Vol.2018-HPC-167 No.9
2018/12/17

OpenCLによるFPGA上の演算と通信を融合した並列処理システムの実装及び性能評価

藤田 典久[1]　小林 諒平[1,2]　山口 佳樹[2,1]　朴 泰祐[1,2]

概要：近年，高性能計算の分野で再構成可能なハードウェアである Field Programmable Gate Array (FPGA) が次世代の演算加速装置として注目されている．FPGA を高性能計算で用いる際の障壁は開発の困難さであったが，高位合成手法の発展に伴いこの問題は解決しつつある．最新の FPGA は最大で 100Gbps×4 の通信性能を有しており，我々はその強力な通信性能に注目している．FPGA の絶対性能は他のアクセラレータよりも低いが，FPGA が持つ演算能力と通信能力を組み合わせることでより広い範囲の問題に FPGA が適用できると考えている．本研究の目的は，高位合成で記述された FPGA アプリケーションから通信機構を操作し並列処理システムを実現することである．通信のスループットやレイテンシだけでなく，姫野ベンチマークを用いた性能評価を行い，高位合成で記述した FPGA アプリケーションで並列計算が可能なことを示す．我々は FPGA 間で直接通信を行う環境として Channel over Ethernet (CoE) というシステムを開発しており，バンド幅は最大で 7.13Gbps を達成し，4 バイト通信時のレイテンシは 980ns であった．姫野ベンチマークで，問題サイズ M を 4 FPGA で実行する場合に 22659 MFLOPS の性能が得られ，4 FPGA 時に 1 FPGA 時と比べて 3.61 倍という良好な Strong Scaling の結果が得られた．

1. はじめに

近年，高性能計算の分野で再構成可能なハードウェアである Field Programmable Gate Array (FPGA) が次世代の演算加速装置として注目されている．高性能計算では絶対性能に加えて電力効率を高めることが重要な問題であり，FPGA を用いてこの問題を解決しようとしている．

従前，FPGA を高性能計算で用いる際の障壁は開発の困難さであったが，高位合成 (High Level Synthesis; HLS) 手法の発展に伴いこの問題は解決しつつある．FPGA 開発では，Verilog HDL や VHDL といった ハードウェア記述言語 (Hardware Description Language; HDL) を用いて回路を記述することが一般的である．HDL は 1 クロック・1 ビット単位で回路の動作を記述するもので，利用に際してハードウェアの知識が必要となり，計算科学者が FPGA を用いる際の障壁となっていた．一方，高位合成は C や C++等のソフトウェア開発で用いられる言語を用いてハードウェアを記述するものであり，ハードウェア開発に精通していない科学者が FPGA を利用することを可能とするものである．

現在の FPGA の絶対性能は，アクセラレータとして広く用いられている Graphics Processing Unit (GPU) に敵わない．したがって，既に GPU で高速に計算できる問題を FPGA に適用しても，GPU の性能を超えられるとは言い難い．FPGA を高性能計算で用いる場合は，アプリケーションの中のどの計算が FPGA に適するかを見極めて適用することが重要だと言える．

我々は FPGA の持つ強力な通信性能に注目している．最新の FPGA は最大で 100Gbps×4 の通信性能を有している．また，それらの通信機構は直接 FPGA に接続されており，オーバーヘッドの少ない FPGA 間通信を可能とする．NVIDIA GPU では GPUDirect for RDMA (GDR) [1] と呼ばれる技術があり，外部の通信機構が GPU のメモリに直接アクセスができる．しかしながら，通信の制御などは CPU にあり，GPU が主体となって通信できる技術ではなく，CPU-GPU 間の同期コストや PCI Express (PCIe) を経由することによるオーバーヘッドは残っている．FPGA を単体のアクセラレータとして見ると適用できる範囲は小さいが，FPGA が持つ演算能力と通信能力を組み合わせることでより広い範囲の問題に FPGA が適用できると考えている．

本研究の目的は，高位合成で記述された FPGA アプリケーションから通信機構を操作し並列処理システムを実現することである．高位合成から通信できるシステムを開発し，それの性能評価を行う．通信のスループットやレイテ

[1] 筑波大学 計算科学研究センター
[2] 筑波大学 システム情報工学研究科

ンシだけでなく，姫野ベンチマークを用いた性能評価を行い，高位合成で記述した FPGA アプリケーションで並列計算が可能なことを示す．

2. 関連研究

OpenCL を FPGA で用いてアプリケーションやベンチマークの性能評価を行った論文はいくつか報告されている．[2] では，元々 GPU 向けに作成されたコードをそのまま FPGA 向けに用いても性能が悪く，OpenCL コードが FPGA 向けに最適化されている必要があると述べられている．HPC 研究会においても，[3] や [4] で FPGA と OpenCL を用いた研究報告がなされているが，どちらでも OpenCL の最適化の困難さ，すなわち，CPU や GPU と異なる記述スタイルが必要であると述べられている．FPGA の絶対性能は GPU など他のアクセラレータと比べると低く，どのような種類の処理を FPGA にオフロードするのかが重要となる．

本研究の独自性は，高位合成が持つ高い生産性と，FPGA が持つ高い通信能力を併せて利用し，並列計算を行うことである．FPGA と OpenCL を用いる研究は行われているが 1 FPGA のみを用いているものであり，並列計算が行われている例は知られていない．

3. Intel FPGA SDK for OpenCL

3.1 概要

Intel 社は自社の FPGA 向けの高位合成の処理系として，Intel FPGA SDK for OpenCL を公開している．この SDK を用いることで，OpenCL 言語を用いて FPGA をプログラミングできる．OpenCL から FPGA 上のハードウェアを生成するだけでなく，ホスト CPU で用いるドライバやランタイムライブラリも提供されており，この SDK のみでシステム全体を構築できる．

OpenCL 利用時の FPGA ハードウェアの構造の概略図を**図 1** に示す．図からわかるように，FPGA の内部構造は，大きくわけて 2 つの部分から構成される．1 つは Board Support Package (BSP) 由来の部分，もう一つはコンパイル対象の OpenCL コードから由来する部分である．

BSP は本 OpenCL SDK 固有の要素であり，異なる FPGA ボード上で同じ OpenCL プログラムを扱うために存在する．OpenCL 対応の FPGA ボードは多数あり，各ボードでハードウェアの構成が異なる．例えば，それぞれのボードで搭載されているメモリの種類や FPGA チップが異なる場合がある．したがって，それぞれのボードに固有の情報を OpenCL コンパイラに与える必要があり，BSP がその役割を担う．BSP にはペリフェラルコントローラが含まれており，OpenCL のアクセラレータとして利用するための最低限のものとして，PCIe コントローラと Dual Data Rate (DDR) メモリコントローラが含まれる．

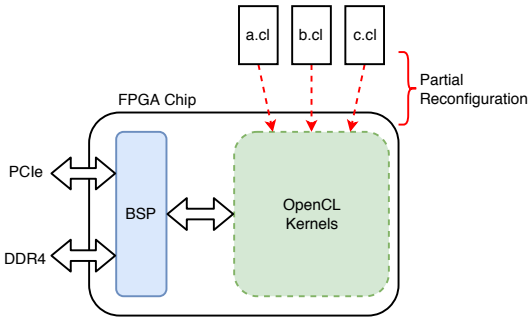

図 1: OpenCL 利用時における FPGA の内部構造の概略図．

3.2 通信機構に対応する BSP

通信機構を持つ OpenCL 対応の FPGA ボードであっても，そのボード用の BSP にコントローラが含まれていることは一般的ではなく，OpenCL から通信機構を操作し並列計算を行うためには，制御用のロジックを BSP に組み込む必要がある．我々は，これまでの研究 [5], [6] で，OpenCL の BSP に通信機構のコントローラを追加し，それを OpenCL コードから制御できることを明らかにしている．

本稿で実装した OpenCL 通信機構は，これらの研究で得られた知見を元に開発したものである．なお，BSP に対してコントローラを追加し，OpenCL カーネルから用いる手法の詳細については本稿では省略する．前述した研究会報告と論文を参照して頂きたい．

3.3 Channel 拡張

Intel FPGA SDK for OpenCL は OpenCL 言語に対していくつかの FPGA に特化した拡張を加えている．拡張の 1 つに，“Channel” 機構がある．Channel は OpenCL カーネル間でデータを直接交換するパイプのようなものである．カーネル間でデータをやりとりする場合，グローバルメモリに経由してデータを交換する方法が一般的であるが，Channel を用いる場合は FPGA 内部にデータパスが構成され，チップ外にあるメモリにアクセスすることなく通信ができる．したがって，従来手法と比べて高性能であり，メモリアクセスに関する回路が生成されないため省リソースとなる．Channel には “I/O Channel” と呼ばれる種類の Channel があり，これはカーネル間ではなく，BSP とカーネルの間を接続するために存在し，主に BSP にあるペリフェラルコントローラと OpenCL カーネル間の接続に用いられる．

本研究では，**図 2** にある様に，BSP に通信機構を操作するためのコントローラ (40Gbit Ethernet) を追加し，それを OpenCL コードから操作するために I/O Channel を用いる．また，Channel を用いて複数のカーネルの間を接続し，通信に必要な処理を実現する．通信機構の実装の詳

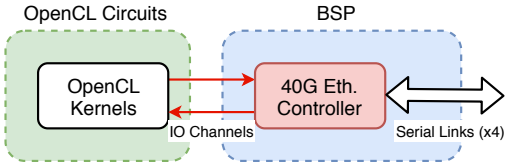

図 2: BSP に含まれる通信機構のコントローラと OpenCL カーネル間の接続.

sender code on FPGA1

```
__kernel void sender(__global float* restrict x, int n) {
    for (int i = 0; i < n; i++) {
        float v = x[i];
        write_channel_intel(simple_out, v);
    }
}
```

receiver code on FPGA2

```
__kernel void receiver(__global float* restrict x, int n) {
    for (int i = 0; i < n; i++) {
        float v = read_channel_intel(simple_in);
        x[i] = v;
    }
}
```

図 3: CoE を用いて通信を行うコード例.

細については第 4 章で述べる.

4. CoE: Channel over Ethernet

4.1 概要

我々は FPGA 間で直接通信を行う環境として Channel over Ethernet (CoE) というシステムを開発している. CoE の基本コンセプトは，前述した Channel による通信を FPGA 内だけでなく，ノード間に拡張するものである. CoE システムは，BSP に追加された Ethernet コントローラおよび周辺回路 (HDL で記述)，OpenCL カーネルで記述された制御カーネル群と，それらの間をつなぐ Channel，CoE システムとアプリケーションの境界にある Channel である “CoE Channel” から構成される.

図 3 に CoE を用いて通信する際の簡単な例を示す. 2 つのカーネル関数は異なる FPGA 上で動作しており，送信側の “`simple_out`” channel と受信側の “`simple_in`” channel が CoE システムを通じて繋っており，通信が可能となる.

CoE の通信プロトコルはその名前が示す通り Ethernet を用いている. FPGA で利用できるプロトコルは Ethernet だけではないが，その中から Ethernet を採用している理由は，市販されているスイッチを用いて多数の FPGA からなるネットワークを構築できるからである.

Ethernet を用いるということは，Ethernet の仕様に従ったパケットフォーマットに従って通信をする必要があるが，CoE ではそれらを行う回路を OpenCL 側で実装している. この方式のメリットは OpenCL 側と BSP 側のインターフェイスが送信・受信のあわせて 2 つの I/O Channel で固定される点にある. もし，パケット構築などを BSP 側に構築してしまうと，アプリケーション側で使われる Channel インターフェイスが変化するたびに BSP を再構築しなければならない.

HDL は生産性が低く機能開発や動作確認に時間がかかること，外部インターフェイスで要求される動作周波数[*1]を満すためのチューニングの手間が大きい，といった点から BSP の変更はコストが高いため留めたい. 一方で，こういった処理を OpenCL で実装することは性能面での低下が懸念される. 通信スループット性能については外部リンクに性能の上限があるため，OpenCL で実装したとしても十分満せると考えられるが，OpenCL ではサイクルレベルでの動作記述ができないため，レイテンシの最適化が困難であること，一般的に高位合成は HDL で記述する場合よりも多くの回路リソースを消費するという問題点がある. これらの要素を考慮して，制御に関するロジックを OpenCL で記述するメリットの方が大きいと判断し，OpenCL で記述する方針を選択している.

4.2 一対一通信

CoE における一対一通信の送信側のデータの流れを**図 4** に示す. この図は姫野ベンチマークにおける CoE ネットワークを表したものであり，4 つの袖領域交換用 (X 次元と Y 次元用にそれぞれ 2 つ) の channel と，Allreduce のための channel がある.

CoE の送信回路は以下に示す 4 つのカーネルから構築されており，全て OpenCL で記述されている.

expander カーネルからの入力データを 128bit 単位にパッキングして buffer に送り出す.

buffer expander からのデータをバッファリングし，Ethernet パケットに整形しパケットストリームを作成する.

scaler 単一の値を含むパケットを生成する. バッファリングをしないため高速である.

multiplexer 複数のパケット列をマージし，1 つのストリームを形成する.

CoE は 2 つの送信モードを持つ. 1 つはバッファリングモードであり，高いスループットを得るためにある程度のデータをバッファリングし，1 つのパケットに纏めて送信する. このモードでは expander カーネルと buffer カーネルを用いてパケットが生成される. もう 1 つはスカラーモードであり，これは単一の値を低レイテンシで送信することを意図している. このモードは集団通信を実装する際に用いることを想定しており，バッファリングを行わないため低リソース・低レイテンシな通信を行える. このモードでは scalar カーネルを用いてパケットが生成される.

受信側は複雑なバッファ処理が必要ないため，送信側よりシンプルに構成されている. パケットに含まれている宛先 Channel の ID から経路を判断して Channel にデータ

[*1] 例えば，PCIe バスに関する回路は 250MHz，40GbE に関する回路は 312.5MHz で動作することが求められる.

を分別するカーネルと，し Ethernet ヘッダの除去を行うカーネルの 2 つから構成されており，単一の動作モードから成る．

CoE Channel はそれぞれ固有の ID を持っており，ID と宛先アドレスで通信相手を決定する．宛先は送信用パケットを生成するカーネルを起動する際に引数で指定する．ホスト側のコードと OpenCL カーネルの間のインターフェイスをカーネル起動で行うため，カーネルの実行モデルに由来する制限がある．OpenCL の仕様では実行中のカーネルをホストから停止する手段がない．すなわち，一度カーネルを起動して宛先を決定すると，プログラム実行中は変更できない．しかしながら，高性能計算のアプリケーションにおける一対一通信は通信相手が固定であることが一般的であるため，この制限はさほど問題にならないと考えている．また，Intel OpenCL 環境には Host Channel と呼ばれるホストと FPGA 間を Channel 接続する機能があり，この機能を使えば実行中のカーネルにデータを渡せるため，この制限を解消できるものと考えており，今後実装する予定である．

4.3 集団通信: Allreduce

CoE は一対一通信だけでなく集団通信も同様のアプローチで実装している．現時点で CoE に実装されている集団通信は Allreduce のみであるため，本節では Allreduce についてのみ述べる．

図 5 に CoE で用いている Allreduce のアルゴリズムを示す．ツリー型のアルゴリズムを採用しており，それぞれの正方形はカーネルを表している．“+” は入力を加算して出力するカーネル (add)，“=” は入力をそのまま出力にコピーするカーネル (dup) である．このアルゴリズムは一対一通信のみで構築できるため CoE のシステムに適することと，演算順序が固定であり浮動小数点数の演算を行なっても最終的に全てのプロセスで同じ結果が得られる点が利点である．

前節で述べた通り，CoE の一対一通信は通信相手を高頻度に切り替えることを想定していない．したがって，一対一通信で構築される Allreduce もそれに由来する制限があり，Allreduce に参加する最大ノード数を予め決定しツリーの段数を静的に決めなければならない．add も dup も 2 入力 2 出力 channel のカーネルであり，カーネル引数で入力 channel と出力 channel の有効無効を切り替えられる．ノード数が最大数よりも少ない場合は，ツリー中に何もしない段を作ることで同じ回路で Allreduce が可能である．現在の実装で対応しているデータ型は float のみ，対応している演算も MPI_SUM 相当のみであるが，演算ロジックが OpenCL で組まれているため，他の演算やデータ型の追加は容易であると考えている．

4.4 制限事項

CoE は開発中のシステムであり，いくつかの制限事項が残っている．

- アプリケーションと CoE システムの間の Channel は float 型しか取れない．
- multiplexer におけるバッファリングの不足により，同時に 1 つの channel しか通信できない．
- フロー制御や再送制御の実装はなく，受信側のバッファが不足した場合はパケットが脱落する．

特にバッファリングの機能不足により通信性能が制限されている．複数の channel に対して float 型のデータを書き込んでも，あるクロックサイクルで送信されるデータは 1 つの channel からしか選ばれない．言い換えると，32bit/cycle の通信能力しか持たないということである．これらの制限事項は引き続き開発を行い解消してく予定である．

5. 姫野ベンチマークの CoE 実装

5.1 FPGA 実装および最適化

姫野ベンチマークは非圧縮流体解析を行うベンチマークプログラムである [7]．姫野ベンチマークの計算のパターンは高性能計算で典型的なステンシル計算を行うものであり，ポアソン方程式をヤコビ反復法を用いて解く．姫野ベンチマークは C および Fortran で記述されており CPU 向けの実装はあるが，FPGA 向けの実装は存在しないため，まずは FPGA 向けの実装を行う必要がある．本研究では，C 言語版の姫野ベンチマークをベースとして，FPGA OpenCL 版の実装を行う．

姫野ベンチマークは配列 p に関して 19 点ステンシル計算であり，配列 p を複数回参照するコードである．通常，CPU や GPU などのプロセッサで実行する場合はキャッシュを通じてメモリにアクセスをするため，配列 p に対するアクセスはキャッシュヒットすることが期待でき，配列 p のアクセスで消費されるメモリ帯域が減少する．しかしながら，現在の OpenCL 環境ではメモリアクセスに対するキャッシュの作成は限定的であり，ステンシル計算のパターンではキャッシュがないに等しい．

FPGA でステンシル計算のメモリアクセスを最適化する場合，シフトレジスタを用いて最適化する手法が一般的である [8]．また，同手法が OpenCL で記述する際にも有効なことが明らかとなっている [2]．本研究における姫野ベンチマークの実装も，同様の手法で最適化を行っている．DDR メモリから読み出した配列 p の要素を再利用のためにシフトレジスタに格納し，再び同じ要素にアクセスする際は DDR メモリからではなくシフトレジスタから取り出す．

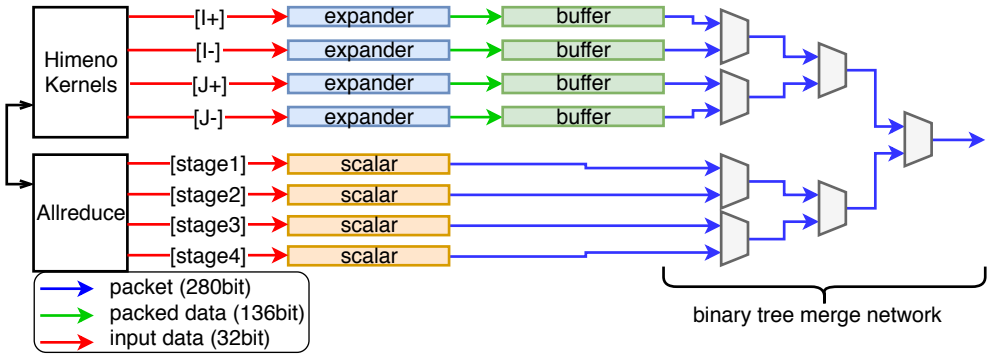

図 4: CoE 送信回路の構成.

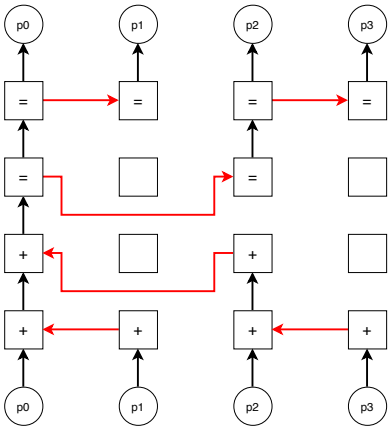

図 5: Allreduce 実装の通信パターン図．p0〜p4 はそれぞれプロセス，赤矢印は FPGA 間通信，黒矢印は FPGA 内の通信を表す.

```
A  for (i = 1; i < imax-1; i++)
    for (j = 1; j < jmax-1; j++)
     for (k = 1; k < kmax-1; k++) {
      s0 = ...;
      ss = ...;
      wgosa += ss * ss;
      wrk2[i][j][k] = p[i][j][k] + omega * ss;
   }

B  for (i = 1; i < imax-1; i++)
    for (j = 1; j < jmax-1; j++)
     for (k = 1; k < kmax-1; k++) {
      p[i][j][k] = wrk2 [i][j][k];
   }

C  sendp(ndx, ndy, ndz);
   MPI_Allreduce(&wgosa, &gosa, ...);
```

図 6: CPU 版の姫野ベンチマークの計算の流れ.

5.2 CoE の適用

姫野ベンチマークの CPU 向け C 言語版のカーネル部全体の疑似コードを**図 6** に示す．カーネルは大きくわけて，演算部 (A)，更新部 (B)，通信部 (C) の 3 部分から構成されている．CPU の実装では，ノード間の並列化の通信手法に Message Passing Interface (MPI) を用いており，通信部で MPI 関数を呼び出すことで袖領域の更新および gosa 変数のリダクションを行う.

ステンシル計算では，通信最適化の 1 つとして，通信と計算のオーバーラップがよく用いられる [9]．オーバーラップを行う際は，各タイムステップでまず袖領域の値を先に計算し，その後，袖領域の通信とそれ以外の領域の計算を同時に行うことで，通信時間の隠蔽を図る．姫野ベンチマークでは，計算と通信はオーバーラップしておらず，計算 (A+B) が終ってから袖領域の通信 (C) を行うが，通信と計算のオーバーラップを適用可能であり，適用しても計算は破綻しない.

図 7 に CoE を適用した姫野ベンチマークコードの一部を示す．この部分は図 6 の A の部分のループに相当する部分である．`write_channel_intel` 関数が 4 つ記述されているが，この Channel への書き込みによってデータが CoE システムに渡り，最終的に隣接 FPGA の Channel に配送される．“`if (enable)`” で囲まれた演算部と，4 つの `write_channel_intel` がシーケンシャルに実行されるように見えるが，FPGA 上ではパイプラインのハードウェアとして表現されているため，これらの要素はそれぞれ並列して動作する.

姫野ベンチマークに CoE を適用する場合を考えると，CoE の通信はパイプラインの動作モデルを前提しているため，MPI のように通信と計算のオーバーラップを行うという概念が不要なことがわかる．計算と通信が一体のパイプラインを形成しているため，計算を司る回路と通信を司る回路は平行に動作し，計算をしながら通信を行うという挙動が自然に記述できる.

6. 性能評価

6.1 評価環境

性能評価には Pre-PACS version X (PPX) クラスタシステムを用いる．PPX は筑波大学 計算科学研究センターで運用中のシステムであり，同センターが開発を計画している PACS シリーズ・スーパーコンピュータ次世代機のプロトタイプシステムである.

PPX は Intel FPGA を持つノード，Xilinx FPGA を持

```
bool enable = (1 <= i && i < t.imax - 1) &&
              (1 <= j && j < t.jmax - 1) &&
              (1 <= k && k < t.kmax - 1);
float value = 0;
if (enable) {
 s0 = ...;
 ss = ( s0 * v.a[3] - v.p[18] ) * v.bnd;
 wgosa += ss*ss;
 value = v.p[18] + t.omega * ss;
 t.wrk2[iter] = value;
}
if (!t.last && t.pi > 0 && enable && i == 1) {
 write_channel_intel(himeno_comp3_i_neg, value);
}
if (!t.last && t.pi < t.ndx0 - 1 &&
    enable && i == t.imax - 2) {
 write_channel_intel(himeno_comp3_i_pos, value);
}
if (!t.last && t.pj > 0 &&
    enable && j == 1) {
 write_channel_intel(himeno_comp3_j_neg, value);
}
if (!t.last && t.pj < t.ndy0 - 1 &&
    enable && j == t.jmax - 2) {
 write_channel_intel(himeno_comp3_j_pos, value);
}
```

図 7: CoE を適用した姫野ベンチマークにおける袖通信部のコード.

つノードの 2 グループからなるが，本研究では Intel FPGA のみを持ちいる．**表 1** に PPX システムの Intel FPGA ノードの仕様を示す．各ノードに Broadwell Xeon CPU ×2, NVIDIA P100 GPU×2, InfiniBand EDR HCA×1, BittWare FPGA ボード ×1 が搭載されている．また，**図 8** にあるように CPU, GPU 向けの InfiniBand ネットワークだけでなく，FPGA だけが接続されている専用ネットワークがあり Mellanox 社製のスイッチを用いて FPGA 間が接続されている．

本稿の性能評価は最大で PPX 4 ノードを用いて行う．スイッチはポートあたり最大で 100Gbps の能力を有するが，FPGA ボード側が最大で 40Gbps までの通信しか対応していないため，40Gbps の速度で用いる．また，それぞれの FPGA ボードは 2 つの QSFP+ポートを持つが，今回の実験では片側のみ利用している．

6.2 pingpong ベンチマーク

CoE 通信機構の基礎性能を評価するために pingpong ベンチマークの評価を行う．pingpong ベンチマークは**図 9** の様に実装しており，2 つの FPGA 間でデータを交換し性能を測定する．ただし，2 つの FPGA 間は直結ではなくスイッチを用いて接続しているため，それによるオーバーヘッドが含まれている．

時間の測定は**図 10** にあるように，3 箇所を OpenCL カーネルの動作クロック精度で時間を測定している．Arria 10 FPGA で OpenCL を利用する場合，カーネルコードによって動作周波数は変化するが，一般的に 200〜250MHz の範囲になり，4〜5ns の精度で時間を測定できる．通信レイテンシおよびバンド幅は片道レイテンシに通信時間を足した $\frac{t_1 - t_0}{2} + (t_2 - t_1)$[s] を元に計算する．

レイテンシの測定結果を**図 11** に，バンド幅の測定結果を**図 12** に示す．なお，今回測定に使用した実装は 230MHz

表 1: 評価環境

CPU	Intel Xeon E5-2660 v4 × 2
CPU Memory	DDR4 2400 MHz 64 GB (8 GB × 8)
Host OS	CentOS 7.3
Host Compiler	gcc 4.8.5
OpenCL SDK	Intel FPGA SDK for OpenCL 17.1.2.304
FPGA	BittWare A10PL4 (10AX115N3F40E2SG)
FPGA Memory	DDR4 2133 MHz 8 GB (4 GB × 2)
Communication Port	QSFP+ × 2 (40 Gbps × 2)
Ethernet Switch	Mellanox MSN2100

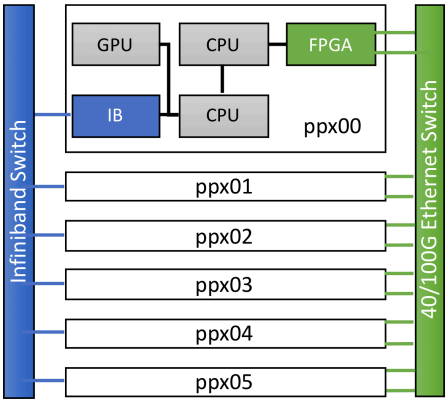

図 8: PPX のネットワーク図.

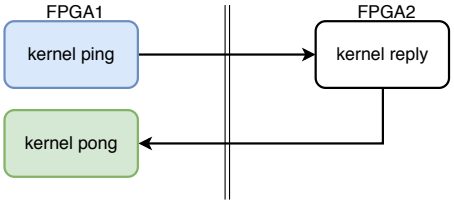

図 9: pingpong ベンチマークの構成.

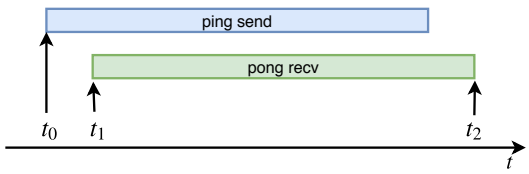

図 10: pingpong ベンチマークにおける時間計測.

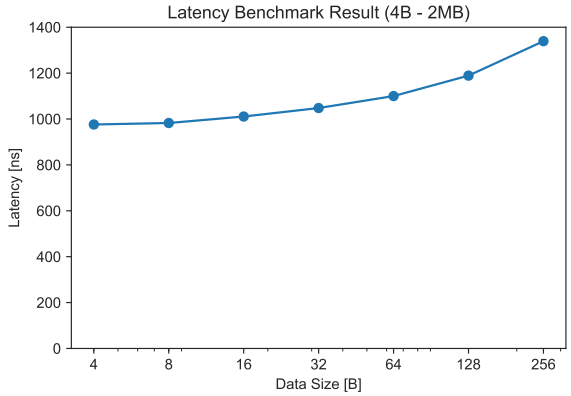

図 11: レイテンシの測定結果.

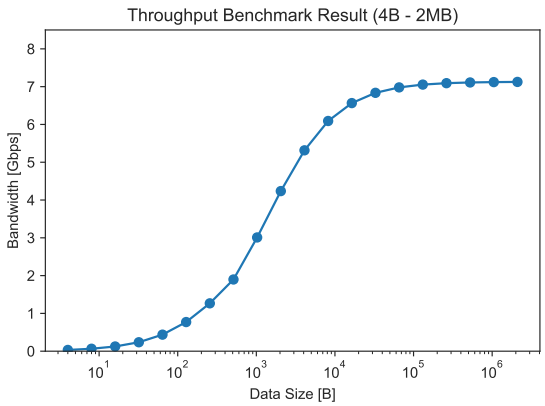

図 12: バンド幅の測定結果.

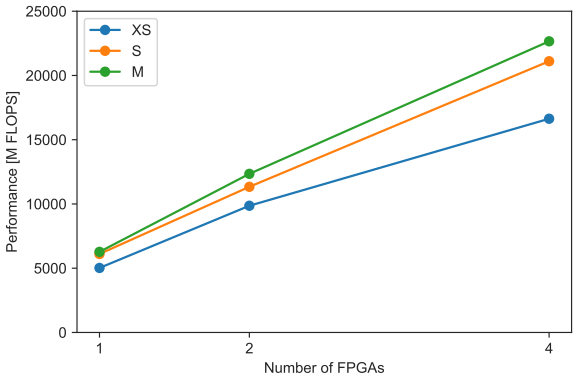

図 13: 姫野ベンチマークの性能のグラフ.

表 2: 姫野ベンチマークの性能 [M FLOPS].

	1 FPGA	2 FPGAs	4 FPGAs
XS	5,021	9,861	16,633
S	6,107	11,336	21,109
M	6,273	12,345	22,659

表 3: 各問題サイズで 1 FPGA を 1 とした場合の速度向上率.

	1 FPGA	2 FPGAs	4 FPGAs
XS	1.00	1.96	3.31
S	1.00	1.86	3.46
M	1.00	1.97	3.61

で動作しており，1 クロックサイクルの時間は 4.35ns である．Ethernet のパケット最大サイズは 512byte になるように作成する．ただし，Ethernet ヘッダ (12 バイト) および CoE ヘッダ (2 バイト) でパケットあたり 16 バイトを消費するため，ペイロードは 1 パケットあたり最大で 496 バイトとなる．

CoE 通信のバンド幅は最大で 7.13Gbps を達成し，CoE を用いた 4 バイト通信時の最小レイテンシは 980ns であった．ただし，OpenCL カーネルのクロックに同期して動くため，レイテンシは OpenCL カーネルの動作周波数に影響を受ける．また，送信回路のサイクル数がアプリケーションからいくつの Channel が CoE に接続されているか依存し，アプリケーションの構造によってもレイテンシは変動するため，全てのアプリケーションでこの値になるということではない．しかしながら，FPGA による直接通信が低レイテンシで行えていることを十分示せているといえる．

6.3 姫野ベンチマーク

本節では CoE を適用した姫野ベンチマークの性能を示す．第 5 章で述べた通り，姫野ベンチマークの実装はシフトレジスタを用いて最適化している．また，袖領域通信に必要なバッファを FPGA 内部のメモリに格納しており，これらのメモリ使用量は 1 FPGA あたりの問題サイズに比例する．そのため，本稿で用いた FPGA では M サイズまでの問題しか解くことができない．

オリジナルの姫野ベンチマークは実行時間が 1 秒になるように反復回数を決定し性能を測定するが，それでは問題サイズ，FPGA 数，実行毎の性能ゆらぎなどの影響で反復回数が変化してしまい性能比較が難しい．そのため本稿では反復回数を固定して測定しており，それぞれ XS=10000 回，S=1250 回，M=150 回に固定している．また，利用する FPGA 数を 1, 2, 4 FPGA と変化させ，ストロングスケーリングの条件で測定を行う．

CoE を適用した姫野ベンチマークの性能を**図 13** と**表 2** に示す．また，カーネルの動作周波数を**表 4** に，それぞれの問題サイズで 1 FPGA の時の性能を 1 とした性能向上率を**表 3** に示す．問題サイズ M を 4 FPGA で実行する場合に 22659 MFLOPS の性能が得られ，また，問題サイズ M で 4 FPGA 時に 1 FPGA 時と比べて 3.61 倍という良好なストロングスケーリングの結果が得られた．

7. 考察

CoE 通信のバンド幅は最大で 7.13Gbps を達成したが，物理層として 40GbE を用いていることを鑑みるとこの性能

表 4: 姫野ベンチマークのカーネル動作周波数 [MHz].

	1 FPGA	2 FPGAs	4 FPGAs
XS	210.00	227.08	227.08
S	226.19	219.44	220.37
M	219.44	220.83	210.42

表 5: 姫野ベンチマーク XS サイズにおける問題サイズとメモリサイズの比較.

	XS 1 FPGA	XS 2 FPGAs	XS 4 FPGAs
メモリサイズ	(33,33,65)	(19,33,65)	(19,19,65)
内点サイズ	(30,30,62)	(15,30,62)	(15,15,62)
FLOP/mesh	26.80	23.28	20.21
FLOP/mesh × MHz	5628.5	10474.1	16817.3
ratio	×1.00	×1.88	×3.26

は十分ではない．この性能の差は，1 クロックあたり 32bit データしか通信できないという現在の実装の制限から来るものである．ベンチマークの回路が 230MHz で動作しているため，得られる最大の性能は $0.23 \times 32 = 7.36$Gbps である．今回の実装のピークに対して 96%の性能が得られており，システムは想定通り動作しているといえるが，スループットの改善が今後の課題である．

姫野ベンチマークの性能測定結果から，問題サイズ XS の 4 FPGA の時のスケーリングが 3.3 倍となっていることがわかる．この場合並列化効率は 83%であり，FPGA の低レイテンシな通信を活かせていないように見える．しかしながら，この問題の原因は，通信ではなく，Strong Scaling により問題サイズが小さくなることで，内点と袖領域の比が悪化していることにある．現在の姫野ベンチマークの実装はメッシュデータのサイズ (パラメータ MIMAX, MJMAX, MKMAX) におおむね比例し[*2]，動作周波数によっても性能が異なる．メッシュサイズと内点サイズから求めた 1 メッシュあたりの FLOP と，それに対して動作周波数を掛けた値の比較を**表 5** に示す．表 5 のデータより，XS サイズで並列化効率が悪い原因は，通信によるオーバーヘッドではなく，問題サイズに対して袖領域の占める割合が多いことであるとわかる．

図 6 からわかるように，姫野ベンチマークは A で計算を行い wrk2 配列に書き込み，B でメモリコピーを行い配列 p に新しいメッシュデータを書き込む．FPGA 実装において B の部分は内部メモリバス幅と同じ 512bit 幅でコピーを行う．したがって，XS 4 FPGA の最も 1 FPGA あたりの問題サイズが小さい場合でも，少なくとも $\frac{15 \times 15 \times 62}{16} = 871.875$ クロックの時間が必要であり，実際には，これにメモリレイテンシやオーバーヘッドなどが加わるため，必要な時間はさらに伸びると考えられる．CoE の通信レイテンシは pingpong ベンチマークの結果より 1μs 程度であり，この時間は 220 クロック (220MHz 動作時) に相当する．したがって，B のメモリコピーにかかる時間の方が通信にかかる時間よりも長く，通信がボトルネックになっていないと考えられる．

5.2 節で，CoE においては計算と通信が一体のパイプラインを形成しているため，MPI のように通信と計算をオーバーラップさせるという概念が不要であり，計算をしながら通信を行うという挙動が自然に記述できると述べた．前述したメモリコピーと通信時間の比較は，計算と通信がパイプラインでほぼ同時に進行するからできることであり，CoE の通信モデルは FPGA で並列計算を行う際に適していると考えられる．

8. おわりに

本研究では，Intel OpenCL 開発環境が FPGA 向け拡張として実装している Channel API を，異なる FPGA 間に拡張するコンセプトである CoE 環境を開発し，性能評価を行なった．システムの性能を評価するために pingpong ベンチマークと姫野ベンチマークに CoE を適用した．

CoE 通信のバンド幅は最大で 7.13Gbps を達成し，CoE を用いた 4 バイト通信時の最小レイテンシは 980ns であった．姫野ベンチマークで，問題サイズ M を 4 FPGA で実行する場合に 22659 MFLOPS の性能が得られ，また，問題サイズ M で 4 FPGA 時に 1 FPGA 時と比べて 3.61 倍という良好なストロングスケーリングの結果が得られた．しかしながら，さらなる最適化の必要や，フロー制御なエラー検出など通信機構として実用するのに必要な機能がまだ不足しており，今後開発を進めていく予定である．

CoE においては計算と通信が一体のパイプラインを形成しているため，計算をしながら通信を行うという挙動が自然に記述でき，通信隠蔽を行いやすい．したがって，CoE の通信モデルは FPGA で並列計算を行う際に適していると考えられる．我々は宇宙物理学のアプリケーションの FPGA 向け最適化も行なっており [10]，今後，CoE の通信システムをそのアプリケーションに適用し，複数 FPGA を用いた並列計算を行う予定である．

また，筑波大学 計算科学研究センターでは，次期スーパーコンピュータ (名称: Cygnus) を 2019 年 5 月から運用する．Cygnus は 1 ノードに 2 CPU, 4 GPU, 2 FPGA を持つマルチヘテロジニアスなシステムとなり，FPGA 間は 4×100Gbps の通信を用いた 2D トーラスネットワークとなる．今後は本システムを Cygnus 上で稼動させるために，100Gbps 通信対応および 2D トーラスネットワーク対応を行う予定である．

謝辞 本研究の一部は，「高性能汎用計算機高度利用事業」における課題「次世代演算通信融合型スーパーコンピュータの開発」及び，文部科学省研究予算「次世代計算

[*2] 姫野ベンチマークは計算に必要な袖領域に加えてパディングを加えるため，内点サイズ+2 よりもメモリサイズは大きくなる．

技術開拓による学際計算科学連携拠点の創出」による．また，本研究の一部は，「Intel University Program」を通じてハードウェアおよびソフトウェアの提供を受けており，Intel の支援に謝意を表する．

参考文献

[1] NVIDIA Corporation: GPUDirect for RDMA, (online), available from ⟨https://docs.nvidia.com/cuda/gpudirect-rdma/index.html⟩.
[2] Zohouri, H. R., Maruyama, N., Smith, A., Matsuda, M. and Matsuoka, S.: Evaluating and Optimizing OpenCL Kernels for High Performance Computing with FPGAs, *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '16, Piscataway, NJ, USA, IEEE Press, pp. 35:1–35:12 (online), available from ⟨http://dl.acm.org/citation.cfm?id=3014904.3014951⟩ (2016).
[3] 大島聡史，塙　敏博，片桐孝洋，中島研吾：FPGA を用いた疎行列数値計算の性能評価，情報処理学会研究報告, 2016-HPC-153 (2016).
[4] 塙　敏博，伊田明弘，大島聡史，河合直聡：FPGA を用いた階層型行列ベクトル積，情報処理学会研究報告, 2016-HPC-155 (2016).
[5] 大畠佑真，小林諒平，藤田典久，山口佳樹，朴　泰祐：OpenCL と Verilog HDL の混合記述による FPGA 間 Ethernet 接続，情報処理学会研究報告, 2017-HPC-160 (2017).
[6] Kobayashi, R., Oobata, Y., Fujita, N., Yamaguchi, Y. and Boku, T.: OpenCL-ready High Speed FPGA Network for Reconfigurable High Performance Computing, *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, HPC Asia 2018, New York, NY, USA, ACM, pp. 192–201 (online), DOI: 10.1145/3149457.3149479 (2018).
[7] RIKEN: Himeno Benchmark, (online), available from ⟨http://accc.riken.jp/en/supercom/documents/himenobmt/⟩.
[8] Sano, K., Kono, F., Nakasato, N., Vazhenin, A. and Sedukhin, S.: Stream Computation of Shallow Water Equation Solver for FPGA-based 1D Tsunami Simulation, *SIGARCH Comput. Archit. News*, Vol. 43, No. 4, pp. 82–87 (online), DOI: 10.1145/2927964.2927979 (2016).
[9] Idomura, Y., Nakata, M., Yamada, S., Machida, M., Imamura, T., Watanabe, T., Nunami, M., Inoue, H., Tsutsumi, S., Miyoshi, I. and Shida, N.: Communication-overlap techniques for improved strong scaling of gyrokinetic Eulerian code beyond 100k cores on the K-computer, *The International Journal of High Performance Computing Applications*, Vol. 28, No. 1, pp. 73–86 (online), DOI: 10.1177/1094342013490973 (2014).
[10] 藤田典久，小林諒平，山口佳樹，朴　泰祐，吉川耕司，安部牧人，梅村雅之：並列 FPGA システムにおける OpenCL を用いた宇宙輻射輸送コードの演算加速，情報処理学会研究報告, 2018-HPC-165 (2018).