Vol.2019-HPC-170 No.5
2019/7/24

OpenCL対応FPGA間通信機能による GPU・FPGA複合型演算加速

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

概要：我々は，高い演算性能とメモリバンド幅を有する GPU (Graphics Processing Unit) に演算通信性能に優れている FPGA (Field Programmable Gate Array) を連携させ，双方を相補的に利用する GPU-FPGA 複合システムに関する研究を進めている．GPU，FPGA といった異なるハードウェアを搭載するシステム上では，各デバイスで実行される演算をどのようにプログラミングし，全デバイスを協調動作させるかが重要な課題となる．そこで本稿では，OpenCL コードから制御可能な FPGA 間通信技術と GPU-FPGA 間 DMA 転送技術を融合した，複数ノード上における GPU-FPGA 間連携手法を提案する．GPU-FPGA 間 DMA 転送は，GPU デバイスのグローバルメモリを PCIe アドレス空間にマップし，アドレスマップの結果をベースに OpenCL カーネル内で作成したディスクリプタを最終的に FPGA 内の PCIe DMA コントローラに書き込むことによって実現される．また，FPGA 間通信は，Verilog HDL で実装された Ethernet 通信を実行するハードウェアと，そのハードウェアの制御モジュール (OpenCL カーネル) を I/O Channel で接続することによって構成されているシステムで実現される．この提案手法を用いて，ノードを跨いだ GPU 同士の pingpong ベンチマークを実装し，それが正しく動作していることを確認した．

1. はじめに

高い演算性能とメモリバンド幅を有する GPU (Graphics Processing Unit) を演算加速装置として搭載する CPU-GPU 構成のクラスタが今日の HPC 分野において広く用いられている．このような構成のクラスタで並列処理を実行するためには，複数ノードをまたがる GPU 間の通信において CPU を介した複数回のメモリコピーが必要であり，このレイテンシの増加によってアプリケーションの性能が低下する問題があった．そこで，筑波大学計算科学研究センターでは，演算加速装置間を低レイテンシの通信ネットワークで密に接続する TCA (Tightly Coupled Accelerators) と呼ばれるコンセプトを提唱し，そのための通信機構である PEACH2 (PCI Express Adaptive Communication Hub Ver.2) [1] を独自開発した．コンセプトの実証システムとして，PEACH2 を搭載した HA-PACS/TCA (Highly Accelerated Parallel Advanced System for Computational Sciences/TCA) を運用し，ノードをまたぐ GPU 同士で低レイテンシ通信が実現されていることを確認した．

PEACH2 は FPGA (Field Programmable Gate Array) を用いて開発されており，FPGA とは任意の論理回路を電気的にプログラムすることができる集積回路である．その特性から，アプリケーションに特化した演算パイプラインと内部メモリシステムを実現する回路を FPGA 上に実装してユーザ所望の処理を加速させることが可能である．[2], [3] では，低レイテンシの通信を実行する回路に加えて，GPU が不得手とする処理を実行する回路を FPGA 上に実装し，それを FPGA に適宜にオフロードすることによってアプリケーション全体の性能を向上させる研究事例が報告されている．このような，FPGA に演算をオフロードし，通信機能と連携することによって演算と通信とを融合するコンセプトを我々は AiS (Accelerator in Switch) と呼んでおり，CPU-GPU クラスタ構成である現在の HPC システムの性能を更に向上させる鍵であると睨んでいる．**図 1** に AiS コンセプトの概要を示す．各ノードには GPU と FPGA が搭載され，それらは PCIe バスを介して接続されている．アプリケーションにおける大規模な粗粒度並列処理部分は従来通り GPU が担当しつつ，GPU ではカバーできない並列性の低い演算部分のオフロードおよび高速ノード間通信処理に FPGA を適用することによって，より効率的でレイテンシボトルネックの少ない強スケーリングの実現を目指す．

しかし，GPU や FPGA といった異なる種類の計算デバイスがノード内に混在するような複雑なプラットフォーム

[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

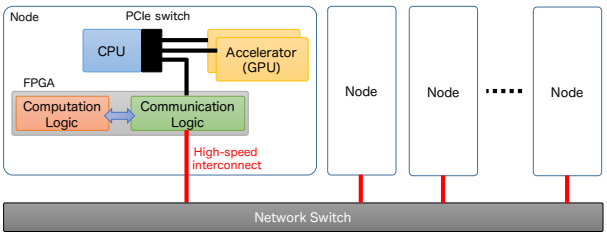

図 1: AiS コンセプトの概要．GPU では粗粒度並列処理を担当する計算カーネルが実行され，FPGA では GPU が不得手とする演算や集団通信を含む高速ノード間通信を担当するカーネルが実行される．CPU はこれらのカーネルの起動および全計算デバイスの調停を行う．

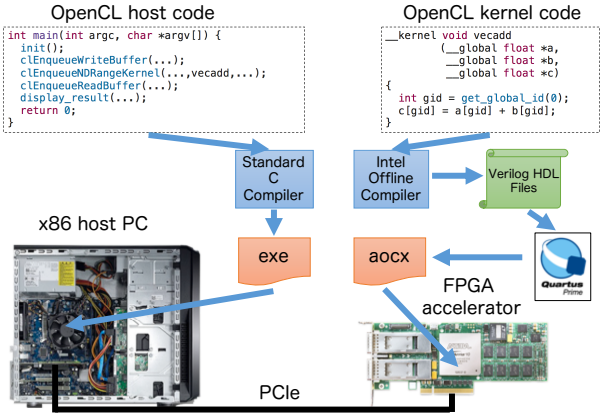

図 2: Intel FPGA SDK for OpenCL のプログラミングモデル．

上では，各デバイスで実行される演算をどのようにプログラミングし，全デバイスを協調動作させるかが重要な課題となる．これまでに我々は，GPU デバイスのグローバルメモリと FPGA デバイスの外部メモリ間で CPU を介さずにデータ転送を実現する機能を，PCIe DMA 転送用の IP (Intellectual Property) コアを用いて FPGA 上に実装し，その機能を FPGA ベンダーの提供する OpenCL ツールチェインの仕組みと Veriog HDL とを活用することによって制御する手法を提案している [4]．その GPU-FPGA 間 DMA 転送技術に加え，我々は OpenCL から制御可能な FPGA 間通信技術 [5] も開発しており，本稿ではこれらの技術を融合することによって実現される複数ノード上における GPU-FPGA 間連携について述べる．

2. Intel FPGA SDK for OpenCL

Intel は OpenCL を用いて FPGA 回路を設計できる開発環境 [6] を提供しており，我々の提案する手法はこのツールの利用を前提としている．**図 2** に Intel FPGA SDK for OpenCL におけるプログラミングモデルを示す．ユーザはホスト PC 上で動作するホストコードと FPGA 上で動作するカーネルコードとの 2 種類のコードを記述する．ホストコードは主に OpenCL API (Application Programming Interface) を用いての FPGA のコンフィグレーション，メモリ管理，カーネル実行管理などの FPGA デバイスの制御を担当し，カーネルコードは FPGA にオフロードされる演算を担当する．このプログラミングモデルでは，ホストコードとカーネルコードは別々にコンパイルされ，オフラインコンパイルのみがサポートされている．これは論理合成と配置配線，特に配置配線に数時間要するためである．ホストコードは gcc や Intel Compiler などの標準的な C コンパイラにてコンパイルされ，ホスト PC 上で動作する実行バイナリが生成される．カーネルコードは Intel FPGA SDK for OpenCL に付属している専用コンパイラにて，論理合成可能な Verilog HDL ファイルに変換され，バックエンドで動作する Quartus Prime がその Veirlog HDL ファイルから，FPGA の回路データを含む aocx ファイルを生成する．OpenCL API を用いることで，ホストアプリケーションの実行時に aocx ファイルが FPGA にダウンロード・回路の再構成が行われ，カーネルの実行に必要なデータやカーネルの実行結果などは PCIe バスを介して転送される．

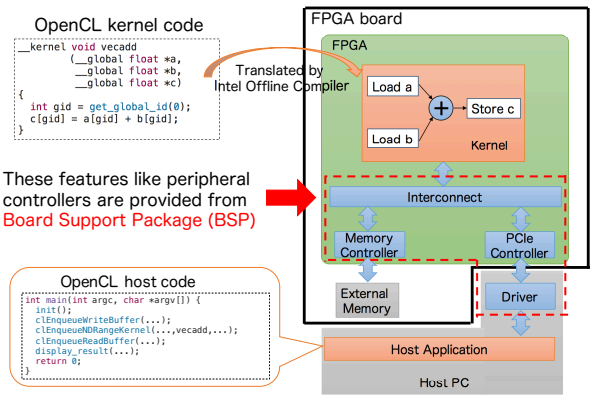

図 3: Intel FPGA SDK for OpenCL プラットフォームの構成図．

図 3 に Intel FPGA SDK for OpenCL プラットフォームの構成図を示す．C コンパイラによってホストコードからホストアプリケーションの実行バイナリが生成され，Intel FPGA SDK for OpenCL に付属している専用コンパイラによってカーネルコードに記述されている演算をパイプライン処理するハードウェアがカーネルコードから生成される．PCIe コントローラやデバイスドライバ，FPGA デバイスの外部メモリコントローラなどは Bittware や Terasic などの FPGA ボードベンダーから提供される BSP (Board Support Package) に同梱されている．FPGA ボード毎に，FPGA チップや外部ペリフェラル構成は異なる．ボード間のそれらの差異を吸収するために，ボード固有のパラメータや回路は BSP という形で提供され，カーネルコードのコンパイル時に BSP を読み込み利用する．一般的に，

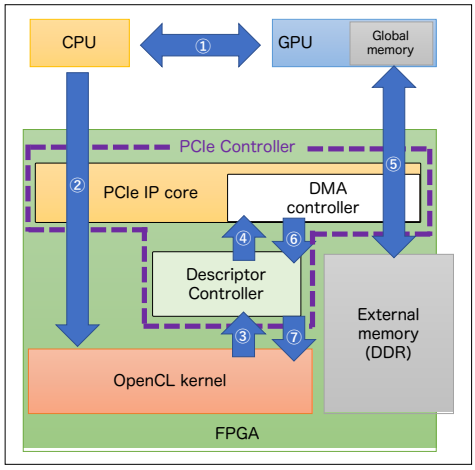

図 4: OpenCL による GPU-FPGA 間データ転送の概略図.

```
1  #define SIZE  1000000
2
3  tcaresult tcaCreateHandleGPU(unsigned long long *paddr,
       void *ptr, size_t size);
4
5  int main(void) {
6    uint32_t data[SIZE/4];
7    void* ptr;
8    cudaSetDevice(0);
9    cudaMalloc(&ptr, SIZE);
10
11   unsigned long long paddr;
12   tcaCreateHandleGPU(&paddr, ptr, SIZE);
13
14   printf("paddr = 0x%016llx¥n", paddr);
15
16 return 0;
17 }
```

図 5: PCIe アドレス空間へ GPU メモリをマップするコード．12 行目の tcaCreateHandleGPU() 関数で PCIe アドレス空間に GPU メモリをマップし，そのメモリアドレスを paddr に格納する．

OpenCL 対応の FPGA ボードを利用する場合，ボードの開発元から BSP が提供され，ユーザはその BSP を利用して OpenCL を用いた回路開発を行う．そのため，ユーザはホストコードとカーネルコードの実装のみに注力すればよく，たとえ異なる FPGA ボードを利用するとしても，その FPGA ボードの BSP が提供されていれば，既存のコードを移植することが可能である．

また，ユーザが使用したい外部ペリフェラルを BSP でサポートしていれば，それを OpenCL カーネルから制御することが可能である．しかし，基本的に BSP は OpenCL プログラミングを可能にする最低限のインターフェース，すなわち外部メモリコントローラと PCIe コントローラ，デバイスドライバしか提供していない．したがって，BSP でサポートされていない外部ペリフェラルを OpenCL カーネルからアクセスするためには，ユーザ自身でその外部ペリフェラルを操作するハードウェアコントローラを実装し，BSP に組み込む必要があり，そして BSP に組み込んだコントローラは，ベンダー拡張機能である I/O Channel API を使用する OpenCL カーネルコードを記述することによって，FPGA (OpenCL カーネル) からアクセスすることができる．すなわち，OpenCL から制御可能な GPU-FPGA 間データ転送 [4] と FPGA 間通信技術 [5] では，前者は BSP 中の PCIe コントローラに GPU-FPGA 間データ転送を実行する機能を追加し，それを OpenCL カーネルから I/O Channel API を介して制御しており，後者は FPGA ボードに搭載されているネットワークポートである QSFP+ (Quad Small Form-factor Pluggable Plus) を操作するコントローラを Verilog HDL で実装し，それを BSP に組み込み，OpenCL カーネルから制御している．以降の章にて，それらの詳細について述べる．

3. OpenCL から制御可能な GPU-FPGA 間データ転送

図 4 に，OpenCL から制御可能な GPU-FPGA 間データ転送の概要を示す．この機能は，GPU デバイスのグローバルメモリ，FPGA デバイスの外部メモリを PCIe アドレス空間にマッピングすることで，PCIe コントローラ IP が持つ DMA 機構を用いて双方のメモリ間でデータのコピーを行う．これは，かつて HA-PACS/TCA の開発 [1] において実現した，PCIe 上に接続された GPU と FPGA を PCIe のパケット通信プロトコルを用いて通信させる技術と基本的に同じであるが，この手法では **FPGA が自律的に DMA 転送を起動する**．FPGA から GPU に対しての DMA 転送は以下の手順で実行される．

- CPU 側での設定
  - (1) GPU のグローバルメモリを PCIe アドレス空間にマップ
  - (2) マップしたメモリアドレス情報を FPGA に送信
- FPGA 側での設定
  - (3) GPU メモリアドレス情報を元にディスクリプタを生成し，ディスクリプタコントローラに送信
  - (4) DMA コントローラにディスクリプタを書き込む
  - (5) デバイス間 DMA が起動
  - (6) DMA コントローラが完了信号を発行
  - (7) OpenCL カーネルで完了信号を検出

なお，CPU 側での設定だが，計算中は FPGA 上に保存された GPU 側アドレス情報やディスクリプタを繰り返し用いるため，①と②は**計算開始時に一度実行するだけで良い**．

3.1 PCIe アドレスマッピング

GPU のグローバルメモリを PCIe アドレス空間からアクセスするためには，NVIDIA が提供している API を用いてグローバルメモリを PCIe アドレス空間にマップす

表 1: ディスクリプタの形式.

Bits	Name
[31:0]	Source Low Address
[63:32]	Source High Address
[95:64]	Destination Low Address
[127:96]	Destination High Address
[145:128]	DMA Length
[153:146]	DMA Descriptor ID
[159:154]	Reserved

る必要がある．GPU メモリは CPU 上で動作する CUDA ライブラリや GPU ドライバによって管理されており，この API も GPU ドライバに実装されている．したがって，FPGA から GPU に対して直接通信を行う場合であっても，まず CPU 上で API を用いて PCIe アドレス空間から GPU メモリにアクセスできるように設定しなければならない．そして，DMA 転送を行う際に，GPU を指す PCIe アドレスを DMA 転送先あるいは転送元に指定することで，GPU-FPGA 間の DMA を実現できる．GPU メモリに関する制御には，PEACH2[1] で用いていたカーネルモジュールおよびライブラリを用いる．

PEACH2 で用いていた API を用いた PCIe アドレス空間への GPU メモリのマップ方法を**図 5** に示す．PEACH2 の API である tcaCreateHandleGPU() 関数にホスト側で作成したポインタを渡すことにより，PCIe アドレス空間にマップされた GPU メモリのアドレスである paddr を知ることができる．この関数は，もともと PEACH2 の通信対象とするメモリ領域を識別するためのハンドルを作成する関数であるが，内部的には前述した NVIDIA が提供する Kernel API を用いて GPU アドレスを PCIe アドレスにマップしそのアドレスを取得しており，この手法ではその機能を流用している．

3.2 ディスクリプタの生成

BSP 内の PCIe コントローラは，Intel が自社 FPGA 向けに提供している “Arria 10 Hard IP for PCI Express Avalon-MM with DMA” の IP を利用している．この IP には DMA コントローラ (DMAC: DMA Controller) が内蔵されており，DMAC に対してディスクリプタを書き込むことによって，DMA 転送が行われる．ディスクリプタは**表 1** に示すように特定の形式に従って DMA 転送に必要なデータが格納されている．Source は DMA 転送元 PCIe アドレス，Destination は DMA 転送先 PCIe アドレス，DMA Length は転送長 (ワード単位)，DMA Descriptor ID は転送完了時にどの転送が完了したかを判別するために用いる ID である．このディスクリプタ内の Source や Destination の Address に前節で述べた PCIe アドレス空間にマップされた GPU メモリアドレスをセットすることにより，FPGA

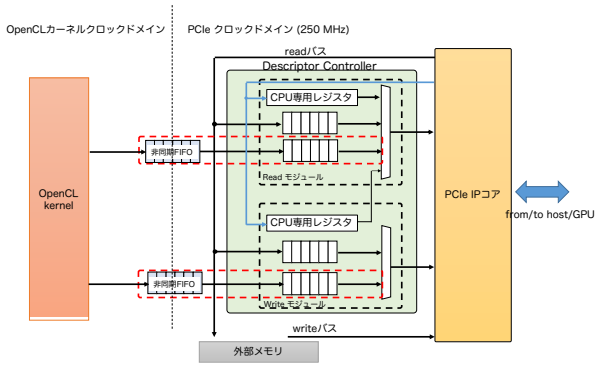

図 6: ディスクリプタコントローラの構成図．赤色の破線で囲まれたコンポーネントを加えることにより OpenCL カーネルからディスクリプタコントローラを操作し，ディスクリプタを DMAC に書き込むことができる．

は PCIe DMAC を用いて GPU デバイスメモリからのデータ読み出しや GPU デバイスメモリへのデータ書き込みを実行できる．本稿では，PCIe アドレス空間にマップされた GPU メモリアドレスを OpenCL API によって FPGA に送信し，FPGA (OpenCL カーネル) は，受信したアドレス情報を元にディスクリプタを生成し，それを DMAC に書き込むことによって，GPU-FPGA 間データ転送を実行する．

3.3 ディスクリプタの書き込み

図 6 に DMAC にディスクリプタを書き込むためのモジュールであるディスクリプタコントローラの構成図を示す．この手法は，OpenCL カーネル内で生成したディスクリプタを I/O Channel API (write_channel_intel 関数) を介してこのモジュールに渡し，ディスクリプタコントローラが受け取ったディスクリプタを DMAC に書き込むことによって GPU-FPGA 間データ転送を実行している．ただし，CPU もホスト-FPGA 間で OpenCL API を用いた DMA 転送 (clEnqueueReadBuffer や clEnqueueWriteBuffer) を実行するためにディスクリプタコントローラを操作する．したがって，それに競合しないように OpenCL カーネルからディスクリプタコントローラに対してアクセスする必要がある．以下にディスクリプタコントローラの動作について述べる．

ディスクリプタコントローラは FPGA からデータを送信するためのディスクリプタを DMAC に書き込むための Write モジュール，データを受信するためのディスクリプタを書き込むための Read モジュールから構成され，それぞれのモジュールは CPU のみがアクセスできるレジスタ，ホスト-FPGA 間の DMA 転送を実行するためのディスクリプタを格納するための FIFO を有する．ホスト-FPGA 間で DMA データ転送を実行する場合，CPU はまず PIO

```
1  #pragma OPENCL EXTENSION cl_intel_channels : enable
2
3  typedef struct __attribute__((packed)) cldesc {
4    ulong src;
5    ulong dst;
6    uint  id_and_len;
7    uint  unused0;
8    uint  unused1;
9    uint  unused2;
10 } cldesc_t;
11
12 channel cldesc_t fpga_dma __attribute__((depth(0)))
       __attribute__((io("chan_fpga_dma")));
13 channel ulong dma_stat __attribute__((depth(0)))
       __attribute__((io("chan_dma_stat")));
14
15 __kernel void fpga_dma(__global float *restrict fpga_mem,
16                         const ulong gpu_memadr,
17                         const uint id_and_len)
18 {
19   cldesc_t desc;
20   // DMA transfer GPU -> FPGA
21   desc.src = gpu_memadr;
22   desc.dst = (ulong)(&fpga_mem[0]);
23   desc.id_and_len = id_and_len;
24   write_channel_intel(fpga_dma, desc);
25   ulong status = read_channel_intel(dma_stat);
26 }
```

図 7: GPU から FPGA への DMA 転送を実行する OpenCL カーネルコード.

(Programable IO) アクセスによって，Read モジュール，もしくは Write モジュール内にあるレジスタを操作し，その DMA 転送を実行するためのディスクリプタをホストメモリから FPGA にロードするためのディスクリプタを生成する．そのディスクリプタを Read モジュールから DMAC に書き込むことによって，DMA 転送を実行するためのディスクリプタはホストメモリから読み出され，Read モジュール，もしくは Write モジュール内の FIFO に格納される．その後，FIFO に格納されたディスクリプタを DMAC に書き込むと，ホスト-FPGA 間で DMA データ転送が実行される．

これらの動作を妨げることなく OpenCL カーネルコードから GPU-FPGA 間 DMA データ転送を実行するためには，Read モジュール，Write モジュール内に GPU-FPGA 間 DMA データ転送を実行するためのディスクリプタを格納する FIFO を用意し，プライオリティエンコーダによってそれぞれのモジュールからのディスクリプタの発行を適切に排他制御すれば良い．それらを実行するために図の赤色の破線で囲まれたコンポーネントを Verilog HDL で実装し，ディスクリプタコントローラに付け加えた．なお，OpenCL カーネルとディスクリプタコントローラのクロックドメインは異なるため，OpenCL カーネルコードからディスクリプタコントローラにディスクリプタを送信するためには非同期 FIFO が必要となる．そして [7] と同様に，BSP 内のハードウェアコンポーネントと OpenCL カーネルコードとを関連付けている board_spec.xml を適切に編集することによって，GPU-FPGA 間データ転送を実行する OpenCL カーネルコードを記述することが可能となる．

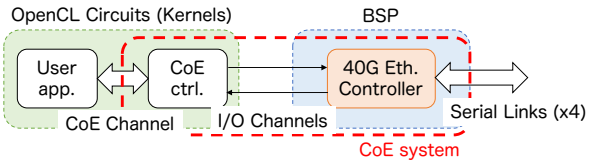

図 8: Channel over Ethernet (CoE) システムの概略図.

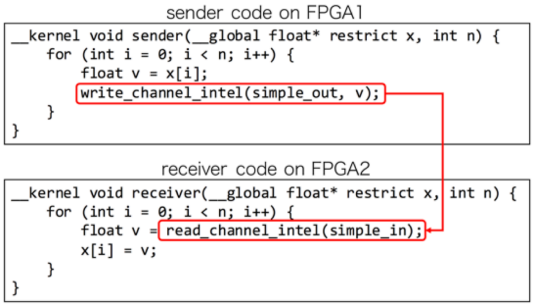

図 9: CoE を用いて通信を行うコード例.

3.4 GPU-FPGA DMA コード例

図 7 は GPU から FPGA への DMA 転送を実行する OpenCL カーネルコードであり，1 行目の pragma は Intel FPGA SDK for OpenCL の独自拡張である channel の有効化をコンパイラに指示するためのものであり，3 ～ 10 行目で DMA コントローラに書き込むためのディスクリプタの構造体を，12，13 行目で I/O Channel 変数である fpga_dma と dma_stat を定義している．GPU から FPGA への DMA 転送なので，ディスクリプタの Source に PCIe アドレス空間にマップした GPU メモリアドレスである gpu_memadr を，Destination に FPGA 外部メモリアドレス (fpga_mem) をセットしている．また，0~127 の id はホスト CPU が利用しているため，OpenCL カーネルで生成されるディスクリプタの id は 128~255 としている．生成されたディスクリプタは write_channel_intel 関数によって，ディスクリプタコントローラにおける Read モジュールに送信され，モジュール内の FIFO でバッファリングされる．その後，適切なタイミングで DMA コントローラに書き込まれ，GPU から FPGA への DMA 転送が実行され，その DMA 転送に要したサイクル数が read_channel_intel 関数を介して，OpenCL カーネルコード内で読み出される．

4. OpenCL から制御可能な FPGA 間通信技術

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 [5]．**図 8** に示すように，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

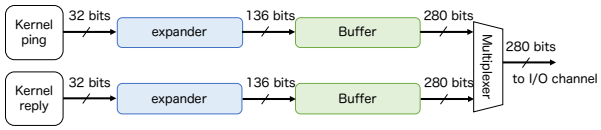

図 10: CoE 送信回路の構成.

の境界にある Channel である “CoE Channel” から構成される.

CoE を用いて通信する際の簡単な例を**図 9** に示す．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ためである.

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 10** に示す．この図は pingpong 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，送信用のカーネル ping，返信用のカーネル reply がある．なお，CoE システムは一対一通信だけでなく集団通信もサポートしているが，集団通信の機能を用いた GPU-FPGA 連携は未実装であるため，本稿では集団通信の説明は割愛する.

CoE の送信回路は以下に示す 3 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.

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 システムはいくつかの送信モードを持ち，図 10 に示す通信は，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バッファリングモードである．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.

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 にデータを分別するカーネルと，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 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.

[*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.

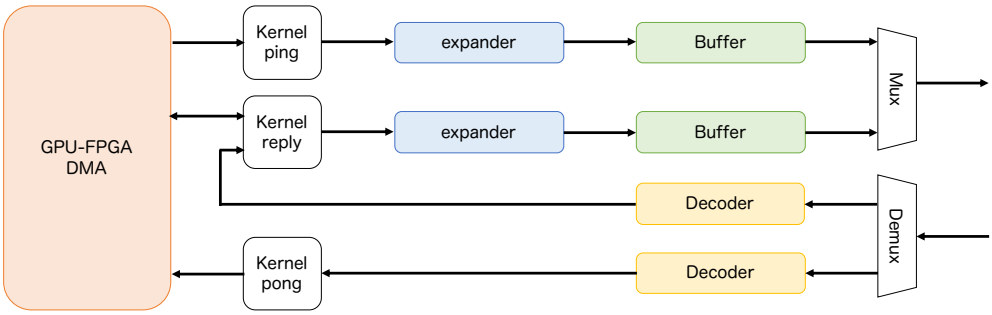

図 11: ノードを跨いだ GPU 同士の pingpong ベンチマークを実行する OpenCL カーネル群.

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おり，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．すなわち，通信能力は 32bit/cycle 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. ノードを跨いだ GPU 同士の pingpong ベンチマークの実装

これまでに述べた機能を用いて，ノードを跨いだ GPU 同士の pingpong ベンチマークを実装する．すなわち，ping 通信は，ノード 0 に搭載される GPU から FPGA に OpenCL から制御可能な GPU-FPGA 間データ転送 [4] の機能を用いてデータを転送し，その後，FPGA 間通信技術 [5] によってノード 1 に搭載される FPGA にデータを送信，最後にノード 1 に搭載される GPU に対して FPGA から DMA することで実現される．

図 11 にノードを跨いだ GPU 同士の pingpong ベンチマークを実行する OpenCL カーネル群を示す．図 10 に示されているカーネル群に加え，pong 通信を受信する Kernel pong，受信した Ethernet パケットからデータを取り出す Decoder，ping 通信/pong 通信を識別してスイッチングする Demux，そして GPU とデータ転送を実行するための GPU-FPGA DMA カーネルを OpenCL で実装した．

GPU-FPGA DMA カーネルは，カーネルの起動時にホスト CPU から PCIe アドレス空間にマップされた GPU メモリのアドレスを受け取る．その後，Kernel ping，Kernel reply，Kernel pong から，DMA 要求を受け取り，それらを特定のスケジューラにしたがって処理する．

このようにすることで，CPU の干渉を極力抑えたまま，複数ノード上における GPU と FPGA とを連携させることが可能であり，pingpong ベンチマークが実際に正しく動作していることを確認できた．以降の章にて，pingpong ベンチマークを実行した場合の通信レイテンシの評価について述べる．

表 2: 評価環境 (PPX)

CPU	Intel Xeon E5-2660 v4 × 2
CPU Memory	DDR4 2400MHz 64GB (8GB × 8)
GPU	NVIDIA Tesla P100 × 2 (PCIe Gen3 x16 card version)
GPU Memory	16 GiB CoWoS HBM2 @ 732 GB/s with ECC
InfiniBand	Mellanox ConnectX-4 EDR
Host OS	CentOS 7.3
Host Compiler	gcc 4.8.5
GPU Compiler	CUDA 9.1.85
MPI	OpenMPI 3.0.1
OpenCL SDK	Intel FPGA SDK for OpenCL 17.1.2.304
FPGA	BittWare A10PL4 (10AX115N3F40E2SG)
FPGA Memory	DDR4 2133MHz 8GB (4GB × 2)
Communication Port	QSFP+ × 2 (40Gbps × 2)
Ethernet Switch	Mellanox MSN2100

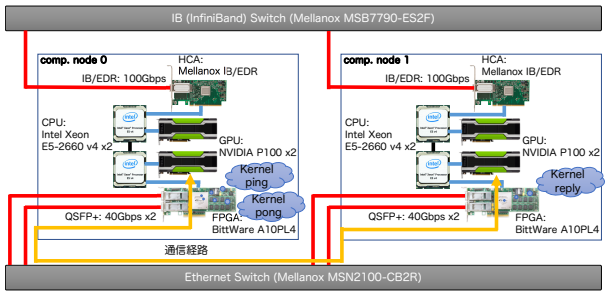

図 12: ノードを跨いだ GPU 同士の pingpong ベンチマークの概要.

6. 評価

6.1 評価環境

通信レイテンシの観点における提案手法の評価には，筑波

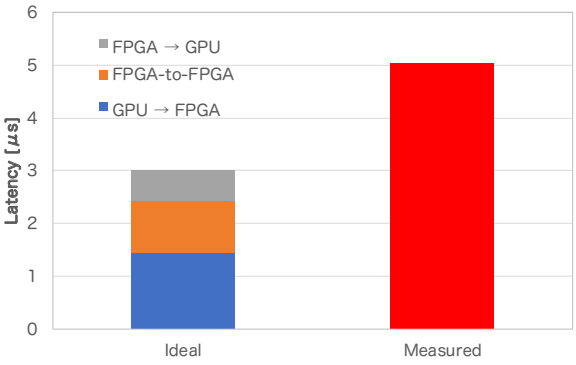

図 13: ノードを跨いだ GPU 同士の pingpong ベンチマークにおける通信レイテンシ．Ideal は [4], [5] の結果をベースとしている．

大学計算科学研究センターで運用中の Pre-PACS version X (PPX) クラスタシステムを用いる．PPX は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り，Intel FPGA ノードグループ，Xilinx FPGA ノードグループの 2 グループから構成される．Intel FPGA と Xilinx FPGA は FPGA プラットフォーム比較用に導入され，それらの FPGA をそれぞれ搭載したノードを一体運用しているが，この評価では Intel FPGA のみを利用している．そのため，本節では Intel FPGA を搭載するノードのみの詳細について述べ，それを**図 2** に示す．ノードには，Intel Xeon E5-2660 v4 CPU × 2，NVIDIA Tesla P100 GPU × 2，Mellanox InfiniBand ConnectX-4 EDR HCA × 1，BittWare A10PL4 FPGA ボード × 1 が搭載されており，CPU-GPU 間は PCIe Gen3 x16 レーンにて，CPU-FPGA 間は FPGA ボードの仕様のため PCIe Gen3 x8 レーンにてそれぞれ接続されている．

本稿の性能評価は PPX 2 ノードを用いて行い，CPU，GPU，FPGA は同一ソケット内に配置されているものを利用する．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

図 12 に，提案手法を用いて実現されるノードを跨いだ GPU 同士の pingpong ベンチマークを示す．この pingpong ベンチマークでは，初期データはノード 0 の GPU メモリに cudaMemcpy によって書き込まれ，それは FPGA を介することによって，ノード 1 の GPU に送信される．その後，ノード 1 の GPU で受信したデータは，FPGA を経由してノード 0 の GPU に返信される．これを実現するために，ノード 0 の FPGA では，Kernel ping と Kernel pong が起動しており，ノード 1 の FPGA では，Kernel reply が起動している．なお，起動するカーネルは，MPI の rank に応じて選択している．

通信レイテンシの測定は，通信開始時 t0 と通信終了時 t1 の 2 箇所を OpenCL カーネルの動作クロック精度で時間を測定し，$\frac{t1-t0}{2}$[s] の計算式によって求められる．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．

4 バイト通信時の最小レイテンシは 5.04μsec であり，理論性能との比較を**図 13** に示す．なお，今回測定に使用した実装は 222.5MHz で動作しており，1 クロックサイクルの時間は 4.49ns である．理論性能の Ideal は [4], [5] の結果をベースとしており，3.02μsec である．このため，理論性能と比べて 2.02μsec 余分にかかっていることが分かる．しかしこれは，現在の実装が最適化されていないことに大きく依存している．現在の実装では，CoE システムと GPU-FPGA DMA との間は完全なストアアンドフォワード方式でデータを渡さなければならない．すなわち，Kernel ping，Kernel pong，Kernel reply では，メモリコピーが必ず発生しており，これはレジスタでデータの送受信を行うようにすることで解消できる．そのようにすることで，レイテンシの改善が見込める．

7. おわりに

本稿では，OpenCL から制御可能な FPGA 間通信技術と GPU-FPGA 間 DMA 転送技術を組み合わせた，複数ノード上における GPU-FPGA 間連携手法を提案した．GPU-FPGA 間 DMA 転送は，GPU デバイスのグローバルメモリを PCIe アドレス空間にマップし，アドレスマップの結果をベースに OpenCL カーネル内で作成したディスクリプタを最終的に FPGA 内の PCIe DMA コントローラに書き込むことによって実現される．また，FPGA 間通信は，Verilog HDL で実装された Ethernet 通信を実行するハードウェアと，そのハードウェアの制御モジュール (OpenCL カーネル) を I/O Channel で接続することによって構成されている CoE システムで実現される．

提案手法を用いてノードを跨いだ GPU 同士の pingpong ベンチマークを実装し，それが正しく動作していることを確認した．通信レイテンシを評価したところ，5.04μsec でであり，理論性能と比べて 2.02μsec 余分にかかっていることが分かった．しかしこれは，現在の実装が最適化されていないことに大きく依存しているため，通信レイテンシを改善する余地は十分残されている．

今後の研究においては，提案手法の実アプリへの適用とその性能評価を行い，それと同時に GPU と FPGA を効率的に同期させる機構や GPU-FPGA 複合システムにおけるデバイス間連携を統合的にホスト CPU から制御するためのソフトウェア的枠組みについて検討していく．

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」，文部科学省研究予算「次世代計算技術開拓による学際計算科学連携拠点の創出」，及び科学研究費補助金一般 (B)「再構成可能システムと GPU による複合型高性能プラットフォーム」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 社の支援に謝意を表する．

参考文献

[1] Hanawa, T., Kodama, Y., Boku, T. and Sato, M.: Interconnection Network for Tightly Coupled Accelerators Architecture, *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*, pp. 79–82 (online), DOI: 10.1109/HOTI.2013.15 (2013).

[2] Kuhara, T., Tsuruta, C., Hanawa, T. and Amano, H.: Reduction calculator in an FPGA based switching Hub for high performance clusters, *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–4 (online), DOI: 10.1109/FPL.2015.7293985 (2015).

[3] Tsuruta, C., Miki, Y., Kuhara, T., Amano, H. and Umemura, M.: Off-Loading LET Generation to PEACH2: A Switching Hub for High Performance GPU Clusters, *SIGARCH Comput. Archit. News*, Vol. 43, No. 4, pp. 3–8 (online), DOI: 10.1145/2927964.2927966 (2016).

[4] 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 GPU-FPGA デバイス間連携，技術報告 2018-HPC-167 (2018).

[5] 藤田典久，小林諒平，山口佳樹，朴 泰祐：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，技術報告 2018-HPC-167 (2018).

[6] Overview: Intel FPGA SDK for OpenCL, https://www.altera.com/products/design-software/embedded-software-developers/opencl/overview.html.

[7] 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).