Vol.2018-HPC-167 No.11
2018/12/17

OpenCLとVerilog HDLの混合記述による GPU-FPGAデバイス間連携

小林 諒平[1,2]　藤田 典久[1]　山口 佳樹[2,1]　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，GPU プログラミングと FPGA プログラミングの連携を効率的に行うためのデバイス間データ転送について提案する．GPU デバイスメモリの PCIe アドレスマッピング結果をベースに作成されたディスクリプタを FPGA に送信し，FPGA 内の PCIe DMA コントローラに書き込むことによって，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．通信レイテンシと通信バンド幅の観点から提案手法を評価した結果，従来手法と比較して，通信レイテンシの面では最大で 83 倍の性能差，通信バンド幅の面では最大で 2.4 倍の性能差が確認された．

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ており，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，GPU 間低レイテンシ通信がシステム上にて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．例えば PEACH2 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 にオンザフライ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 [2], [3]．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

[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

© 2018 Information Processing Society of Japan

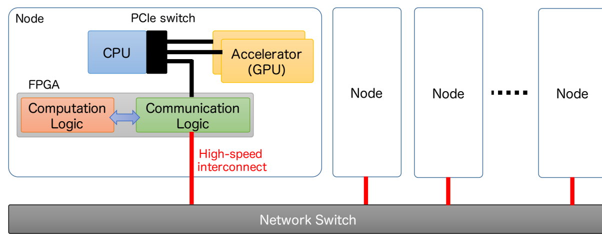


図 1: AiS コンセプトの概要．GPU では粗粒度並列処理を担当する計算カーネルが実行され，FPGA では GPU が不得手とする演算や集団通信を含む高速ノード間通信を担当するカーネルが実行される．CPU はこれらのカーネルの起動および全計算デバイスの調停を行う．

となる．そこで我々は，GPU プログラミングと FPGA プログラミングの連携を効率的に行うためのデバイス間データ転送について提案している [4]．本稿では，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 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について述べる．開発した通信機能を CPU を介する従来手法と比較した結果，最大で 83 倍の性能差，通信バンド幅の面では最大で 2.4 倍の性能差が確認された．

以下に本稿の構成を示す．第 2 章で我々が開発した GPU-FPGA 間通信機能について述べ，第 3 章でその機能を OpenCL カーネルコードから制御する手法について述べる．第 4 章にて通信レイテンシと通信バンド幅の評価を述べ，最後に第 5 章で本稿をまとめる．

2. GPU-FPGA 間通信

図 **2** に，提案する GPU-FPGA 間通信手法の概要を示す．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，我々の提案手法では **FPGA が主体的に 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．

- ホスト CPU 側での設定
  (1) 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させる
  (2) PCIe アドレス空間にマップされた GPU デバイスのグローバルメモリアドレス情報を FPGA に送信

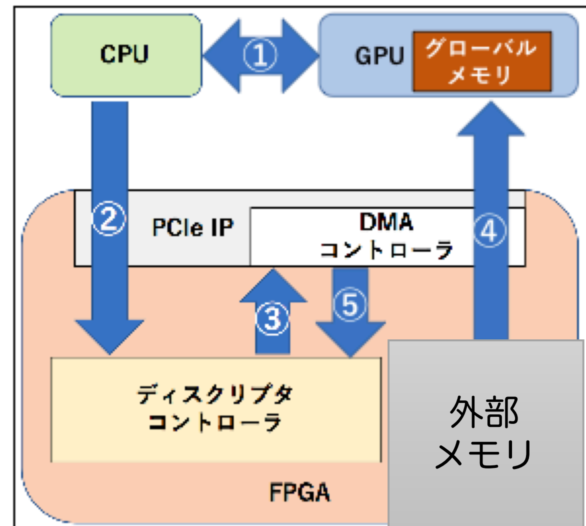


図 2: 開発した GPU-FPGA 間通信機能の概要．PCIe DMA 転送用の IP コアを用いて，デバイス間データ転送を行う．

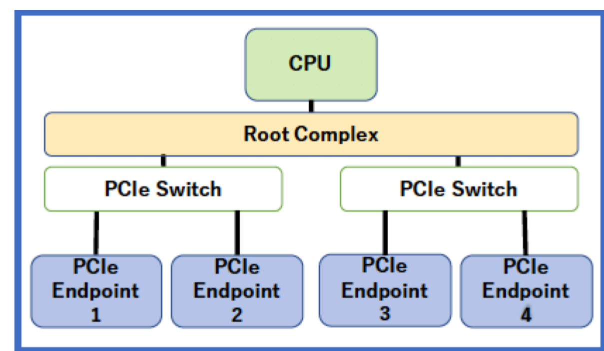


図 3: CPU と PCIe デバイスの接続．

- FPGA 側での設定
  (3) ホストから受け取った GPU メモリアドレス情報を元にディスクリプタを生成し，それを DMA コントローラに書き込む
  (4) デバイス間データ転送が実行される
  (5) 完了信号が発行される

2.1 PCIe アドレスマッピング

PCIe とは，計算機において CPU と周辺機器 (PCIe デバイス) を接続するためのシリアル通信を用いたバスであり，その接続図の例を図 **3** に示す．PCIe バスを介して接続される周辺機器は Endpoint と呼ばれ，HPC 分野では GPU デバイスや InfiniBand HCA (Host Channel Adapter) が Endpoint としてよく用いられる．各 PCIe デバイスは，PCIe バスを通じて CPU や他の PCIe デバイスとの通信が可能である．そして拡張性が足りない場合は，PCIe Switch を用いることによって，より多くの Endpoint を接続することが可能である．

© 2018 Information Processing Society of Japan

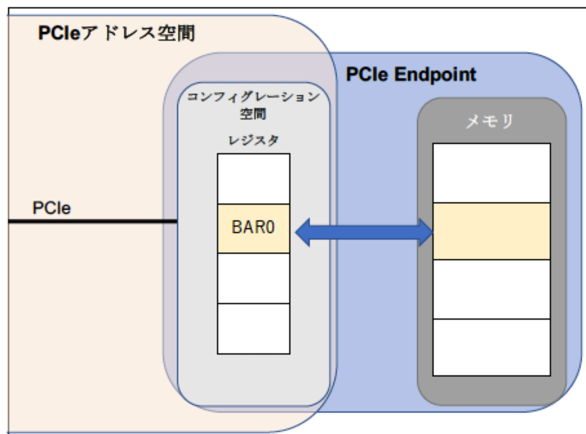


図 4: PCIe アドレス空間

```
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 アドレス空間への GPU メモリのマップ

図 4 に PCIe アドレス空間の概要を示す．PCIe Endpoint は PCIe アドレス空間を公開しており，PCIe アドレス空間にアクセスすることによって PCIe デバイスに対する操作を行うことができる．PCIe Endpoint はそれぞれが独自にコンフィギュレーション空間を持ち，PCIe デバイスとしての各種設定 (デバイス ID など) を保持している．コンフィギュレーション空間には BAR (Base Address Register) と呼ばれるレジスタが 6 個 (これを BAR0〜BAR5 と呼ぶ) あり，このレジスタにそのデバイスのメモリが PCIe アドレス空間のどこに配置されているかが格納される．BAR に値を設定するのは OS やデバイスドライバの仕事であり，ほかのデバイスと衝突しない PCIe アドレス空間が割り当てられ，デバイスの識別に用いられる．ある PCIe デバイスの BAR0 が割り当てられているアドレス空間にメモリアクセスを行うと，そのデバイスへのアクセスと判断され処理される．

GPU のメモリを PCIe アドレス空間からアクセスするには，NVIDIA が提供している 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する．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，前述した API も GPU ドライバに実装されている．したがって，FPGA から直接通信を行う場合であっても，まず CPU 上で前述したメモリ API を用いて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 で用いていたカーネルモジュールおよびライブラリを測定環境へと移植し用いる．HA-PACS/TCA と PPX では，OS のカーネルバージョン及び NVIDIA のドライバのバージョンが異なっているため，幾つかのマクロ等を変更している．

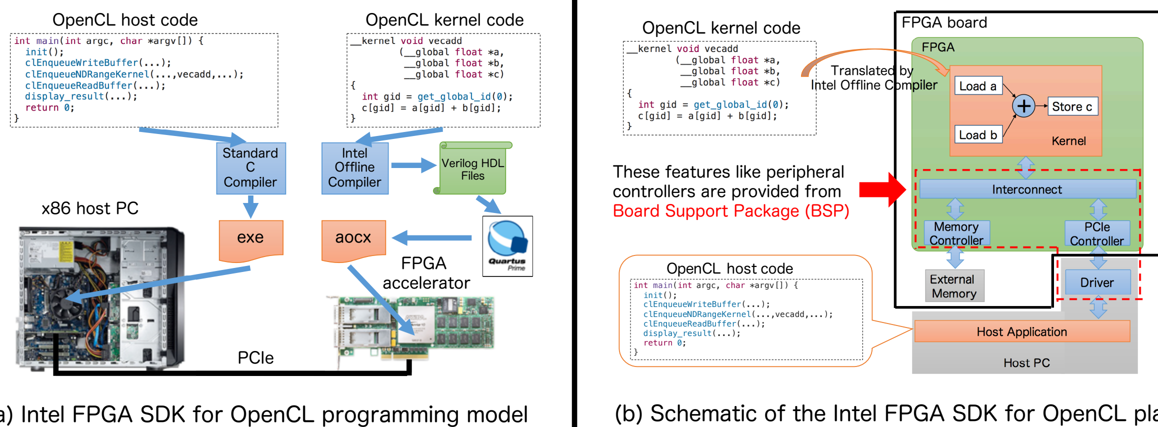
PEACH2 で用いていた API を用いた PCIe アドレス空間への GPU メモリのマップ方法を図 5 に示す．PEACH2 の API である tcaCreateHandleGPU() 関数にホスト側で作成したポインタを渡すことにより，PCIe アドレス空間にマップされた GPU メモリのアドレスである paddr を知ることができる．この関数は，もともと PEACH2 の通信対象とするメモリ領域を識別するためのハンドルを作成する関数であるが，内部的には前述した NVIDIA が提供する Kernel API を用いて GPU アドレスを PCIe アドレスにマップしそのアドレスを取得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

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

2.2 ディスクリプタの発行

FPGA の PCIe 接続には，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 を用いる．この IP には DMA コントローラが内蔵されており，DMA コントローラ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)，



(a) Intel FPGA SDK for OpenCL programming model

(b) Schematic of the Intel FPGA SDK for OpenCL platform

図 6: Intel FPGA SDK for OpenCL の概要．(a) はプログラミングモデル，(b) は Intel FPGA SDK for OpenCL プラットフォームの構成図を示す．

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 は PCIe DMA コントローラを用いて 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．

FPGA の PCIe DMA コントローラにディスクリプタを書き込むために，[4] ではホストであらかじめディスクリプタを生成し，それを幾つかのチャンクに分けて FPGA に送信した後，FPGA 内でそれらをアセンブルしてディスクリプタを再構築し，DMA コントローラに書き込んでいる．それに対し本稿では，ホストは PCIe アドレス空間にマップされた GPU メモリアドレス情報の送信のみを行い，ディスクリプタの構築および DMA コントローラへの書き込みは全て OpenCL カーネル内で実行される．すなわち，既存研究と比較してホストの干渉を極力抑え，かつ GPU-FPGA 間 DMA 転送をユーザーレベルから制御することができる．これは，Intel が提供する OpenCL ツールチェインの仕組みと Verilog HDL とを用いることで実現される．以降にその詳細について述べる．

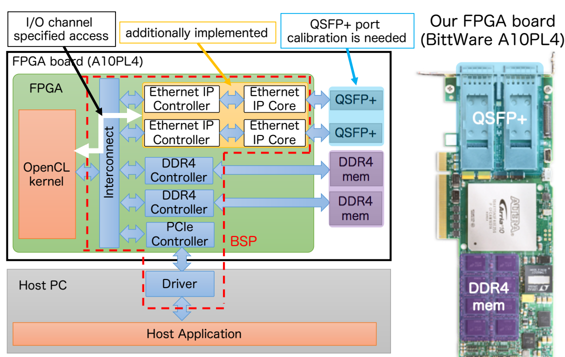
3. OpenCL による制御手法

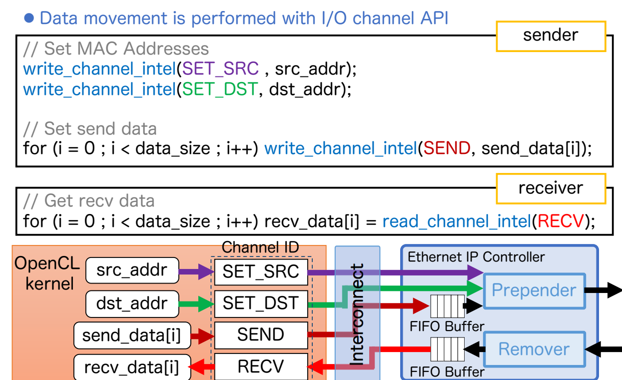
3.1 Intel FPGA SDK for OpenCL

Intel は OpenCL を用いて FPGA 回路を設計できる Intel FPGA SDK for OpenCL[5] という開発環境を提供しており，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 **6** (a) 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 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 6 (b) 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．CPU とは異なり，FPGA デバイスの外部ペリフェラルの構成は FPGA ボード毎に異なる．ボード間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，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

© 2018 Information Processing Society of Japan





(a) ハードウェア実装の概略図 (b) Ping-pong通信用のOpenCLコードの一部

図 7: QSFP+コントローラを I/O Channel API を介して OpenCL カーネルコードから制御する手法の概要

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．

ただし，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．そのため，FPGA ボードに搭載されている QSFP+や QSFP28 などのネットワークポートや本稿のようなユーザが独自開発したハードウェアモジュールを OpenCL カーネルコードからアクセスできるようにするためには，BSP を適切に改変する必要がある．

3.2 I/O Channel を用いた外部ペリフェラル操作

Intel FPGA SDK for OpenCL では，ユーザが独自開発したハードウェアモジュールを OpenCL カーネルコードからアクセスするためのベンダー拡張機能である I/O Channel API が提供されており，我々はこの API を用いた OpenCL と Verilog HDL の混合記述による FPGA プログラミングに関する研究をこれまでに行ってきた [6], [7], [8]．例えば，[8] では，QSFP+ポートを用いて高速イーサネット通信を実行するハードウェアモジュールを I/O Channel API を介して OpenCL カーネルコードから制御する手法について述べており，その概要を図 **7** に示す．

前述したように，Intel FPGA SDK for OpenCL を利用するためには，BSP と呼ばれる FPGA チップと外部ペリフェラル間の接続情報を記述したライブラリが必要であり，図 7 (a) に示す外部メモリコントローラや PCIe コントローラ，PCIe 通信用のソフトウェアドライバが提供される．しかしながら，BSP は一般的に OpenCL のプログラムを実行するために必要な最低限のインターフェースについてのみを提供するため，例えば QSFP+のような通信ポートの使用を望む場合，QSFP+ポートを制御するためのロジックを実装し，それを BSP に追加する必要がある．図に示す黄色のブロックが追加したコンポーネントであり，それを I/O Channel API を通して，OpenCL カーネルコードから制御する．これにより，OpenCL カーネルから高速イーサネット通信を制御することが可能となり，その実証実験として 2 つの FPGA 間で Ping-pong 通信を実行するカーネルを OpenCL で記述した．図 7 (b) にそのカーネルコードの一部を示す．write_channel_intel() によって，OpenCL カーネルから自作モジュールを介してのデータ送信，read_channel_intel() によってデータの受信を行っている．

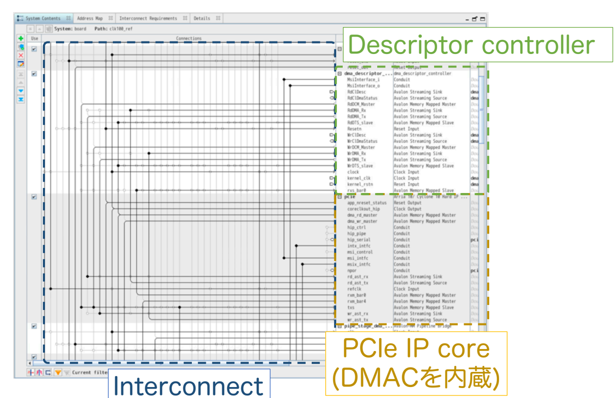


図 8: ディスクリプタコントローラを外部にインスタンス化する様子.

すなわち本稿では，Intel FPGA SDK for OpenCL に関するこれらの仕組みを活用し，提案するデバイス間データ転送を実行する機能を BSP に組み込み，FPGA 主体の GPU-FPGA 間 DMA 転送を I/O Channel API を介して OpenCL カーネルコードから制御する．

3.3 デバイス間データ転送機能の組み込み

本節では，我々の提案するデバイス間データ転送を実行

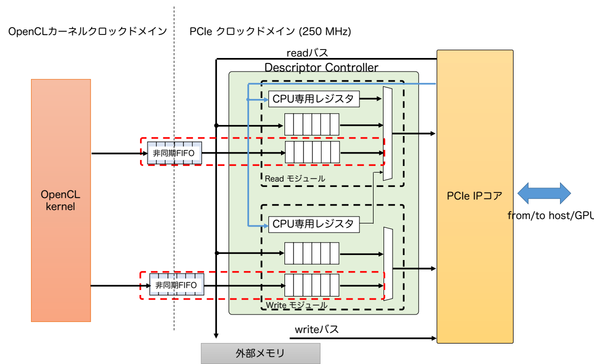


図 9: ディスクリプタコントローラの構成図．赤色の破線で囲まれたコンポーネントを加えることによりOpenCL カーネルコードからディスクリプタコントローラを操作し，ディスクリプタを DMA コントローラに書き込むことができる．

する機能を BSP に統合し，それを OpenCL カーネルコードから制御する手法について述べる．BSP に同梱されている PCIe コントローラと提案手法で用いられているものは同じ ‘ ‘Arria 10 Hard IP for PCI Express Avalon-MM with DMA” の IP コアを利用している．すなわち，IP コアに内蔵されている DMA コントローラに対し，OpenCL カーネルコードからディスクリプタを書き込むことができれば，[4] と同様のデータ転送を実行できる．ただし，[4] では DMA コントローラ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を GPU-FPGA 間 DMA 転送するためだけに専有できていたのに対し，本手法では CPU とシェアする形でディスクリプタコントローラを利用しなければならない．CPU はホスト-FPGA 間で 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ため，それに競合しないようにOpenCL カーネルコード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．また，BSP に同梱されている PCIe コントローラはデフォルトではディスクリプタコントローラをホストしかアクセスできないように隠蔽している．したがって，まずはディスクリプタコントローラを外部にインスタンス化し，OpenCL カーネルコードからもアクセスできる状態にしなければならない．その様子を図 **8** に示す．外部にインスタンス化したディスクリプタコントローラに I/O Channel で利用される Avalon-ST インターフェースを付け加えることによって，OpenCL カーネルコードからのアクセスが可能となる．

図 **9** に外部にインスタンス化したディスクリプタコントローラの構成図を示す．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 DMA コントローラに書き込むための 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

```
1  #pragma OPENCL EXTENSION cl_intel_channels :
       enable
2
3  typedef struct cldesc {
4    uint x0, x1, x2, x3, x4, x5, x6, x7;
5  } cldesc_t;
6
7  channel cldesc_t fpga_dma_r __attribute__((depth
       (0))) __attribute__((io("rdcldesc")));
8  channel ulong dma_r_status __attribute__((depth
       (0))) __attribute__((io("rdcldmastatus")));
9
10 __attribute__((reqd_work_group_size(1,1,1)))
11 __kernel void fpga_dma(
12                        __global uint *restrict
                               RECV_DATA,
13                        __global const uint *
                               restrict SEND_DATA,
14                        __global ulong *restrict
                               E_CYCLE,
15                        __global const uint *
                               restrict NUMBYTE,
16                        const ulong PADDR,
17                        const uint DEBUG_MODE
18                       )
19 {
20
21   cldesc_t desc;
22   ulong    elapsed_cycle = 0;
23
24   ulong src = PADDR;
25   ulong dst = (ulong)RECV_DATA;
26   uint  len = *NUMBYTE;
27   uint   id = ((1 << 7) | (SEND_DATA[0] & 0x7f));
       // id must be set to 128 ~ 255
28   desc.x0 = src & 0xfffffffful;
29   desc.x1 = (src >> 32) & 0xfffffffful;
30   desc.x2 = dst & 0xfffffffful;
31   desc.x3 = (dst >> 32) & 0xfffffffful;
32   desc.x4 = ((id & 0xff) << 18) | ((len >> 2) & 0
       x3ffff);
33   desc.x5 = 0;
34   desc.x6 = 0;
35   desc.x7 = 0;
36   write_channel_intel(fpga_dma_r, desc);
37   elapsed_cycle = read_channel_intel(dma_r_status
       );
38
39   *E_CYCLE = elapsed_cycle;
40 }
```

図 10: GPU から FPGA への DMA 転送を実行する OpenCL カーネルコード

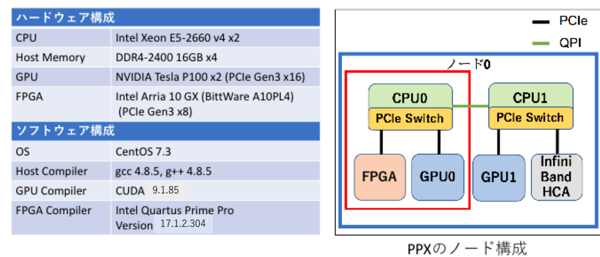
の Read モジュールから構成され，それぞれのモジュールは CPU のみがアクセスできるレジスタ，ホスト-FPGA 間の DMA 転送を実行するためのディスクリプタを格納するための FIFO を有する．ホスト-FPGA 間で DMA データ転送を実行する場合，CPU はまず PIO (Programable IO)

© 2018 Information Processing Society of Japan

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は Read モジュールから DMA コントローラに書き込まれ，ホストメモリに格納されているディスクリプタを FPGA にロードするための DMA が起動し，ホストメモリから読み出されたディスクリプタは read バスを通って，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がデキューされると DMA コントローラにそれが書き込まれ，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．

これらのハードウェアコンポーネントを付け加え，board_spec.xml を適切に編集する [6], [7], [8] ことによって，**図 10** に示すような OpenCL カーネルコードによって GPU-FPGA 間の DMA 転送を制御することが可能となる．図 10 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，3～5 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，7，8 行目で I/O Channel 変数である fpga_dma_r と dma_r_status 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 PADDR を，Destination に FPGA 外部メモリアドレス (RECV_DATA) 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．本稿における通信レイテンシと通信バンド幅の評価にはこのサイクル数を使う．



ハードウェア構成	
CPU	Intel Xeon E5-2660 v4 x2
Host Memory	DDR4-2400 16GB x4
GPU	NVIDIA Tesla P100 x2 (PCIe Gen3 x16)
FPGA	Intel Arria 10 GX (BittWare A10PL4) (PCIe Gen3 x8)
ソフトウェア構成	
OS	CentOS 7.3
Host Compiler	gcc 4.8.5, g++ 4.8.5
GPU Compiler	CUDA 9.1.85
FPGA Compiler	Intel Quartus Prime Pro Version 17.1.2.304

図 11: 評価環境および PPX システムの計算ノードの構成図．評価には赤色の枠線で囲まれたデバイスを用いた．

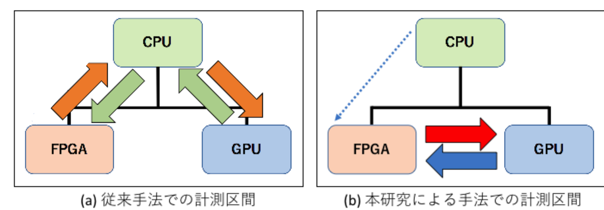


図 12: 通信時間の計測区間．

4. 評価

4.1 評価環境

通信レイテンシと通信バンド幅の観点における提案手法の評価には，筑波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 11** に示す．ノードには，Intel Xeon E5-2660 v4 CPU × 2，NVIDIA 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．図 11 に示すように，この評価では同一ソケットにおける GPU デバイスと FPGA デバイス (赤色の枠線で囲まれた部分) を用い，GPU-FPGA 間データ転送を行っている．

図 12 に，従来手法と提案手法における GPU-FPGA 間データ転送の通信経路を示す．従来手法における GPU-

© 2018 Information Processing Society of Japan

表 2: GPU-FPGA 間通信レイテンシの比較.

FPGA ← GPU の通信	
従来手法	17 μsec
本研究による手法	1.08 μsec
FPGA → GPU の通信	
従来手法	20 μsec
本研究による手法	0.24 μsec

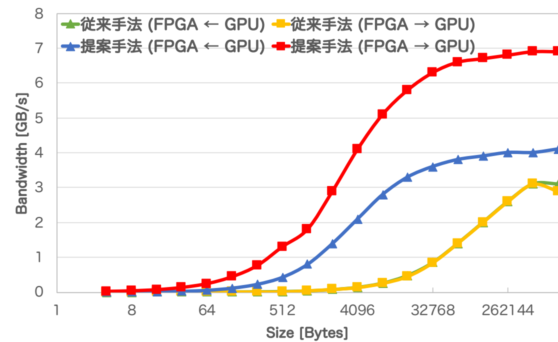


図 13: GPU-FPGA 間通信バンド幅の比較.

FPGA 間データ転送は，1. CPU-FPGA 間，2. CPU-GPU 間に分かれている．すなわち，1 では OpenCL API を用いて，2 では cudaMemcpy によってデータ転送が実行され，評価では最初の送信元から最後の宛先に届くまでに要した時間を chrono ライブラリの high_resolution_clock によって測定した．一方，提案手法では，GPU デバイスメモリの PCIe アドレスマッピング結果をベースに OpenCL カーネル内でディスクリプタを作成し，FPGA 内の PCIe DMA コントローラに書き込むことによって，FPGA が主体的に GPU-FPGA 間のメモリコピーを実行する．FPGA 内の PCIe DMA コントローラに対してディスクリプタの書き込みが完了した時点から DMA 転送の完了信号を受信するまでに要した時間を計測した．なお，図中の破線矢印は CPU から FPGA に対しての PCIe アドレス空間にマップされた GPU メモリアドレス情報の送信を表しており，評価ではその送信に要する時間は含まないことに留意して頂きたい．

4.2 通信レイテンシ

表 2 に，従来手法と提案手法による GPU-FPGA 間データ転送の通信レイテンシを示す．FPGA の PCIe IP に内蔵されている DMA コントローラが転送できる最小データサイズが 4 バイトであるため，通信レイテンシの測定におけるデータサイズは 4 バイトとした．GPU から FPGA へのデータ転送における通信レイテンシは，従来手法を用いた場合では 17 μsec，提案手法を用いた場合では 1.08 μsec となり，15.7 倍の性能差が確認された．また，FPGA から GPU へのデータ転送における通信レイテンシは，従来手法を用いた場合では 20 μsec，提案手法を用いた場合では 0.24 μsec となり，83.3 倍の性能差が確認された．これらの性能差は，従来手法では，CPU-FPGA 間通信と CPU-GPU 間通信をストアアンドフォワードで実行しなければならないのに対し，提案手法では GPU 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で FPGA からの DMA 転送を可能にしていることに起因している．これらの結果から，低レイテンシな通信が提案手法によって実現されることが明らかとなった．

4.3 通信バンド幅

図 13 に，従来手法と提案手法による GPU-FPGA 間データ転送の通信バンド幅を示す．図 13 の横軸は，通信バンド幅の測定におけるデータサイズを表しており，範囲は 4 ～ 1M - 4Byte である．従来手法を用いた場合では，GPU から FPGA へのデータ転送と FPGA から GPU のデータ転送における通信バンド幅はどちらも最大 3.1GB/s であった．前節で述べたように，CPU-FPGA 間通信と CPU-GPU 間通信はストアアンドフォワードで実行されるため，従来手法による GPU-FPGA 間データ転送の理論ピークバンド幅は以下の式で計算出来る．

$$\frac{N}{\frac{N}{8\,\mathrm{GB/s}} + \frac{N}{16\,\mathrm{GB/s}}} = 5.33\,\mathrm{GB/s} \tag{1}$$

ここで，N は通信データサイズ，8 GB/s は CPU-FPGA 間の理論ピークバンド幅 (PCIe Gen3 x8)，16 GB/s は CPU-GPU 間の理論ピークバンド幅 (PCIe Gen3 x16) を示す．したがって，従来方式による GPU-FPGA 間データ転送は 58%の実効性能であることが分かる．

一方，提案手法を用いた場合では，GPU から FPGA へのデータ転送における通信バンド幅は最大 4.1 GB/s，FPGA から GPU へは最大 6.9 GB/s であった．提案手法による GPU-FPGA 間データ転送では，FPGA デバイスの PCIe 接続が通信経路上において最も狭帯域であることから，理論ピークバンド幅は 8 GB/s となる．したがって，前者は 51%，後者は 86%の実効性能であることが分かる．前者の実効性能が低い理由としては，GPU から FPGA への DMA 転送を FPGA から起動すると，まず FPGA から GPU 内の DMA コントローラにメモリリクエストを送信し，GPU 内の DMA コントローラはリクエストを受信後に，データを FPGA に送信するため，実質 2 回分の通信が発生していることに起因していると考えられる．

図 13 より，我々の提案手法は GPU から FPGA，FPGA から GPU のどちらのデータ転送であっても従来手法より優れていることが明らかとなった．データ長が長い場合 (256KB 以上) では，GPU から FPGA への通信では約 1.3 倍，FPGA から GPU への通信では約 2.4 倍の性能差が確

© 2018 Information Processing Society of Japan

表 3: Hardware resource usage

	ALMs	Registers	M20K memory blocks
従来手法	27,801 (6.5 %)	52,080 (3.0 %)	175 (6.5 %)
提案手法	29,510 (6.9 %)	56,592 (3.3 %)	183 (6.7 %)
差分	1,709 (+0.4 %)	4,512 (+0.3 %)	8 (+0.2 %)

認された．特に，我々の提案手法は前述したようにレイテンシが低いため，バンド幅の立ち上がりが優れている．つまり，データサイズが小さくなる細粒度並列処理を行う状況ほど，我々の提案手法は価値を発揮することを意味している．

4.4 ハードウェアリソース使用量

表 3 に，従来手法と提案手法のハードウェアリソース使用量の差分を示す．ALM とは論理的に分割できるルックアップテーブルと複数のフリップフロップ (Register) を含む論理要素であり，ALM の使用率は FPGA に実装されるハードウェアコンポーネントのエリア面積を見積もる指標の 1 つである．The M20K memory block は FPGA チップ内に実装されているメモリブロック (ハードマクロ) であり，基本的に FIFO のような内部バッファの実装に用いられる．表 3 に示すとおり，提案手法はハードウェアリソースをほとんど利用せずに実現可能であることが分かる．これは，提案手法の実現に要するハードウェアコンポーネントが数個の FIFO のみであることが要因として考えられる．

5. まとめ

本稿では，GPU と FPGA を搭載した計算ノードにおけるデバイス間連携を効率的に行うための GPU-FPGA 間 DMA データ転送について提案し，それを OpenCL カーネルコードから制御する手法について述べた．従来手法と異なり提案手法は FPGA が主体的に DMA 転送を起動することが可能である．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，デバイス間 DMA 転送を実現した．

提案手法による GPU-FPGA 間データ転送における通信レイテンシと通信バンド幅を従来手法と比較評価した結果，通信レイテンシの面では GPU から FPGA の通信と FPGA から GPU の通信の両方で提案手法が優れており，GPU から FPGA の通信では 15.7 倍の性能差，FPGA から GPU の通信では 83.3 の性能差が確認された．通信バンド幅の面でも，提案手法は常に従来手法より性能が高く，データ長が長い場合（256KB 以上）では，GPU から FPGA への通信では約 1.3 倍，FPGA から GPU への通信では約 2.4 倍の性能差が確認された．特に，データサイズが小さくなる細粒度並列処理を行う状況ほど，我々の提案手法は価値を発揮することが評価結果より明らかとなった．これは，低レイテンシ通信によってバンド幅の立ち上がりが優れていることに起因する．

今後の研究においては，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」及び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」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] 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，阿部昂之，藤田典久，山口佳樹，朴 泰祐: GPU-FPGA 複合システムにおけるデバイス間連携機構，情報処理学会研究報告, 2018-HPC-165 (2018).

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

[6] 藤田典久，大畠佑真，小林諒平，山口佳樹，朴 泰祐: OpenCL と Verilog HDL の混合記述による FPGA プログラミング，情報処理学会研究報告, 2017-HPC-158 (2017).

[7] 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).

[8] Kobayashi, R., Oobata, Y., Fujita, N., Yamaguchi, Y.

© 2018 Information Processing Society of Japan

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).

© 2018 Information Processing Society of Japan