Vol.2018-HPC-164 No.9
2018/5/7

Tightly Coupled Accelerators/InfiniBandハイブリッド通信を用いたアクセラレータクラスタ用並列言語XcalableACCの評価

中尾 昌広[1,a)]　小田嶋 哲哉[1]　村井 均[1]　田渕 晶大[2]　藤田 典久[3]　塙 敏博[4]　朴 泰祐[3,5]
佐藤 三久[1]

概要：アクセラレータを搭載したクラスタシステム（アクセラレータクラスタ）の性能を引き出すためには，アクセラレータ間の通信レイテンシを小さくすることが重要である．また，アクセラレータクラスタを簡易に利用できるプログラミング言語も求められている．本稿では，Tightly Coupled Accelerators（TCA）/InfiniBandハイブリット通信を用いたアクセラレータクラスタ用並列言語XcalableACC（XACC）の評価を行う．TCA/InfiniBandハイブリット通信とは，TCAが持つ低レイテンシ通信とInfiniBandが持つ高バンド幅を組合せた通信である．TCA/InfiniBandハイブリット通信およびXACCの有用性を調べるため，HPC分野で重要なアプリケーションの1つであるLattice Quantum Chromo-Dynamics（LQCD）の実装を行い，64計算ノードのアクセラレータクラスタ上で評価を行った．また，XACCとの比較を行うために，CUDAとMPI（CUDA+MPI）およびOpenACCとMPI（OpenACC+MPI）を用いたLQCDも実装した．まず性能評価を行った結果，TCA/InfiniBandハイブリッド通信を用いたXACCの性能は，CUDA+MPIの性能よりも9%高く，OpenACC+MPIの性能よりも18%高いことがわかった．さらに，XACCに対して新しい拡張を行うことで，XACCの性能はさらに7%高くなることがわかった．次に生産性評価を行った結果，XACCはCUDA+MPIおよびOpenACC+MPIと比較して少ないコード量で実装を行えることがわかった．また，XACCは逐次コードのイメージを保ったまま並列化を行えるため可読性が高く，さらにポータビリティにも優れていることを示した．

Evaluation of XcalableACC using Hybrid Communication with Tightly Coupled Accelerators/InfiniBand on Accelerated Clusters

Masahiro Nakao[1,a)]　Tetsuya Odajima[1]　Hitoshi Murai[1]　Akihiro Tabuchi[2]　Norihisa Fujita[3]
Toshihiro Hanawa[4]　Taisuke Boku[3,5]　Mitsuhisa Sato[1]

1. はじめに

優れた電力性能比とメモリバンド幅を持つアクセラレータを搭載したクラスタシステム（アクセラレータクラスタ）が計算資源として広く利用されている．2017年11月のTop500リスト[1]とGreen500リスト[2]の上位にランクインしているシステムの多くは，NVIDIA社のGPU，Intel社のXeon Phi，PEZY社のPEZY-SC2などのアクセラレータを利用している．

1 理化学研究所 計算科学研究センター
RIKEN Center for Computational Science
2 富士通研究所
FUJITSU LABORATORIES LTD
3 筑波大学 計算科学研究センター
Center for Computational Sciences, University of Tsukuba
4 東京大学 情報基盤センター
Information Technology Center, The University of Tokyo
5 筑波大学 大学院 システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
a) masahiro.nakao@riken.jp

アクセラレータクラスタ上で計算を行う場合，計算ノードを跨ぐアクセラレータ間のデータ通信速度が重要になる．そのため，例えば Mellanox 社の InfiniBand と NVIDIA 社の GPU を搭載したアクセラレータクラスタでは，ホストメモリを経由せずに GPU 間の高速通信を実現する MVAPICH2-GDR [3] がよく用いられる．しかしながら，近年は強スケーリングにおけるアプリケーション性能が求められているため，通信レイテンシをより小さくすることが重要な課題である [4]．

また，アクセラレータクラスタにおけるプログラミングについても課題がある．例えば NVIDIA 社の GPU を搭載したアクセラレータクラスタでは，CUDA と MPI を用いたプログラミング（CUDA+MPI）が一般的である．しかしながら，CUDA は NVIDIA 社の GPU のみの対応であり，MPI はプリミティブな通信関数しか提供していない．そのため，CUDA+MPI は，システムの性能を引き出せる反面，生産性が低いという問題点がある．生産性を改善させるため，CUDA+MPI の代わりに OpenACC と MPI を用いたプログラミング（OpenACC+MPI）が採用される場合もあるが [5,6]，MPI を原因とするプログラミングの煩雑さは解消されない．

これらの背景から，我々はより少ない通信レイテンシでアクセラレータ間のデータ転送を行うため，密結合並列演算加速機構 Tightly Coupled Accelerators（TCA）を提案している [7,8]．さらに，TCA が持つ低レイテンシ通信と InfiniBand が持つ高バンド幅を組合せた通信である TCA/InfiniBand ハイブリッド通信も提案している [9,10]．また，我々はアクセラレータクラスタにおける生産性の向上のため，指示文ベースの並列言語である XcalableACC（XACC）を開発している [9,11–13]．XACC が提供する指示文から TCA/InfiniBand ハイブリッド通信が利用可能であるため，ユーザは高性能なアプリケーションを少ないコストで開発可能である [9]．

前述した XACC による TCA/InfiniBand ハイブリッド通信の研究 [9] は，最大 16 計算ノードの計算環境で行った．大きな問題に対応するためには，より大規模な計算環境における性能特性を明らかにする必要がある．そこで，本稿では 64 計算ノードで構成されたアクセラレータクラスタを用いて，TCA/InfiniBand ハイブリッド通信を用いた XACC アプリケーションの性能について考察する．対象アプリケーションとして，HPC 分野で重要なアプリケーションの 1 つである Lattice Quantum Chromo-Dynamics（LQCD）を用いる．また，性能評価とともに，XACC が持つ生産性についても評価を行う．XACC と比較するために，既存のプログラミングモデルである CUDA+MPI および OpenACC+MPI を用いる．

本稿の貢献は下記の通りである．(1) XACC を用いて LQCD を開発し，64 計算ノードのアクセラレータクラスタ上で TCA/InfiniBand ハイブリッド通信を用いた性能評価を行う．(2) XACC だけでなく，CUDA+MPI および OpenACC+MPI を用いて LQCD を開発し，性能および生産性について XACC との比較を行う．

本稿の構成は下記の通りである．2 章で TCA と TCA/InfiniBand ハイブリッド通信について述べ，3 章でそれらの性能評価を行う．4 章で XACC による LQCD の実装について述べ，5 章でその性能評価を行う．6 章で関連研究について述べ，7 章で本稿をまとめる．

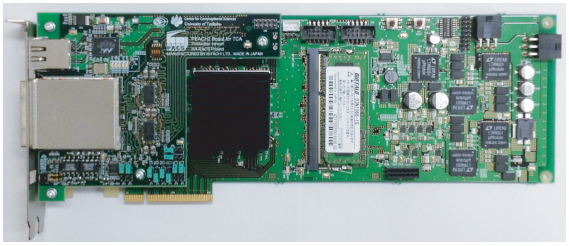

図 1: PEACH2 の画像

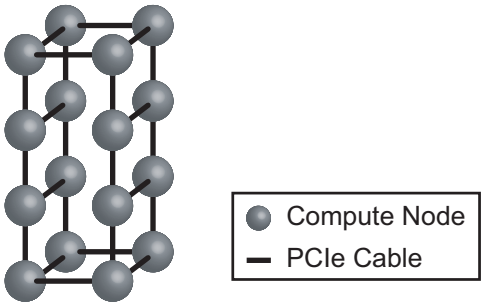

図 2: 16 計算ノードで構成されるサブクラスタ

2. Tightly Coupled Accelerators

2.1 概要

計算ノードを跨ぐアクセラレータ間のデータ通信を少ないレイテンシで行うために，我々は TCA に基づくシステムを開発している．TCA は Peripheral Component Interconnect Express（PCIe）を計算ノード間の通信プロトコルとして用いるため，複数の計算ノードに搭載されているアクセラレータは，同一の PCIe ネットワークに接続されているように扱うことができる．従来の MPI と InfiniBand を利用したデータ通信では，MPI ソフトウェアスタックや PCIe-InfiniBand 間のプロトコル変換が必要であったが，TCA を用いたシステムではそれらは不要であるため，少ないレイテンシでデータ通信が可能になる．

TCA の実装の 1 つに PCIe Adaptive Communication Hub ver.2（PEACH2）[14,15] がある．PEACH2 は Field-Programmable Gate Array（FPGA）である Altera Stratix IV GX [16] で実装されており，計算ノードの PCIe スロットに接続して利用する．PEACH2 の画像を**図 1** に示す．PEACH2 には高性能な DMA コントローラを搭載しているため，高速な Direct Memory Access（DMA）やブロック

表 1: HA-PACS/TCA システムの仕様

CPU	Intel Xeon-E5 2680v2 2.8 GHz x 2 Sockets
Memory	DDR3 1866 MHz x 4 channel, 128GB
GPU	NVIDIA Tesla K20X x 4 GPUs, GDDR5 6GB
Network	InfiniBand FDR 7GB/s
Software	Intel compiler 16.0.4, CUDA 7.5.18 MVAPICH2-GDR 2.2

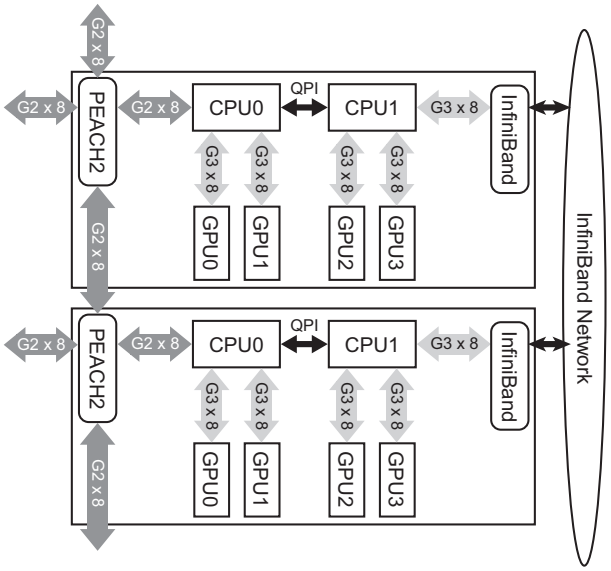

図 3: HA-PACS/TCA の計算ノードの構成

ストライド通信が可能である．PEACH2 は，4 つの PCIe Gen2 × 8 ポートを持ち，1 つはホストと接続し，残り 3 つは隣接ノードの PEACH2 と PCIe ケーブル [17] を介して接続する．PEACH2 のみで大規模なクラスタを構成することは，PCIe ケーブル長の限界やホップ数の増加に伴う性能低下により困難であるため，**図 2** のように 16 計算ノードで構成される 2 × 8 の 2 重リングトポロジとして 1 つのグループを構成して運用を行うことを想定している．本稿では，この 16 計算ノードで構成されたグループを “サブクラスタ” と呼称する．

2.2 HA-PACS/TCA システム

TCA のコンセプトを実証するためのシステムとして，64 計算ノードで構成された GPU クラスタ HA-PACS/TCA が筑波大学計算科学研究センターで運用されていた（2018 年 3 月末に運用停止）．HA-PACS/TCA の仕様を**表 1** に，計算ノードの構成を**図 3** に示す．各計算ノードは，2 ソケットの CPU と 4 枚の GPU を搭載している．CPU0 側には PEACH2 が PCIe Gen2 × 8 ポートで接続され，CPU1 側には InfiniBand が PCIe Gen3 × 8 ポートで接続されている．なお，PEACH2 はすべての GPU にアクセス可能であるが，Intel QuickPath Interconnect（QPI）を経由する通信は性能が低下するため，PEACH2 が GPU2 と GPU3 にアクセスすることは想定していない．図 3 では，GPU と 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 と接続されているため，実際の通信は CPU を介さずに行われる．InfiniBand は HA-PACS/TCA の全 64 計算ノードを単一スイッチでフラットに接続しているが，PEACH2 によるネットワーク構成では 4 つのサブクラスタに分かれている．

```
1  double *send_buf, *recv_buf;
2  size_t byte = ...;
3  tcaMalloc(&send_buf, byte, tcaMemoryGPU);
4  tcaMalloc(&recv_buf, byte, tcaMemoryGPU);
5  tcaHandle *send_handle, *recv_handle;
6  tcaCreateHandleList(&send_handle, 2, send_buf, byte);
7  tcaCreateHandleList(&recv_handle, 2, recv_buf, byte);
8  tcaDesc *desc = tcaDescNew();
9  int target = (my_rank == 0)? 1 : 0;
10 off_t send_offset = 0, recv_offset = 0;
11 tcaDescSetMemcpy(desc, &recv_handle[target], recv_offset,
       &send_handle[my_rank], send_offset, byte, ...);
12 int DMAC_CH = 0;
13 tcaDescSet(desc, DMAC_CH);
14
15 if(my_rank == 0){
16   tcaStartDMADesc(DMAC_CH);
17   tcaWaitDMARecvDesc(&recv_handle[target], ...);
18 }
19 else{
20   tcaWaitDMARecvDesc(&recv_handle[target], ...);
21   tcaStartDMADesc(DMAC_CH);
22 }
```

図 4: PEACH2 のプログラミング例

2.3 PEACH2 のプログラミング

PEACH2 を用いて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．PCIe アドレスは一般的なポインタとは型が異なるため，PEACH2 を用いたプログラミングでは *tcaHandle* 型のハンドルを定義し，PCIe アドレスの管理を行っている．PEACH2 においてリモートノードにデータを送信する手順は下記の通りである．(1) 通信用のハンドルとディスクリプタを作成する．(2) ハンドルとディスクリプタに対して読み込み元および書き込み先のアドレスや通信サイズなどを指定する．(3) ディスクリプタと DMA コントローラとを関連付ける．(4) DMA コントローラを起動する．なお，連続領域に対するデータ通信だけでなく，ブロックストライド通信も行うことができる．PEACH2 では Chained DMA という機能を用いることで，ブロックストライド通信を高速に処理できる．

2 プロセス間で pingpong 通信を行う PEACH2 のプログラミング例を**図 4** に示す．1〜4 行目では，アクセラレータ上に送信用領域と受信用領域を確保している．5〜7 行目では，2 プロセス分の送受信用のハンドルを生成している．8〜11 行目では，ディスクリプタを作成し，読み込み元お

よび書き込み先のアドレスや通信サイズなどをそのディスクリプタに設定している．12〜13 行目では，PEACH2 の DMA コントローラ（PEACH2 には DMA コントローラは 4 つあり，0〜3 の番号で指定する）とディスクリプタの関連付けを行っている．15〜22 行目では，プロセス間で pingpong 通信を行っており，tcaStartDMADesc() で送信を実行し，tcaWaitDMARecvDesc() で受信が完了するまで待機している．なお，11 行目で用いている tcaDescSetMemcpy() は連続データを登録するための関数であり，ブロックストライドデータを登録するには，同じディスクリプタを用いて tcaDescSetMemcpy() を複数回実行すればよい．

2.4 TCA/InfiniBand ハイブリッド通信

TCA は低レイテンシ通信を実現するが，下記の問題点がある．(1) HA-PACS/TCA で用いている InfiniBand FDR の最大バンドは 7GB/s であるのに対し，PEACH2 の最大バンド幅は 4GB/s である．(2) PEACH2 同士の通信は，16 ノードで構成されるサブクラスタ内でしか行えない．

上記の問題点を克服するため，我々は高いバンド幅とスケーラビリティを持つ InfiniBand ネットワークと，サブクラスタ内において低レイテンシ通信を実現する TCA とを組合せた TCA/InfiniBand ハイブリッド通信を提案している [9,10]．TCA/InfiniBand ハイブリッド通信は，データの種類に応じて TCA と InfiniBand がそれぞれ得意とする通信を行うことにより，全体の通信性能を向上させることを目的としている．具体的には，サブクラスタ内かつデータサイズが小さな通信やブロックストライド通信については TCA を用い，サブクラスタ間またはデータサイズが大きい通信には MPI を介して InfiniBand を用いる．この 2 つの通信は同時に利用することも可能である．

3. TCA 通信と TCA/InfiniBand ハイブリッド通信の性能評価

3.1 概要

本章では，TCA 通信および TCA/InfiniBand ハイブリッド通信の性能を HA-PACS/TCA 上で評価する．また，比較のために InfiniBand の性能も評価する．性能評価の内容として，単純な連続データ通信だけでなく，LQCD など多くのアプリケーションで現れる通信パターンである多次元配列に対する袖領域通信（ブロックストライドのデータ通信）も測定する．なお，過去の研究 [9] において，本章で行ういくつかの性能評価を HA-PACS/TCA 上で行っているが，それらの研究以降に HA-PACS/TCA のハードウェアの一部が更新されたため，本稿で改めて性能を評価する．

2.2 節で述べたように，QPI を経由した GPU 間通信は性能が低下してしまう問題がある．そのため，PEACH2 の測定には図 3 の GPU0 同士，InfiniBand の測定には GPU2 同士を用いる．ただし，TCA/InfiniBand ハイブリッド通信では，TCA の性能を有効に利用するため，GPU0 同士を用いる．この場合，QPI による性能低下は InfiniBand に対して発生する．この性能低下も評価するため，InfiniBand については QPI を跨ぐ通信についても測定する．各計算ノードには 1 プロセスずつを割り当て，各プロセスは 1 つの GPU のみを操作する．TCA 通信および TCA/InfiniBand ハイブリッド通信の測定では，サブクラスタ内の隣接計算ノードになるようにプロセスを配置する．

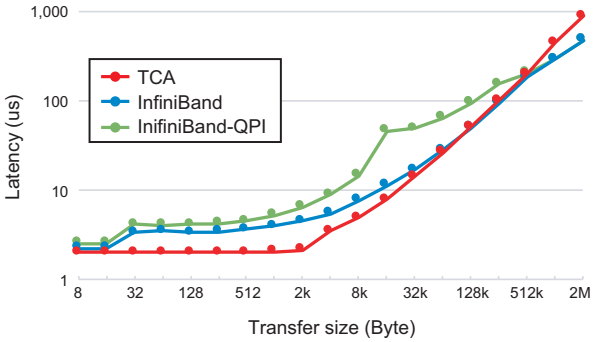

図 5: 連続データの通信性能

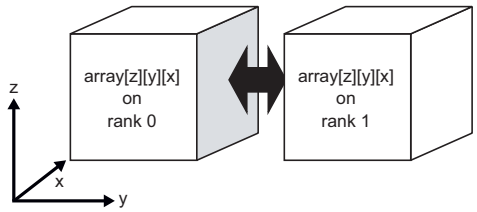

図 6: ブロックストライド通信のデータパターン

3.2 連続データの通信性能

連続データに対する pingpong 通信の性能を**図 5** に示す．図中の “TCA” は PEACH2 による GPU0 同士の通信，“InfiniBand” は InfiniBand による GPU2 同士の通信，“InfiniBand-QPI” は InfiniBand による QPI を跨いだ GPU0 同士の通信を示す．なお，InfiniBand を用いた通信については事前性能評価を行い，MVAPICH2-GDR のパラメータをチューニングしている．この事前性能評価については A.1 節で述べる．

図 5 より，‘TCA” は “InfiniBand” に対して 128kB まで高い性能を示すが，128kB 以降については “InfiniBand” の方が高い性能を示す．この性能差の理由は，各インターコネクトの最大バンド幅の違いが原因と考えられる．また，“InfiniBand” と “InfiniBand-QPI” を比較すると，512kB よりも小さい転送サイズの場合は，“InfiniBand” の方が高い性能を示す．512kB 以上の転送サイズの場合は，“InfiniBand” と “InfiniBand-QPI” の性能はほぼ同じである．

3.3 ブロックストライドデータの通信性能

3 次元配列に対する袖領域通信を想定したブロックスト

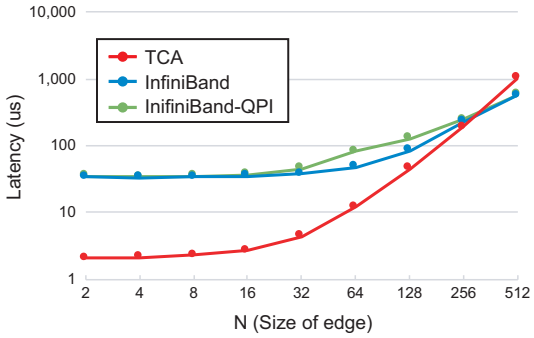

図 7: ブロックストライドデータの通信性能

ライドデータの通信の性能評価を行う．通信データのパターンを**図 6** に示す．図 6 では，C 言語における次元の並びの 3 次元配列の xz 平面を隣接ノード間で交換している．この xz 平面の通信パターンは，3 次元配列の各次元の要素数を N とすると，N 要素の連続データ転送が $N \times N$ 周期で現れるブロックストライドである．

PEACH2 を用いてブロックストライド通信を行う場合，2.1 節で述べた通り，PEACH2 が持つ DMA コントローラで行うことができる．MPI を用いてブロックストライド通信を行う場合，ユーザが任意の MPI_Datatype を定義して通信を行うことが一般的であるが，事前性能評価を行った結果，その方法は性能が低いことがわかった．そこで，MPI の性能は下記のように CUDA を用いて Packing/Unpacking する方法で評価する．(1) GPU メモリ上にバッファを確保する．(2) 3 次元配列の送信領域を CUDA を用いてバッファに Packing する．(3) バッファ上の連続データを相手に転送する．(4) 相手先において，受け取ったデータを 3 次元配列の受信領域に CUDA を用いて Unpacking する．MPI_Datatype を用いる方法と CUDA を用いて Packing/Unpacking する方法との性能比較は A.2 節で述べる．

1 要素が 8 バイトであるサイズ N^3 の 3 次元配列におけるブロックストライドデータの通信性能を**図 7** に示す．この結果より，$N \leq 256$ の場合，TCA は InfiniBand よりも高い性能を示すことがわかる．なお，$N = 2$ から 8 の TCA の性能は，図 5 で示した連続データの通信性能とほぼ同じである．InfiniBand の性能は，Packing/Unpacking を行う必要があるため，連続データの通信性能と比べても性能は低い．QPI を跨ぐ InfiniBand の性能は，$N = 32$ から 256 において，QPI を跨がない InfiniBand よりも性能は低い．

3.4 TCA/InfiniBand ハイブリッド通信を用いた袖通信の通信性能

2.4 節で述べた TCA/InfiniBand ハイブリッド通信の性能評価を行う．性能評価で用いる通信データのパターンを**図 8** に示す．図 8 では，rank 0 は rank 1 と rank 3 に xy 平面を，rank 2 と rank 4 に xz 平面を交換している．xy 平面は連続データであり，xz 平面はブロックストライドデータである．TCA/InfiniBand ハイブリッド通信では，この 2 種類のデータパターンの内，連続データについては QPI を跨ぐ InfiniBand を用い，ブロックストライドデータについては QPI を跨がない PEACH2 を用いる．

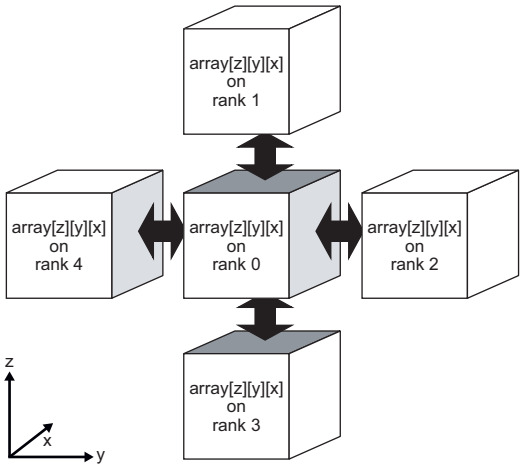

図 8: TCA/InfiniBand ハイブリッド通信のデータパターン

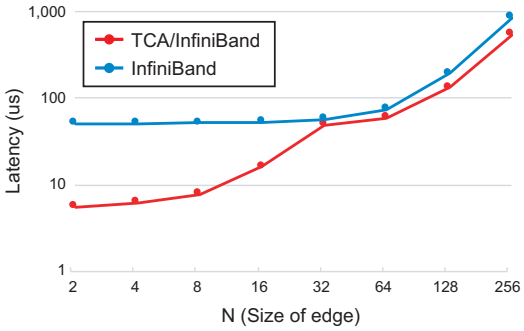

図 9: TCA/InfiniBand ハイブリッド通信の性能

1 要素が 8 バイトであるサイズ N^3 の 3 次元配列におけるハイブリッド通信性能を**図 9** に示す．QPI を跨ぐ InfiniBand の通信は，5 章で述べる LQCD の性能評価では用いないため，計測からは除外した．この結果より，TCA/InfiniBand ハイブリッド通信は InfiniBand のみよりも常に性能が高いことがわかる．この理由は，TCA/InfiniBand ハイブリッド通信は PEACH2 と InfiniBand の 2 本の通信路を同時に用いており，さらに InfiniBand が不得手とするブロックストライド通信を PEACH2 が行っているからである．

4. XcalableACC を用いた LQCD の実装

4.1 XcalableACC とは

XACC は指示文ベースの並列言語 XcalableMP（XMP）[18–20] のアクセラレータ拡張であり，XMP と OpenACC との相互運用を可能にしたプログラミングモデルである．XACC は C 言語および Fortran の拡張として定義されて

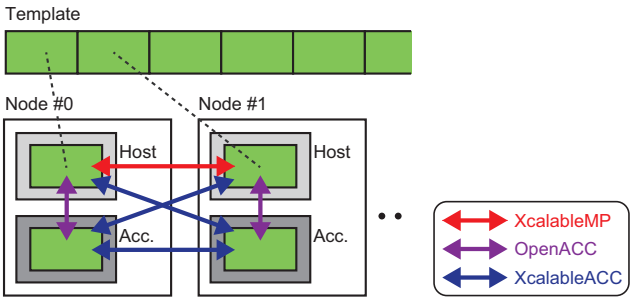

図 10: XcalableACC のメモリモデル

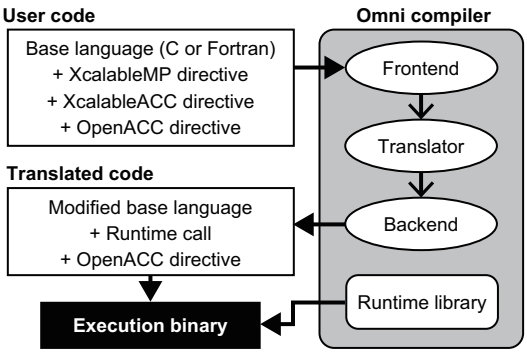

図 11: Omni Compiler のコンパイルの流れ

いる．XMP 指示文は，ループ文の分割，分散配列の定義や通信などをホストに対して行う．OpenACC 指示文は，XMP 指示文による処理をアクセラレータに対して実行する．すなわち，XACC は XMP が提供する分散メモリ並列処理機能を用いて計算ノードに配置された配列イメージを対象に，OpenACC のデータ移動および演算のオフローディングを行うプログラミングモデルである．また，XMP 指示文と OpenACC 指示文だけでは不可能なアクセラレータ間の直接通信をサポートする XACC 指示文も提供している．

XACC の実行単位は “ノード” と呼称する．XACC の実行モデルは，全ノードで同じプログラムが実行される Single Program Multiple Data（SPMD）である．XACC では，仮想インデックス集合である “テンプレート” を用いて分散配列を定義する．XACC のメモリモデルを**図 10** に示す．図 10 において各ノードに存在する緑色の矩形は，各ノードに割り当てられた分散配列を示している．XACC では，XMP 指示文を用いて分散配列をホストメモリに定義し，OpenACC 指示文を用いてその分散配列をアクセラレータメモリに転送する．また，ホスト間のデータ通信には XMP 指示文を用いるのに対し，異なるノードが持つアクセラレータ間およびアクセラレータとホスト間のデータ通信には XACC 指示文を用いる．

4.2 Omni Compiler

我々は, XACC の処理系として Omni Compiler [12,21,22] を開発している．Omni Compiler はベース言語（C 言語もしくは Fortran）と各指示文をランタイムの呼び出しに変換する source-to-source コンパイラである．

Omni Compiler の処理の流れを**図 11** に示す．(1) ユーザコード中に存在する XACC と XMP 指示文を，ランタイムの呼び出しに変換する．必要があればベース言語も変換する．(2) 変換されたコードを OpenACC コンパイラを用いてコンパイルし，オブジェクトファイルを生成する．(3) ネイティブコンパイラ（gcc, intel, or PGI など）を用いてオブジェクトファイルとランタイムとをリンクし，実行ファイルを生成する．

NVIDIA 社の GPU に対してノードを跨ぐ通信を行うために，Omni Compiler は下記の 3 つを実装している．(1) TCA/InfiniBand ハイブリッド通信を用いたもの．(2) GPUDirect RDMA を用いたもの．(3) MPI と CUDA を用いたもの．(1) は他の 2 つと比べて小さいレイテンシで通信を行うことが可能であるが，計算環境に TCA のシステムが必要になる．(2) は (3) と比較して性能に優れているが，MVAPICH2-GDR 等のソフトウェアおよびハードウェアのサポートが必要になる．(1) および (2) は GPU 間の直接通信を実現できるのに対し，(3) はアクセラレータ上のデータを CUDA を用いてホストメモリにコピーした後，MPI を用いて他ノードに転送する実装方法である．そのため，(3) は (1) と (2) に比べて性能は低いが，最も汎用的な実装と言える．

4.3 LQCD とは

QCD（Quantum Chromo-Dynamics : 量子色力学）は物質の最小単位であるクォークと，クォーク間における相互作用を結ぶグルーオン（糊粒子）を表す基本方程式である．LQCD は 4 次元（時間+XYZ 軸）の格子上で QCD のシミュレーションを行うものである．

LQCD の基本的な自由度はクォークとグルーオンであり，それぞれの物理量は複素数で表現される．クォークは 3 つの色を持つ “カラー” と 4 つのカラーを持つ “スピノル” を持つ．すなわちクォークは 4×3 の複素行列として表される [23]．グルーオンは SU(3) 群の元であり，3×3 の複素行列として表される．クォークは 4 次元格子の格子点上に定義されるのに対し，グルーオンは 4 次元格子の格子点を結ぶ格子線上に定義される．

4.4 実装

本節では，既存の LQCD ミニアプリケーション [24] をベースに XACC 化を行う．この LQCD ミニアプリケーションは C 言語で記述されている逐次コードであり, LQCD の実アプリケーションである Bridge++ [25] から作成されている．LQCD ミニアプリケーションの擬似コードを**図 12** に示す．U はグルーオンであり，それ以外の大文字はクォークである．図 12 中の *WD()* は Wilson-Dirac

```
 1  S = B
 2  R = B
 3  X = B
 4  sr = l2_norm(S)
 5  T = WD(U,X)
 6  S = WD(U,T)
 7  R = R − S
 8  P = R
 9  rr = l2_norm(R)
10  rrp = rr
11  do{
10    T = WD(U,P)
11    V = WD(U,T)
12    pap = dot(V,P)
13    cr = rr/pap
14    X = cr ∗ P + X
15    R = −cr ∗ V + R
16    rr = l2_norm(R)
17    bk = rr/rrp
18    P = bk ∗ P + R
19    rrp = rr
20  }while(rr/sr > 1.E−16)
```

図 12: LQCD ミニアプリケーションの擬似コード

```
 1  typedef struct Quark {
 2   double v[4][3][2];
 3  } Quark_t;
 4  typedef struct Gluon {
 5   double v[3][3][2];
 6  } Gluon_t;
 7  Quark_t X[NT][NZ][NY][NX], T[NT][NZ][NY][NX], ...;
 8  Gluon_t U[4][NT][NZ][NY][NX];
 9
10  #pragma xmp template t[NT][NZ]
11  #pragma xmp nodes n[PT][PZ]
12  #pragma xmp distribute t[block][block] onto n
13  #pragma xmp align [i][j][∗][∗] with t[i][j] shadow
        [1][1][0][0] :: X, T, ...
14  #pragma xmp align [∗][i][j][∗][∗] with t[i][j] shadow
        [0][1][1][0][0] :: U
15  #pragma acc enter data copyin(X, T, U, ...)
```

図 13: 分散配列の定義

operator [26] であり，クォークとグルーオンの相互作用を計算する．線形方程式を解くために，この LQCD ミニアプリケーションでは CG 法を用いている．そのため，L2 ノルムなどを計算するための数学的関数が必要になる．

本節で述べる XACC 化は文献 [13] と基本的には同様の手順であるが，生産性と性能の向上を図るため，下記の点において異なる．(1) XMP 指示文による分散配列の定義をより簡易に行うため，文献 [13] の Figure 19 で示した新記法を用いる．(2) 袖通信の性能を向上させるため，文献 [9,12] にある **reflect_init** と **reflect_do** 指示文を用いる．(3) 性能評価で用いるアクセラレータに適したスレッド割り当てを行うため，OpenACC の **loop** 指示文に **gang** 節を追加する（詳細は，文献 [13] の IV 章 B 節の第 3 段落を参考にされたし）．

4.4.1 分散配列の定義

図 13 にクォークとグルーオンの分散配列の定義を示す．1〜8 行目では，クォークとグルーオンの構造体配列を定義している．各構造体の最後の “[2]” は複素数の実数と虚数を表している．NT，NZ，NY，NX は，時間（T 軸）・Z 軸・Y 軸・X 軸の要素数である．10〜14 行目では，XMP 指示文を用いて 2 次元ノード集合を定義し，各ノードに対して上記で定義した分散配列の T 軸と Z 軸をホスト上に 2 次元ブロック分散している．PT と PZ は，T 軸と Z 軸に対するノード数である．Wilson-Dirac operator はステンシル計算であり，1 つ隣の要素を用いて計算を行う．そのため，**align** 指示文に **shadow** 節を用いることで，分散する各次元の領域は幅 1 の袖を持つように定義している．15 行目では，**enter data** 指示文を用いることで，XMP 指示文により定義された分散配列を，各ホストが持つアクセラレータに転送している．

```
1  #pragma xmp reflect_init(T, X, ...) width(/periodic/1,/
       periodic/1,0,0) orthogonal
2  #pragma xmp reflect_init(U) width(0,/periodic/1:0,/
       periodic/1:0,0,0) orthogonal
3    :
4  #pragma xmp reflect_do(U, X) acc
5  WD(T, U, X);
6  #pragma xmp reflect_do(T) acc
7  WD(S, U, T);
```

図 14: 袖通信と Wilson-Dirac operator の呼び出し

4.4.2 袖通信

Wilson-Dirac operator はステンシル計算であるため，その実行前に袖通信が必要である．袖通信には，XACC **reflect_init** と **reflect_do** 指示文を用いる．

図 14 に袖通信と Wilson-Dirac operator の呼び出しを示す．まず，**reflect_init** 指示文を用いて交換する袖の範囲を指定する．LQCD は周期境界を持つため，**width** 節中に **periodic** 修飾子を用いることで，周期的な袖の更新の設定を行う．なお，Wilson-Dirac operator はグルーオンの下部の袖のみを必要とするため，2 行目において “1:0” という設定を行うことで，下部の袖のみが更新されるように設定する．また，Wilson-Dirac operator は直交ノードが持つ要素のみを必要とするため，**orthogonal** 節も用いることで，直交ノード同士のみが袖交換を行うように設定する．袖領域の転送範囲はプログラム実行中に変わることはないため，**reflect_init** 指示文はプログラム中に 1 回だけ実行する（具体的には，図 12 の do-while 文の前で行う）．次に，**reflect_do** 指示文を用いて袖交換を行う．**acc** 節を用いることで，アクセラレータメモリ上のそれぞれの配列に対して袖交換を行うことを示している．関数 *WD()* の第 1 引数は出力結果を格納する配列を指定し，第 2・3 引数は入力のための配列を指定する．そのため，最初の関数 *WD()* の前には配列 U と X に対して袖通信を行うが，その直後の関数 *WD()* の前には，更新されている配列 T に対してのみ袖通信を行う．

MPI を用いた **reflect_init** 指示文の実装では，MPI の永続通信関数である *MPI_Send_init()*，*MPI_Recv_init()* と *MPI_Startall()* を用いている．TCA/InfiniBand ハイブリッド通信を用いた **reflect_init** と **reflect_do** 指示文の実装では，図 4 で示した PEACH2 が提供する API と

```
1  void WD(Quark_t out[NT][NZ][NY][NX], const Gluon_t u
       [4][NT][NZ][NY][NX], const Quark_t v[NT][NZ][NY][
       NX]){
2  #pragma xmp align [i][j][*][*] with t[i][j] shadow
       [1][1][0][0] :: out, v
3  #pragma xmp align [*][i][j][*][*] with t[i][j] shadow
       [0][1][1][0][0] :: u
4   ...
5  #pragma xmp loop (t,z) on t[t][z]
6  #pragma acc parallel loop collapse(4) present(out, u, v)
       gang(static:128) vector_length(128)
7   for(int t=0;t<NT;t++)
8    for(int z=0;z<NZ;z++)
9     for(int y=0;y<NY;y++)
10     for(int x=0;x<NX;x++){
```

図 15: Wilson-Dirac operator の一部

```
1  double norm(const Quark_t v[NT][NZ][NY][NX]){
2  #pragma xmp align [i][j][*][*] with t[i][j] shadow
       [1][1][0][0] :: v
3   double a = 0.0;
4
5  #pragma xmp loop (t,z) on t[t][z] reduction (+:a)
6  #pragma acc parallel loop collapse(7) present(v) gang(
       static:128) vector_length(128) reduction(+:a)
7   for(int t=0;t<NT;t++)
8    for(int z=0;z<NZ;z++)
9     for(int y=0;y<NY;y++)
10     for(int x=0;x<NX;x++)
11      for(int i=0;i<4;i++)
12       for(int j=0;j<3;j++)
13        for(int k=0;k<2;k++)
14         a += v[t][z][y][x].v[i][j][k]*v[t][z][y][x].v[i][j][k];
15
16  return a;
17 }
```

図 16: L2 ノルムのコードの一部

MPI の永続通信関数を用いて実装している．

4.4.3 ループ文の分割処理

図 15 に Wilson-Dirac operator のコードの一部を示す．2・3 行目では，関数 *WD()* のすべての引数は分散配列であるため，XMP 指示文を用いて引数の分散情報を再定義する．5 行目の **loop** 指示文は後に続く 4 重ループの内の外側 2 つを分割する．6 行目の **parallel loop** 指示文は **collapse** 節を用いて分割された 2 重ループを含む 4 重ループを統合し，アクセラレータ上で並列にループ文を処理する．

図 16 は CG 法で用いている数学的関数の 1 つである L2 ノルムのコードを示している．このコードでは図 15 と同様に，XMP 指示文と OpenACC 指示文を用いたループ分割を行っている．6 行目にある **reduction** 節は各アクセラレータにおいて集約演算を行い，ホスト上の変数 a にコピーされる．その後，5 行目にある **reduction** 節は各ホストが持っている a に対して集約演算を行う．

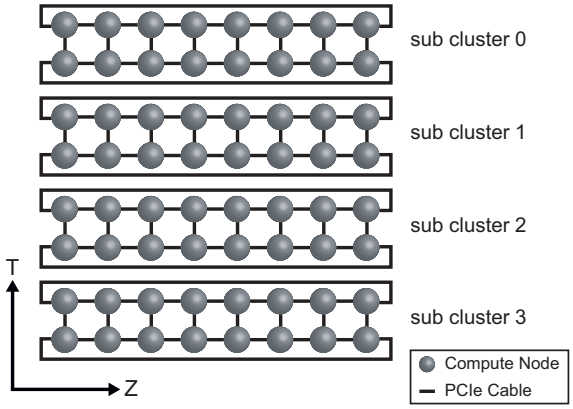

図 17: TCA/InfiniBand ハイブリッド通信利用時のプロセス配置

5. 評価

5.1 概要

本章では，Omni Compiler と XACC を用いた LQCD の実装の評価を行う．また，CUDA+MPI および OpenACC+MPI による LQCD の実装を用いて XACC との比較を行う．CUDA+MPI と OpenACC+MPI を用いた実装の通信には CUDA-Aware MPI による MPI 永続通信を用いている．比較項目としては，性能と生産性の 2 つの評価を行う．

5.2 性能評価

表 1 に示した HA-PACS/TCA システムを用いて性能評価を行う．XACC の性能評価には，4.2 節の第 3 段落で述べた (1) と (2) である TCA/InfiniBand ハイブリッド通信を用いたものと MVAPICH2-GDR を用いたものを用いる．問題サイズは $(NT, NZ, NY, NX) = (16, 16, 16, 16)$ であり，強スケーリングで計測する．3 章と同様に，TCA/InfiniBand ハイブリッド通信を用いた性能評価には GPU 0 を用い，MPI を用いた性能評価には GPU 2 を用いる．1 つの計算ノードにつき，1 つのプロセスを割り当て，最大 64 計算ノードで性能評価を行う．通信のバランスが良くなるように，T 軸と Z 軸へのプロセス分割はできる限り均等に行う．例えば，64 プロセス時は，T 軸と Z 軸をそれぞれ 8 プロセスずつ分割する．64 プロセス時の TCA/InfiniBand ハイブリッド通信を用いた XACC 実装のプロセス配置を**図 17** に示す．図 17 では省略しているが，全計算ノードは InfiniBand ネットワークでフラットに接続されている．Z 軸方向のブロックストライドデータ通信は PEACH2 が用いられ，T 軸方向の連続データ通信は MPI が用いられる．注意点として，T 軸方向の通信には PEACH2 は用いられない．

性能結果を**図 18** に示す．この結果より，高並列度において TCA/InfiniBand ハイブリッド通信を用いた XACC

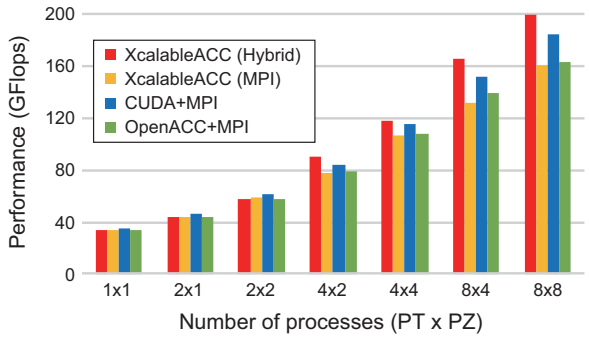

図 18: LQCD の性能

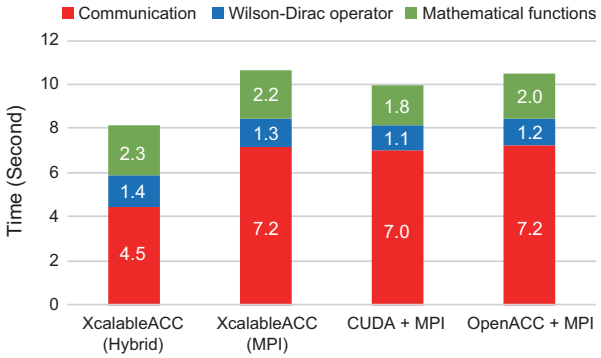

図 19: 64 計算ノード利用時の計算時間の内訳

```
1 #pragma xmp reflect_do(U, X) acc nowait
2 WD(T, U, X);
3 #pragma xmp reflect_do(T) acc nowait
4 WD(S, U, T);
```

図 20: **nowait** 節を追加した **reflect_do** 指示文

の性能が最も高いことがわかる．TCA/InfiniBand ハイブリッド通信を用いた XACC の性能は，CUDA+MPI よりも最大 9%，OpenACC+MPI よりも最大 18%性能が高かった．2 番目に性能が高いのは CUDA+MPI であり，MPI を利用した XACC と OpenACC+MPI はほぼ同じ性能であった．

図 18 における 64 計算ノード利用時の計測時間の内訳を**図 19** に示す．図 19 の通信時間には，ブロックストライドデータに対する Packing/Unpacking の時間も含まれている．この結果より，主に通信時間において，TCA/InfiniBand ハイブリッド通信を用いた XACC の性能が高いことがわかる．通信以外の，Wilson-Dirac operator や CG 法で用いる数学的関数の性能は，CUDA+MPI が他の実装と比較してわずかに良い．この理由を精査した結果，CUDA は 1 次元配列を用いているのに対し，OpenACC は 4 次元構造体配列を用いているため，配列の要素に対するインデックス計算が原因であることがわかった．そのため，図 18 において CUDA+MPI の性能は，MPI を利用した XACC と OpenACC+MPI と比較して性能が高かったと考えられる．

次に，XACC に対してさらなる性能向上について検討する．XACC の **reflect_do** 指示文内では，他の XACC 指示文との兼ね合いのため，袖通信が行われた後にバリア同期が発行される．しかしながら，本実装ではバリア同期は必要ないため，余分な同期となっている．そこで，**nowait** 節を新設し，それを **reflect_do** 指示文に追加することにより，バリア同期を抑制することを考えている．この場合，図 14 中の **reflect_do** 指示文は，**図 20** のようになる．この指示文を利用した 64 計算ノードを利用した性能結果を**図 21** に示す．**nowait** 節の追加により，TCA/InfiniBand ハイブリッド通信を利用した XACC の実装は 7%の性能向上，MPI を利用した XACC の実装は 4%の性能向上を達成できた．

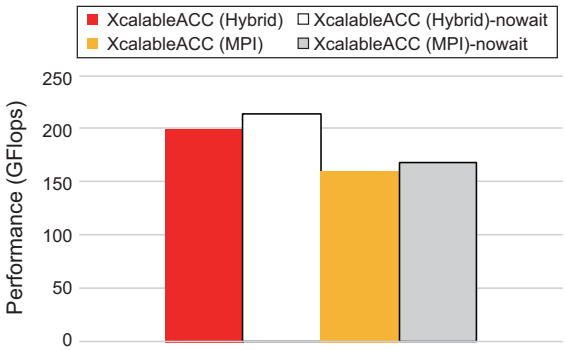

図 21: 64 計算ノード利用時の **nowait** 節を追加した LQCD の性能

本実装で作成した Z 軸方向の袖交換の通信データサイズは，グルーオンは $(576 \times NT \times NY \times NX/PT)$ Byte であり，クォークは $(192 \times NT \times NY \times NX/PT)$ Byte である．具体的には，本実験における 64 計算ノード時の通信サイズは，グルーオンは 288k Byte であり，クォークは 96k Byte である．3 章の結果より，これらのサイズよりも小さい方が，TCA/InfiniBand ハイブリッド通信は効果的であると考えられる．本実装では，T 軸と Z 軸に対して分割を行ったが，さらに大規模な計算環境においては Y 軸以降も分割する場合が考えられる．その場合，さらに細粒度の通信が発生するため，TCA/InfiniBand ハイブリッド通信は MPI 通信と比較してより有利になると考えられる．

5.3 生産性評価

本節では，XACC の生産性評価を，定量的評価と定性的評価に分けて行う．

5.3.1 定量的評価

各実装の行数を**表 2** に示す．表 2 には，逐次コードの行数およびコード中に含まれる各指示文と MPI 関数の行数も記載している．また，コメント，空行，波括弧のみの行は除外している．表 2 より，XACC の行数は最も少ないことがわかる．XACC の行数は，CUDA+MPI と比較して 21%少なく，OpenACC+MPI と比較して 10%少ない．

実装の行数以外の生産性のための評価基準の 1 つに Delta

表 2: 各実装の行数

	Serial	Xcalable-ACC	CUDA +MPI	OpenACC +MPI
SLOC	842	922	1,117	1,015
#XcalableMP	-	56	-	-
#OpenACC	-	16	-	21
#XcalableACC	-	2	-	-
#MPI function	-	-	54	54

表 3: 各実装の DSLOC

	XcalableACC	CUDA +MPI	OpenACC +MPI
Total	86	767	219
Add	80	348	173
Delete	0	73	0
Modify	6	346	46

```
1  #define LT ((NT/PT)+2)
2  #define LZ ((NZ/PZ)+2)
3  double norm2(const QCDSpinor_t v[LT][LZ][NY][NX]){
4    double a = 0.0;
5
6  #pragma acc parallel loop collapse(7) present(v)
       num_gangs(static:128) vector_length(128) reduction
       (+:a)
7    for(int it = 1; it < LT-1; it++)
8      for(int iz = 1; iz < LZ-1; iz++)
9        for(int iy = 0; iy < NY; iy++)
10         for(int ix = 0; ix < NX; ix++)
11           for(int ii = 0; ii < 4; ii++)
12             for(int jj = 0; jj < 3; jj++)
13               for(int kk = 0; kk < 2; kk++)
14                 a += v[it][iz][iy][ix].v[ii][jj][kk] * v[it][iz][iy
                       ][ix].v[ii][jj][kk];
15
16   MPI_Allreduce(MPI_IN_PLACE, &a, 1, MPI_DOUBLE,
         MPI_SUM, MPI_COMM_WORLD);
17   return a;
18 }
```

図 22: OpenACC+MPI による L2 ノルムのコードの一部

Source Lines of Code（DSLOC）[27] がある．DSLOC は，ベースとなるコード（LQCD の逐次コード）から目的のコード（LQCD の並列コード）を作成するために必要な作業の内訳（追加・削除・変更）の行数をカウントしたものである．DSLOC が小さければ，プログラミングコストは小さく，またバグが混入する確率も低いと言える．DSLOC の結果を**表 3** に示す．表 3 より，XACC の DSLOC は最も少ないことがわかる．XACC の DSLOC は，CUDA+MPI と比較して 89%少なく，OpenACC+MPI と比較して 61%少ない．

表 3 において，MPI を利用した 2 つの実装で追加行が多い理由は，袖通信のために送受信データを Packing/Unpacking し，また MPI 永続通信のためのデータ登録と通信実行を行っているからである．これに対し，XACC を用いた実装では，図 14 のように **reflect_init** 指示文と **reflect_do** 指示文を利用するだけでよい．また，MPI を利用した 2 つの実装で変更行が多い理由は，クォークなどの配列中のインデックスを袖を考慮したインデックスに変更する必要があるからである．例えば，図 16 に示した XACC による L2 ノルム関数において，逐次コードと XACC コードとの違いは指示文の有無のみである．比較のために，**図 22** に OpenACC+MPI による L2 ノルム関数を示す．図 22 より，OpenACC+MPI の実装では，OpenACC 指示文と MPI 関数の追加のみでなく，3・7・8 行目のようにインデックスの書き換えが必要であることがわかる．また，OpenACC+MPI と比較して CUDA の+MPI の DSLOC が多い理由は，GPU に対する処理を CUDA を用いて記述するため，クォークなどの多次元配列を 1 次元化し，カーネル関数を新規に作成する必要があるからである．特に，カーネル関数は逐次コードと大幅に異なるため，そのプログラミングコストは大きいと言える．

5.3.2 定性的評価

CUDA+MPI および OpenACC+MPI と比較した XACC の利点として，XACC 指示文は通信の実装を隠蔽するため，本稿で用いたような TCA/InfiniBand ハイブリッド通信をユーザは簡易に利用可能な点が挙げられる．また，XACC はアクセラレータに対する処理と計算ノード間の通信を 1 つの言語が扱うため，コンパイラによる通信の最適化が可能である．具体的には，コンパイル時に行われるコードの静的解析から，通信データの性質に応じた最適化（本稿で行ったような異種通信路の同時利用）が可能になる．

CUDA を用いた実装は NVIDIA 社の GPU のみの対応であるが，OpenACC および XACC を用いた実装は各コンパイラが対応しているハードウェアがであれば，あらゆる環境で動作することができる．そのため，OpenACC および XACC を用いた実装の方がポータビリティは高いと言える．また，OpenACC および XACC は既存コードに指示文を追加することでアクセラレータに対する処理を記述できるため，独自言語である CUDA を用いた実装と比較して，OpenACC および XACC を用いたコードの方が可読性は高くまた学習コストは小さいと言える．

5.3.1 節で述べた通り，MPI を利用した実装はインデックスに変換が必要であるのに対し，XACC を用いた実装は XMP 指示文が自動的にインデックス変換を行う．すなわち，XACC は逐次コードのイメージを保ったまま並列化可能であるため，XACC のコードの方が可読性は高いと言える．また，5.3.1 節の最後の段落で述べた Y 軸以降の分割を XACC を用いて行うには，図 13 と図 14 にある XACC と XMP 指示文に対して次元を 1 つ追加するだけでよい．これに対し，同じ並列化を MPI を用いた実装で行う場合は，コードは現実装よりも複雑化すると考えられる．この

ことから，XACC では多次元配列を用いたアプリケーションの並列化が非常に容易であるといえる．

6. 関連研究

本稿で用いた PEACH2 の最大バンド幅は PCIe Gen2 の制約により 4GB/s であり，HPC クラスタシステムで一般的な InfiniBand FDR と比較して低い．この問題点を克服するため，低レイテンシだけでなく高いバンド幅も持る PCIe Gen3 に基づく PEACH3 [28] が開発されている．ただし，PEACH3 も PEACH2 と同様にサブクラスタを越える通信を行うことはできないため，サブクラスタを跨ぐ大規模計算においては，本稿で用いた TCA/InfiniBand ハイブリッド通信が有効であると考えられる．

PEACH2 に類似した GPU 間の直接通信を実現する研究として APEnet+ [29,30] がある．APEnet+は，PEACH2 と同様に FPGA を用いたネットワークインタフェースであり，3-D トーラスネットワークを構築する．PEACH2 は PCIe プロトコルを用いているのに対し，APEnet+は独自のプロトコルを用いて GPU 間通信を実現する点が異なる．そのため，APEnet+はデータ通信時に PEACH2 では必要のないプロトコル変換が必要になると考えられる．また，APEnet+は，MPI に類似した通信 API のみを提供しているため，プログラミングのコストは MPI による記述と同等と考えられる．一方，PEACH2 は，XACC という高い抽象度を持つインターフェイスを用いることで簡易に利用することができる．また，APEnet+と InfiniBand などのコモディティネットワークによるハイブリッド通信の研究は行われていない．

GPU を搭載したクラスタシステムのための並列言語に X10 [31] や Chapel [32] がある．両言語は独自のシンタックスを用いて GPU を操作する．X10 および Chapel と XACC との違いは，X10 および Chapel は新言語であるのに対し，XACC は HPC 分野で広く用いられている C 言語と Fortran に対する指示文による並列拡張である点である．そのため，XACC の方が学習コストは小さいと考えられる．また，XACC では，既存の OpenACC や XMP で記述されたコードの大部分を XACC のコードとして再利用することができる．

Kokkos [33]，RAJA [34]，Alpaka [35]，Phalanx [36] はヘテロジニアスアーキテクチャのための C++テンプレートライブラリである．C++テンプレートライブラリの利点として，既存の C++コンパイラをそのまま利用可能な点が挙げられる．これに対して，XACC は C と Fortran に対する拡張であるため，指示文などを解析するための独自のコンパイラが必要になる．その代わり，XACC はベース言語に制限されずに，言語拡張を行えるという利点がある．例えば，XACC では Fortran2008 の Coarray 記法を C 言語にも導入することで，片側通信や部分配列を簡易に表現できる [11].

7. まとめと今後の課題

本稿では，XACC を用いて LQCD を開発し，64 計算ノードのアクセラレータクラスタ上で TCA/InfiniBand ハイブリッド通信を用いた評価を行った．また，性能および生産性について XACC との比較を行うために，CUDA+MPI および OpenACC+MPI を用いて LQCD を開発した．性能比較を行った結果，TCA/InfiniBand ハイブリッド通信を用いた XACC の性能は，CUDA+MPI の性能よりも 9%性能が高く，OpenACC+MPI の性能よりも 18%が高いことがわかった．また，XACC に対して新しい拡張である **nowait** 節を追加することで，XACC はさらなる性能向上を達成できることがわかった．次に生産性の定量的な比較を行った結果，XACC の行数は，CUDA+MPI と比較して 21%少なく，OpenACC+MPI と比較して 10%少ないことがわかった．また，XACC の DSLOC は，CUDA+MPI と比較して 89%少なく，OpenACC+MPI と比較して 61%少ないことがわかった．さらに，生産性の定性的な比較を行った結果，XACC 指示文は通信を隠蔽するため通信の最適化を簡易に行うことができ，また XACC は CUDA を用いた実装と比較してポータビリティに優れていることを述べた．

今後の課題として，GPU 以外のアクセラレータを搭載したクラスタシステムを用いて，XACC の性能ポータビリティについて調べる点が挙げられる．PEACH2 は PCIe を用いているため，Intel 社の Xeon Phi などの他のアクセラレータについても適用可能であり，また XACC で記述したコードは書き換えることなく，他のアクセラレータクラスタで動作可能である．他の課題として，XACC の性能と生産性についてさらに調べるため，LQCD のようなステンシルアプリケーション以外のアプリケーションの実装を行う予定である．その場合，Coarray 記法の利用が有効な場合があると考えている [37]．XACC では指示文はステンシル計算のような典型的な通信パターンに用いられるのに対し，Coarray はより柔軟な並列アルゴリズムの記述が可能になるからである．

Acknowledgements

We would like to extend grateful thanks to Hideo Matsufuru who provided us the Lattice QCD code in OpenACC. This research used the HA-PACS/TCA system provided by Interdisciplinary Computational Science Program in the Center for Computational Sciences, University of Tsukuba. We thank to Toshihiro Suzuki working in Cray Japan Inc. who did the maintenance of the HA-PACS/TCA system. The work was supported by the Japan Science and Technology Agency, Core Re-

search for Evolutional Science and Technology program entitled “Research and Development on Unified Environment of Accelerated Computing and Interconnection for Post-Petascale Era” in the research area of “Development of System Software Technologies for Post-Peta Scale High Performance Computing.”

参考文献

[1] TOP500 Supercomputer Sites. http://www.top500.org.
[2] The Green500 List. http://www.green500.org.
[3] S. Potluri and K. Hamidouche and A. Venkatesh and D. Bureddy and D. Panda. Efficient Inter-node MPI Communication using GPUDirect RDMA for InfiniBand Clusters with NVIDIA GPUs . October 2013.
[4] Jack Dongarra et. al. The international exascale software project roadmap. *The International Journal of High Performance Computing Applications*, Vol. 25, No. 1, pp. 3–60, 2011.
[5] M. Otten and J. Gong and A. Mametjanov and A. Vose and J. Levesque and P. Fischer and M. Min. An MPI/OpenACC Implementation of a High Order Electromagnetics Solver with GPUDirect Communication. *International Journal of High Performance Computing Applications*, pp. 1–15, 03 2015.
[6] Blair, Stu and Albing, Carl and Grund, Alexander and Jocksch, Andreas. Accelerating an MPI Lattice Boltzmann Code Using OpenACC. In *Proceedings of the Second Workshop on Accelerator Programming Using Directives*, WACCPD ’15, pp. 3:1–3:9. ACM, 2015.
[7] T. Hanawa, Y. Kodama, T. Boku, and M. Sato. Interconnection network for tightly coupled accelerators architecture. In *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*, pp. 79–82, Aug 2013.
[8] T. Hanawa, Y. Kodama, T. Boku, and M. Sato. Tightly coupled accelerators architecture for minimizing communication latency among accelerators. In *2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*, pp. 1030–1039, May 2013.
[9] T. Odajima, T. Boku, T. Hanawa, H. Murai, M. Nakao, A. Tabuchi, and M. Sato. Hybrid communication with tca and infiniband on a parallel programming language xcalableacc for gpu clusters. In *2015 IEEE International Conference on Cluster Computing*, pp. 627–634, Sept 2015.
[10] K. Matsumoto, T. Hanawa, Y. Kodama, H. Fujii, and T. Boku. Implementation of cg method on gpu cluster with proprietary interconnect tca for gpu direct communication. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, pp. 647–655, May 2015.
[11] XcalableACC Specification. http://xcalablemp.org/XACC.html.
[12] Masahiro Nakao et al. XcalableACC: Extension of XcalableMP PGAS Language Using OpenACC for Accelerator Clusters. In *Proceedings of the First Workshop on Accelerator Programming Using Directives*, WACCPD ’14, pp. 27–36, 2014.
[13] Masahiro Nakao and Hitoshi Murai and Hidetoshi Iwashita and Akihiro Tabuchi and Taisuke Boku and Mitsuhisa Sato. Implementing Lattice QCD Application with XcalableACC Language on Accelerated Cluster. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 429–438, Sept 2017.
[14] Toshihiro Hanawa and Yuetsu Kodama and Taisuke Boku and Mitsuhisa Sato. Interconnect for tightly coupled accelerators architecture. In *IEEE 21st Annual Symposium on High-Performance Interconnects (HOT Interconnects 21)*, pp. 79–82, 2013.
[15] Yuetsu Kodama and Toshihiro Hanawa and Taisuke Boku and Mitsuhisa Sato. PEACH2: FPGA based PCIe network device for Tightly Coupled Accelerators. In *Fifth International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART 2014)*, Vol. 42, pp. 3–8, December 2014.
[16] Altera Corp. Stratix IV Device Handbook. http://www.altera.co.jp/literature/lit-stratix-iv.jsp.
[17] PGI-SIG. Pci express external cabling specification, rev. 1.0, 2007.
[18] XcalableMP Specification. http://xcalablemp.org/specification.
[19] Masahiro Nakao et al. 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 ’12, pp. 402–409, 2012.
[20] Masahiro Nakao et al. Productivity and Performance of the HPC Challenge Benchmarks with the XcalableMP PGAS Language. In *7th International Conference on PGAS Programming Model*, pp. 157–171, 2013.
[21] Akihiro Tabuchi et al. A Source-to-Source OpenACC Compiler for CUDA. In *Euro-Par Workhops*, pp. 178–187, 2013.
[22] Omni Compiler. http://omni-compiler.org.
[23] 土井淳. XcalableMP による格子 QCD の並列化と Blue Gene/Q における性能評価. Technical Report 28, 研究報告ハイパフォーマンスコンピューティング（HPC）, Dec. 2014.
[24] Hideo Matsufuru. http://research.kek.jp/people/matufuru/Research/Programs/Tuning_Cpp/Solv_Wilson_Cpp/.
[25] Bridge++. http://bridge.kek.jp/Lattice-code/.
[26] Wilson, K. G. Confinement of quarks. *Phys. Rev. D*, Vol. 10, pp. 2445–2459, Oct 1974.
[27] Andrew I. Stone et al. Evaluating coarray fortran with the cgpop miniapp. In *Proceedings of the Fifth Conference on Partitioned Global Address Space Programming Models (PGAS)*, October 2011.
[28] T. Kuhara, T. Kaneda, T. Hanawa, Y. Kodama, T. Boku, and H. Amano. A preliminarily evaluation of peach3: A switching hub for tightly coupled accelerators. In *2014 Second International Symposium on Computing and Networking*, pp. 377–381, Dec 2014.
[29] R Ammendola, A Biagioni, O Frezza, F Lo Cicero, A Lonardo, P S Paolucci, D Rossetti, A Salamon, G Salina, F Simula, L Tosoratto, and P Vicini. Apenet+: high bandwidth 3d torus direct network for petaflops scale commodity clusters. *Journal of Physics: Conference Series*, Vol. 331, No. 5, p. 052029, 2011.
[30] R. Ammendola, M. Bernaschi, A. Biagioni, M. Bisson, M. Fatica, O. Frezza, F. Lo Cicero, A. Lonardo, E. Mastrostefano, P. S. Paolucci, D. Rossetti, F. Simula, L. Tosoratto, and P. Vicini. Gpu peer-to-peer techniques applied to a cluster interconnect. In *2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*, pp. 806–815, May 2013.

[31] Cunningham Dave et al. GPU Programming in a High Level Language: Compiling X10 to CUDA. In *Proceedings of the 2011 ACM SIGPLAN X10 Workshop*, X10 '11, pp. 8:1–8:10, 2011.
[32] A. Sidelnik et al. Performance Portability with the Chapel Language. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, pp. 582–594, May 2012.
[33] H. C. Edwards and C. R. Trott. Kokkos: Enabling Performance Portability Across Manycore Architectures. In *2013 Extreme Scaling Workshop (xsw 2013)*, pp. 18–24, Aug 2013.
[34] R. D. Hornung, J. A. Keasler. The RAJA Portability Layer: Overview and Status. Technical Report LLNL-TR-661403, LLNL, 2014.
[35] E. Zenker and B. Worpitz and R. Widera and A. Huebl and G. Juckeland and A. Knpfer and W. E. Nagel and M. Bussmann. Alpaka – An Abstraction Library for Parallel Kernel Acceleration. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 631–640, May 2016.
[36] Garland, Michael and Kudlur, Manjunath and Zheng, Yili. Designing a Unified Programming Model for Heterogeneous Machines. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pp. 67:1–67:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.
[37] Akihiro Tabuchi et al. Implementation and Evaluation of One-sided PGAS Communication in XcalableACC for Accelerated Clusters. In *International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, CCGrid '17, 2017.

付　　録

A.1　連続データの通信性能の事前評価

本節では，3.2 節で述べた InfiniBand を用いた連続データの通信性能の事前評価について述べる．MVAPICH2-GDR に対して性能チューニングを行った結果，QPI を跨がない通信については MVAPICH2-GDR の環境変数である “MV2_GPUDIRECT_LIMIT” の値を 524,288 に，QPI を跨ぐ通信については “MV2_USE_GPUDIRECT_RECEIVE_LIMIT” の値を 8,192 に設定すると高い性能を発揮することがわかった．

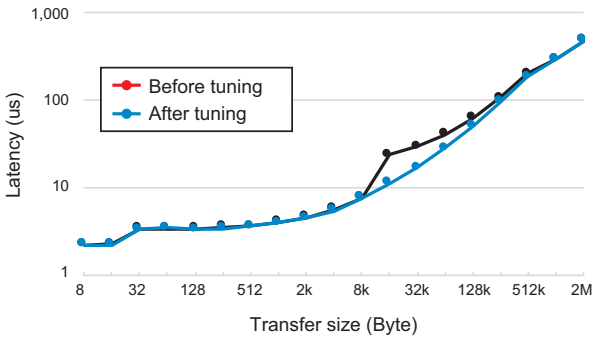

図 A·1: QPI を跨がない連続データの事前通信性能

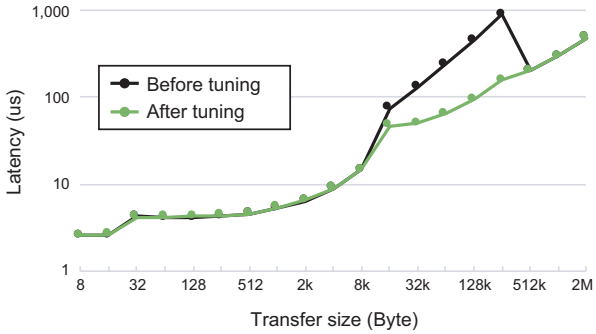

図 A·2: QPI を跨ぐ連続データの事前通信性能

上記のパラメータを設定した場合と，何も設定しなかった場合の結果を図 **A·1** と図 **A·2** に示す．これらの結果から，特に 8k〜512kByte において，パラメータを設定した方が高い性能を発揮することがわかる．

A.2　ブロックストライドデータの通信性能の事前評価

本節では，3.3 節で述べた InfiniBand を用いたブロックストライドデータの通信性能の事前評価について述べる．ブロックストライド通信を行う上で一般的な MPI_Datatype を用いる方法と，CUDA を用いて Packing/Unpacking する方法の性能評価を図 **A·3** と図 **A·4** に示す．これらの結果から，CUDA を用いて Packing/Unpacking した方が高い性能を発揮することがわかる．

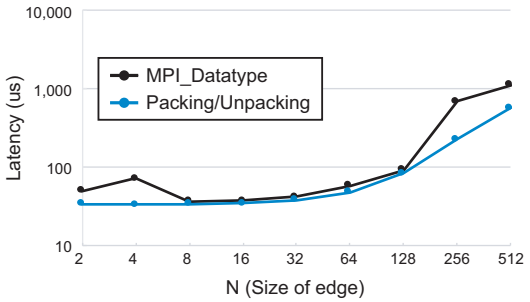

図 A·3: QPI を跨がないブロックストライドデータの事前通信性能

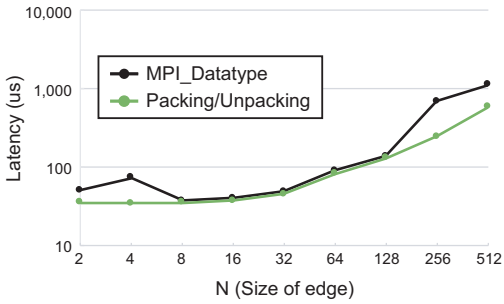

図 A·4: QPI を跨ぐブロックストライドデータの事前通信性能