV2GDR-QPI の通信では 256KB までのバンド幅が最高でも 290MB/s 程度しか出ていない．デフォルトの MV2GDR は，データサイズによって以下のような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．

$x \leq$ 8KB
: GDR による GPU 間直接通信

8KB $< x \leq$ 256KB
: 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

256KB $< x$
: 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

本来 QPI を経由する通信は性能が大きく低下してしまうが，プロトコルスイッチによって 512KB からはホストメモリを経由した通信となるため「MV2GDR-QPI」の性能は「MV2GDR」と同様になっている．つまり，QPI を経由したことによる性能低下は，デバイス間の 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ということである．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，我々

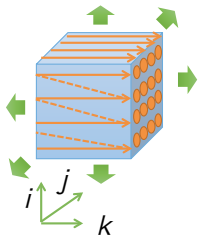

図 5 3 次元配列における袖領域のアクセスパターン

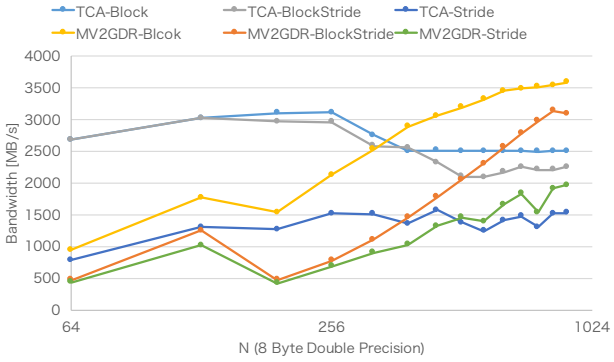

図 6 3 次元配列における Ping-Pong 通信の性能．縦軸はバンド幅，横軸は通信対象である 3 次元配列のサイズを示す．データサイズは N^3 の 3 次元配列の 1 面（$N \times N \times 8$ Byte）である．

のこれまでの調査により，GDR のみで通信するデータサイズを 8KB，以降はホストメモリを経由した通信とするのが最適であることがわかっている．これを施し，通信性能を最適化したのが同図の「MV2GDR-QPI-Tuned」である．本稿では今後，QPI を経由する MV2GDR の通信には前述の「MV2GDR-QPI-Tuned」を用いる．

次に，3 次元配列の袖領域通信の基礎性能評価を行う．**図 5** に 3 次元配列における袖領域のアクセスパターンを示す．3 次元配列では，jk-平面はメモリアドレスが連続する Block アクセス，ik-平面は N 要素の連続転送が $N \times N$ 周期で現れる Block Stride アクセス，ij-平面は 1 要素の転送が N 周期で現れる Stride アクセスとなる．一般的に，Block Stride 転送や Stride 転送は細粒度の転送を何度も行う必要があるため，転送効率が非常に悪い．そのため，InfiniBand 上の MPI では，バッファとして用いる配列にデータを Packing し，相手ノードにそのバッファを転送した後に，Unpacking をすることで転送の効率を上げている．TCA/PEACH2 では，Block Stride 転送に Chained DMA を用いることで，Pack/Unpack が必要なくなり，高い性能が得られる．一方，Stride 転送に関しては，Chained DMA を用いたとしても性能低下が避けられないため，MPI と同様， Pack/Unpack を用いた方がよいことが既存研究でわかっている [12]．

図 6 に，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の GPU 間で通信するパターンを想定し，これを Ping-Pong 通信として実施した場合の性能を示す．連続アドレスデータ（jk 平面）の Block 転送について，TCA/PEACH2 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おり，TCA/PEACH2 と MV2GDR の性能は，$N = 320 \sim 384$ で逆転する．一方，TCA/PEACH2 の非連続アドレスデータ（ik 平面）の Block Stride 転送は，$N = 384$ まで Block 転送とほぼ同等の通信性能を保っている．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わかる．一方，MV2GDR は同じ非連続アドレスデータ転送では，Pack/Unpack を行う必要があるため Block 転送に比べ性能が低く，$N = 512 \sim 576$ まで TCA/PEACH2 の性能が優位である．非連続アドレスデータ（ij 平面）の Stride 転送では，ともに Pack/Unpack を行うため，$N = 448 \sim 512$ で TCA/PEACH2 と MV2GDR の性能が逆転する．

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 GPU 間の低レイテンシ通信を実現するが，PEACH2 実装を用いた TCA/PEACH2 の適用範囲は PCIe 等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る．しかしながら，より大きな問題やサイズに対応するために，サブクラスタを跨いだ通信が必要となる．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想定として自然であり， 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捉えることができる（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，システムスケーラビリティを向上させ，また InfiniBand のみの場合を上回る通信性能を実現することを目的としたハイブリッド通信を実現する．これは，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だけではなく，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が目的である．具体的には，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ことが期待できる．

また，ハイブリッド通信はサブクラスタを跨いだ集団通信においても有効であると考えられる．松本らは，TCA/-PEACH2 による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], [21]．アプリケーションの強スケーリングにおいて，集団通信の各メッセージサイズが小さくなることは，InfiniBand でのボトルネックの一つとな

る．TCA/PEACH2 では，強スケーリングした際の短メッセージサイズでも高い性能が得られる．そこで，PEACH2 通信と InfiniBand 通信を階層的に用い，TCA/PEACH2 によるサブクラスタ内通信と InfiniBand によるサブクラスタ間通信を組み合わせることで，各サブクラスタ内では TCA の低レイテンシ通信を活かし，最終的にはサブクラスタ間の InfiniBand での高バンド幅を活用することにより，全体として高速な通信を行えることが期待できる．このような階層的な高速集団通信は，単に InfiniBand のバンド幅を増やすだけでは実現できず，InfiniBand ネットワークに TCA/PEACH2 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

ハイブリッド通信による袖領域交換については文献 [22] に詳しいが，ここではその概要を説明する．4 章より，TCA/PEACH2 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わかった．そのため，単純に TCA/PEACH2 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取れず，全体の性能向上が見込まれない．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/PEACH2 または MV2GDR の通信を適宜選択し，PEACH2 と InfiniBand の特徴を活かした通信を行うことを考える．3 次元配列では図 5 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．本稿では，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，通信パターンは Block 通信と Block Stride 通信のみとなる．3 次元配列の場合 3 通りの分割パターンが存在するが，4 章の基礎性能より，TCA に Block Stride 通信，MPI に Block 通信を割り当てるのが最適である．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することが可能になる．

5.3 XACC におけるハイブリッド通信の実装

本節では，3.1 節で説明した 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，袖領域交換の通信を設定する指示文である．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 が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．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，ネットワーク形状が基本的にトーラス結合であることから，同じ通信方向に異なる PEACH2 からのパケットが流れる時，または複数の PEACH2 から同時にパケットを受信した時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わかっている [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，通信するノードを次元ごとに 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する．一方，MPI 通信は常に InfiniBand のフラット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 で通信は，非同期 Send/Recv 通信を用いる．

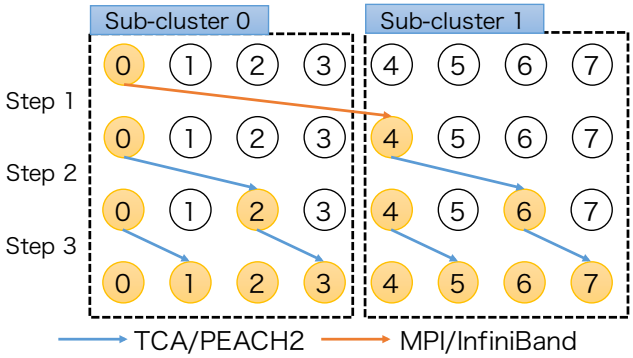

図 7 Broadcast 通信のアルゴリズム

reflect_do 指示文

reflect_do 指示文は，袖領域交換の通信を実際に実行するための指示文である．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 MPI の Send/Recv 通信，TCA の DMA 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 **tcaDMAWaitRecv()** 関数でそれぞれの通信の完了を待つ．

5.4 ハイブリッド通信による Broadcast 通信

本節では，Broadcast 通信に対してサブクラスタを跨いだハイブリッド通信を適用することについて述べる．本稿では，Broadcast の通信アルゴリズムとして**図 7** のように $\log_2 p$（p はプロセス数を示す）ステップで通信が完了するアルゴリズムを用いる．図 7 では，8 つのプロセスが 2 つのサブクラスタに属している．これらのサブクラスタ間では TCA による通信ができないため，Step 1 では MPI を用いて Root プロセスである Rank0 から Rank4 にデータを転送する．データを受け取ったプロセスは，以降のステッ

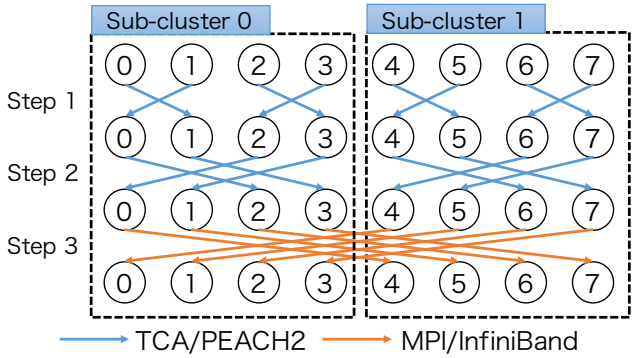

図 8 Allgather 通信のアルゴリズム：Recursive doubling

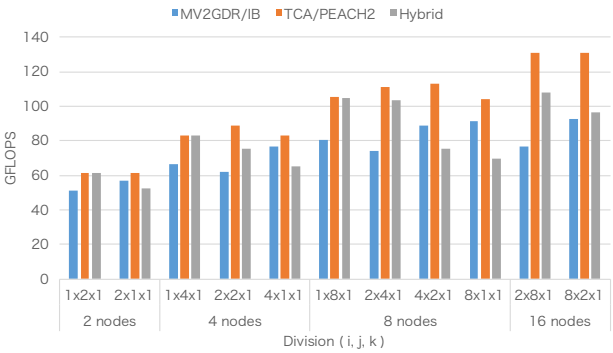

図 9 姫野ベンチマーク：データサイズ Small

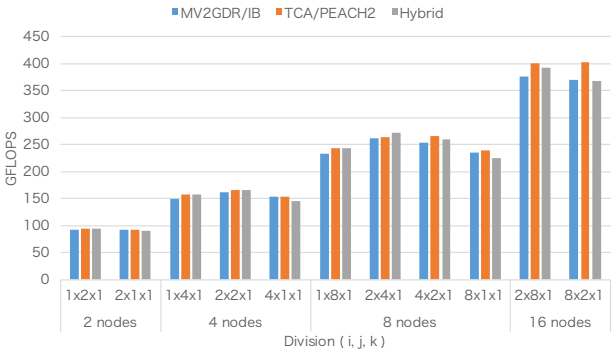

図 10 姫野ベンチマーク：データサイズ Middle

プでは Root プロセスのように他のプロセスにデータを転送する．Step 2 以降は TCA による通信が可能な領域となり，本実装では Chained DMA 機構を用いて，DMA 通信を 1 回だけ発行することで通信が完了するようにしている．

5.5 ハイブリッド通信による Allgather 通信

本節では，Allgather 通信に対してサブクラスタを跨いだハイブリッド通信を適用することについて述べる．Allgather にはいくつかの通信アルゴリズムが存在するが，本稿では通信路のバンド幅を最大限利用するために「Recursive Doubling」アルゴリズムを用いる．**図 8** に Allgather の通信アルゴリズムを示す．これより，自ノードとデータを交換するプロセスとの距離はステップが進むごとに倍になっていく．同様に，Recursive Doubling アルゴリズムでは，交換されるデータのサイズもステップが進むごとに倍になっていく．つまり，最初の数ステップはデータサイズが小さく，低レイテンシ通信が必要となり，後半のステップではデータサイズ非常に大きくなるため，高いバンド幅が必要になる．そこで本稿における実装では，ハイブリッド通信の性能を最大限に発揮するために，前半のステップは TCA による低レイテンシ通信，最終ステップのみ InfiniBand による高バンド幅通信を割り当てることとする．最終的には，InfiniBand によるサブクラスタを跨いだ通信によって，全てサブクラスタ内のプロセスとデータを交換することが出来る．

6. 性能評価

本章では，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（TCA/PEACH2），MV2GDR のみ（MPI/InfiniBand），ハイブリッド通信で行い，実行性能を比較する．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．姫野ベンチマークの XACC によるソースコードは文献 [10] 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[23] 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み合わせとする．集団通信の Broadcast および Allgather については，現在 XACC に対してハイブリッド通信を適用する実装を行っているため，本稿の評価では Hand-coding で作成したプログラムを用いて MV2GDR に対するハイブリッド通信の性能評価を行う．評価には 4 章と同様に HA-PACS/TCA を用いる．本稿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，サブクラスタを跨いだ通信を再現する．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想定で最大 16 ノードまでの実験を行う．

6.1 姫野ベンチマークの評価

本評価の問題サイズは Small（64 × 64 × 128），Middle（128 × 128 × 256）とする．分割（i, j, k）は図 5 に対応している．本評価には，i 方向と j 方向を分割し，k 方向は分割せず常に 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．i 方向の MV2GDR の通信がサブクラスタ間の通信に当たる．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

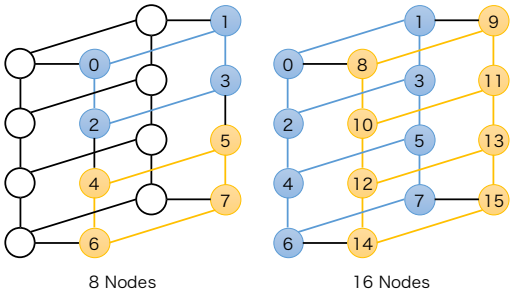

図 11 サブクラスタの分割とプロセスマッピング．数字は Rank 番号，各色のグループは仮想的に分割したサブクラスタを示している．仮想的に分割したサブクラスタ間では TCA 通信を一切行わない．

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，**MPI_Allreduce()** を使ってデータを更新する．そのため，袖領域交換の性能だけが全体の性能に影響することになる．

図 9，**図 10** にそれぞれ問題サイズ Small，Middle の実行性能を示す．図 10 の問題サイズ Middle では，通信データサイズはたかだか 256KB であり，TCA が有効なデータの範囲である．このため，TCA を用いた時の性能が 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, 8, 1）において MV2GDR よりも高速であり，分割（2, 4, 1）においては 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，性能向上が得られたと考えられる．図 9 の問題サイズ Small は，通信サイズが最大で 64KB であり，分割（2, 8, 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た．

6.2 Broadcast の評価

本節では，ハイブリッド通信を適用した Broadcast 通信について MV2GDR の **MPI_Bcast()** 関数との性能比較を行う．評価には 8 ノードおよび 16 ノードを用いて，**図 11** のように 2 つのサブクラスタに分割した環境で評価を行う．Broadcast では，TCA および MPI の通信は常に Block 転送となり，各通信ステップにおける通信データサイズは常に一定である．Root プロセスは共に Rank0 とする．

図 12 および**図 13** に，それぞれ 8 ノード，16 ノードにおける Broadcast の通信時間と，MV2GDR に対するハイブリッド通信の相対性能を示す．グラフでは通信時間

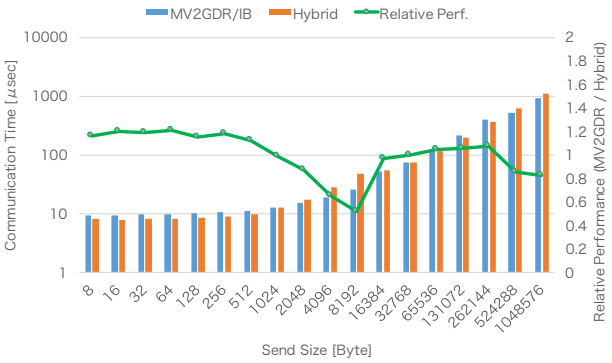

図 12 Broadcast：8 ノード

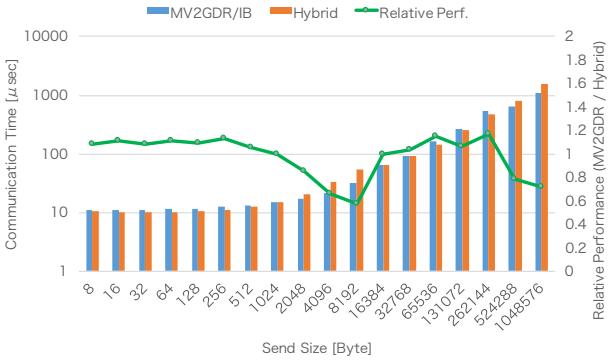

図 13 Broadcast：16 ノード

の差が小さいように見えるが，左辺の縦軸が log スケールで表示されているためで，実際の通信時間の差は見た目以上に大きい．そこで，MV2GDR 通信に対する Hybrid 通信の相対性能を折れ線で表現し，グラフの右辺の縦軸に値を示す．相対性能が 1.0 より大きい場合は，ハイブリッド通信が MV2GDR に対して高速であることを示している．図 12 より，8 ノードの時は，MV2GDR に対して最大 21%の性能向上が得られている．しかしながら，$2KB \sim 8KB$ のデータサイズにおいて，相対性能が 1.0 より低くなっている．これは 4 節で述べた MV2GDR のプロトコルスイッチが関係していると考えられる．ハイブリッド通信における MV2GDR の通信は，8KB までは GDR 機能を用いた InfiniBand 経由による GPU 間直接通信を行い，それ以上のデータサイズではホスト経由の通信に切り替えを行っている．このため，図 4 より，特に 8KB において「MV2GDR-QPI-Tuned」の性能は「MV2GDR」の性能に対して非常に低いことが分かる．同様に，MV2GDR の **MPI_Bcast()** 関数内でも異なるプロトコルスイッチが起きている可能性がある．このために，$2KB \sim 8KB$ においてハイブリッド通信の相対性能が低下したと考えられる．図 12 のデータサイズが 512KB より大きい時は，TCA と MV2GDR の性能分岐点を超えてしまい，MV2GDR に対する TCA の優位性が無いため相対性能が 1.0 を下回っている．

図 13 の 16 ノードの結果は，8 ノードの結果と同様な傾向となり，MV2GDR に対して最大 14%の速度向上が得られている．しかしながら，8 ノードの Broadcast に対して，16 ノードで得られる速度向上率が小さい．この原因として，TCA 通信のホップ数の増加が影響していると考えられる．16 ノードのハイブリッド通信を用いた Broadcast は 4 ステップ（$= \log_2 16$）の通信で構成されており，Step 1 ～ 3 は TCA の通信，Step 4 は MV2GDR の通信となる．Step 1 ～ 2 の TCA 通信は隣接ノードへの DMA 転送となるが，Step 3 の TCA 通信は 2 つ先にノードへの DMA 転送となる．そのため，1 ホップにつき 0.3 ～ 0.4μsec 程度 TCA のレイテンシが増加し，特にデータサイズが小さい時に性能低下を招く可能性がある．そこで，サブクラスタ間だけではなくサブクラスタ内で TCA のホップ数が増加する通信を MV2GDR に割り当てることによってより性能が向上する可能性がある．16 ノードよりも大きい環境ではよりホップ数が増加するため，このような通信の割り当ての最適化をする必要があると考えられるため，今後の課題とする．

6.3 Allgather の評価

本節では，ハイブリッド通信を適用した Allgather 通信について MV2GDR の **MPI_Allgather()** 関数との性能比較を行う．Broadcast と同様に 8 ノードおよび 16 ノードを用いて，図 11 のように 2 つのサブクラスタに分割することを想定する．Allgather では通信ステップが増えるたびに通信データサイズは倍になり，最終的に全ノードで共有するデータサイズは「Date Size × Number of Process」となる．また，PEACH2 のハードウェアの制限により，ハイブリッド通信ではスカラー値の Allgather を実行できないため，Double Precision 2 要素（16 Byte）の Allgather から評価を行う．**図 14** および**図 15** に，それぞれ 8 ノードおよび 16 ノードにおける Allgather の通信時間と，MV2GDR に対するハイブリッド通信の相対性能を示す．グラフは，Broadcast と同様に左辺の縦軸が log スケールの通信時間，右辺の縦軸が相対性能を示している．相対性能は 1.0 より大きい場合は，ハイブリッド通信が MV2GDR に対して高速であることを示している．図 14 より，ハイブリッド通信は $1KB \sim 4KB$ を除き 32KB まで MV2GDR に対して高い性能を示しており，最大で 46%の性能向上が得られた．同様に図 15 より，ハイブリッド通信は $512B \sim 1KB$ を除き 16KB まで MV2GDR に対して高い性能を示し，最大で 40%の性能向上が得られた．$1KB \sim 4KB$ または $512B \sim 1KB$ で一時的に性能が低下する原因は，Broadcast と同様に MV2GDR のプロトコルスイッチが関係していると考えている．8 ノードおよび 16 ノードにおいて，最終的に MV2GDR に対するハイブリッド通信の相対性能が 1.0 より小さくなるデータサイズはそれぞれ 64KB，32KB である．このとき，最終的に全プロセスで共有されるデータサイズはともに 512KB（$= 64KB \times 8nodes$，$32KB \times 16nodes$）となり，MV2GDR に対して TCA に優位性があるデータサイズを超えてしまうため，相対性能が 1.0 より小さくなったと考えられる．

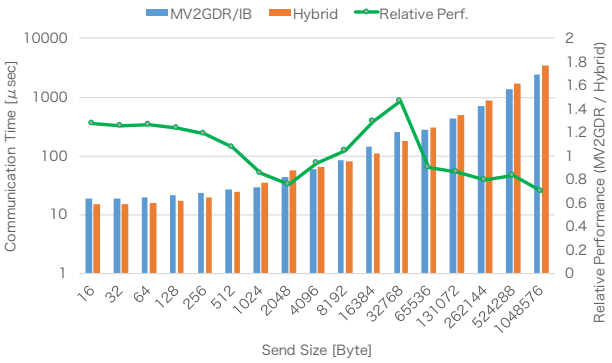

図 14 Allgather：8 ノード

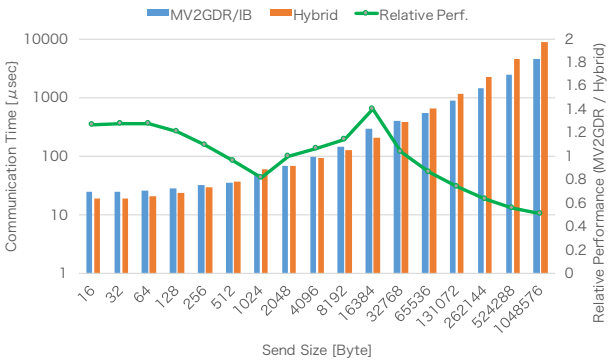

図 15 Allgather：16 ノード

6.4 ハイブリッド通信についての考察

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跨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 のみに頼らざるを得ないが，姫野ベンチマークの評価の結果，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．一方で，集団通信である Broadcast と Allgather にハイブリッド通信を適用したところ，最大で 46%の性能向上を達成することができた．これらの評価から TCA+InfiniBand ハイブリッド通信について要約すると，2 つの利点があることがわかった．第一に，通信の特性に合わせて異なるネットワークの通信を同時に実行することで，全体の通信性能が向上する．第二に，データサイズが小さい時に有効な TCA の低レイテンシ通信と，データサイズが大きい時に有効な高バンド幅の InfiniBand 通信を，集団通信のようにデータサイズに応じて異なる通信ステップで適宜選択することで，全体の通

信性能が非常に高くなる．前者は，2 つのネットワークによって全体のバンド幅が向上することによって実現し，後者は，InfiniBand ネットワークの中に局所的な TCA ネットワークが存在する階層的なネットワークによって実現できる．

また，本研究では XACC コンパイラ及びランタイムライブラリにハイブリッド通信を適用することにより，煩雑になりがちな袖領域交換のハイブリッド通信を同言語の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することが可能であることがわかった．

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 [24]．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．また，InfiniBand を用いた通信では，Block Stride 転送のような細粒度の転送を何度も行う必要がある場合，通信の効率を上げるために Pack/Unpack が必要である．一方，PEACH2 における通信ではハードウェア Block Stride 機能を用いることで，Pack/Unpack を行うことなく高速に転送することが可能である．

NVIDIA 社が提供する NVLink[25] 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よい GPU 間直接通信が可能になると言われている．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．

8. 結論

本研究では，TCA/PEACH2 におけるサブクラスタを跨いだ通信を実現するために，TCA/PEACH2 と InfiniBand によるハイブリッド通信を提案し，実装を行った．そして，多次元ステンシル計算における袖領域交換の通信や Broadcast，Allgather といった集団通信に対してハイブリッド通信を適用し評価を行った．その結果，InfiniBand を用いた MVAPICH2-GDR に対して，姫野ベンチマークでは最大 40%，Broadcast では 21%，Allgather では 46%の性能向上を達成した．また，アクセラレータ向け並列言語である XcalableACC のコンパイラ及びランタイムライブラリに対して，ハイブリッド通信を適用する実装を行い，ハイブリッド通信における複雑なプログラミングのコストを軽減することができた．

今後の課題として，袖領域交換の通信や集団通信に対して実際に複数のサブクラスタを跨いだ 16 ノードより大きい環境での評価を行う．また，今回実装できていない集団通信を XcalableACC に適用することや，通信の最適化を行う．さらに，実際のアプリケーションに対して，ハイブリッド通信を適用した XcalableACC による記述と性能評価を行う．

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://top500.org/.
[2] HPCI 技術ロードマップ白書 2012 年 3 月. 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, Oct. 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, MikeClark. 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, Jul. 2014.
[5] N. Fujita, H. Fujii, T. Hanawa, Y. Kodama, T. Boku, Y. Kuramashi, and M. Clark. QCD Library for GPU Cluster with Proprietary Interconnect for GPU Direct Communication. In *12th The 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,

Feb. 2015.

[7] J. Lee and M. Sato. Implementation and Performance Evaluation of XcalableMP: A Parallel Programming Language for Distributed Memory Systems. In *Third International Workshop on Parallel Programming Models and Systems Software for High-End Computing (P2S2)*, pp. 413–420, Sep. 2010.

[8] M. Nakao, J. Lee, T. Boku, and M. Sato. Productivity and Performance of Global-View Programming with XcalableMP PGAS Language. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2012)*, CCGRID '12, pp. 402–409, 2012.

[9] OpenACC. `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, Jul. 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, Sep. 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, Jan. 2014.

[13] T. Hanawa, Y. Kodama, T. Boku, and M. Sato. Interconnection Network for Tightly Coupled Accelerators Architecture. In *IEEE 21st Annual Symposium on High-Performance Interconnects (HOT Interconnects 21)*, pp. 79–82, Aug. 2013.

[14] T. Hanawa, Y. Kodama, T. Boku, and M. Sato. Tightly Coupled Accelerators Architecture for Minimizing Communication Latency among Accelerators. In *The Third 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] Altera Corporation. Stratix IV Device Handbook. `http://www.altera.co.jp/literature/lit-stratix-iv.jsp`.

[16] 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, Jul. 2011.

[17] NVIDIA Corporation. NVIDA GPUDirect. `https://developer.nvidia.com/gpudirect`.

[18] MVAPICH2: A High Performance MPI Library for NVIDIA GPU Clusters with InfiniBand. `http://on-demand.gputechconf.com/gtc/2013/presentations/S3316-MVAPICH2-High-Performance-MPI-Library.pdf`.

[19] MVAPICH2-GDR 2.0 README. `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, Dec. 2014.

[21] K. Matsumoto, T. Hanawa, Y. Kodama, H. Fujii, and T. Boku. Implemention of CG method on GPU cluster with proprietary interconnect TCA for GPU direct communication. In *The Fifth International Workshop on Accelerators and Hybrid Exascale Systems (AsHES 2015) in conjunction with 2015 IEEE International Parallel and Distributed Processing Symposium (IPDPS 2015)*, pp. 647–655, May 2015.

[22] 小田嶋哲哉, 塙敏博, 児玉祐悦, 朴泰祐, 村井均, 中尾昌広, 佐藤三久. HA-PACS/TCA における TCA および InfiniBand ハイブリッド通信. Vol. 2014-HPC-147, No. 32, pp. 1–8, Dec. 2014.

[23] A. Tabuchi, M. Nakao, and M. Sato. A Source-to-Source OpenACC Compiler for CUDA. In *11th The 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.

[24] Mellanox Technologies: Mellanox OFED GPUDirect. `http://www.mellanox.com/page/products_dyn?product_family=116`.

[25] NVIDIA NVLINK HIGH-SPEED INTERCONNECT. `http://www.nvidia.com/object/nvlink.html`.