一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
VLD2018-89,CPSY2018-99,RECONF2018-63(2019-01)

異デバイス間でのPCIe通信を実現するOpenCL対応FPGAモジュールの提案と検証

小林　諒平[†,††]　藤田　典久[†]　山口　佳樹[††,†]　朴　泰祐[†,††]

† 筑波大学 計算科学研究センター　〒305–8577 茨城県つくば市天王台 1–1–1
†† 筑波大学 システム情報工学研究科　〒305–8573 茨城県つくば市天王台 1–1–1
E-mail: †{kobayashi,yoshiki,taisuke}@cs.tsukuba.ac.jp, ††fujita@hpcs.cs.tsukuba.ac.jp

あらまし　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 33.3 倍の性能差，通信バンド幅の面では最大 2.0 倍の性能差が確認された．

キーワード　GPU, FPGA, PCIe, OpenCL, BSP, DMA, ディスクリプタ, I/O Channel

Ryohei KOBAYASHI[†,††], Norihisa FUJITA[†], Yoshiki YAMAGUCHI[††,†], and Taisuke BOKU[†,††]

† Center for Computational Sciences, University of Tsukuba
1–1–1, Tennodai, Tsukuba-city, Ibaraki, 305–8577 Japan
†† Graduate School of Systems and Information Engineering, University of Tsukuba
1–1–1, Tennodai, Tsukuba-city, Ibaraki, 305–8573 Japan
E-mail: †{kobayashi,yoshiki,taisuke}@cs.tsukuba.ac.jp, ††fujita@hpcs.cs.tsukuba.ac.jp

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いるため，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．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

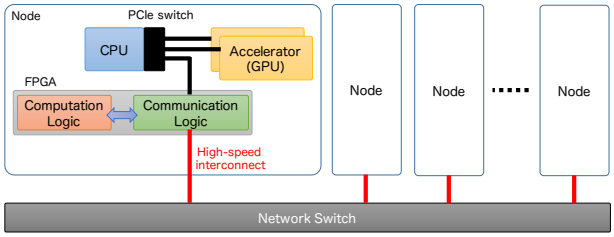


図 1　AiS コンセプトの概要．GPU では粗粒度並列処理を担当する計算カーネルが実行され，FPGA では GPU が不得手とする演算や集団通信を含む高速ノード間通信を担当するカーネルが実行される．CPU はこれらのカーネルの起動および全計算デバイスの調停を行う．

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となる．そこで本稿では，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 (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，通信レイテンシの面では最大 33.3 倍の性能差，通信バンド幅の面では最大 2.0 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，我々の提案手法の土台となるツールである Intel FPGA SDK for OpenCL について述べる．第 3 章で OpenCL カーネルコードから制御可能な GPU-FPGA 間データ転送について述べ，第 4 章で我々の提案手法を CPU を介する従来手法と比較し評価する．そして第 5 章で本稿をまとめる．

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 [4] 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

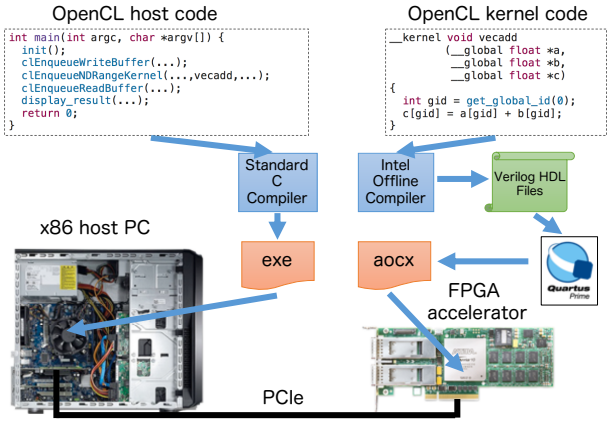


図 2　Intel FPGA SDK for OpenCL のプログラミングモデル．

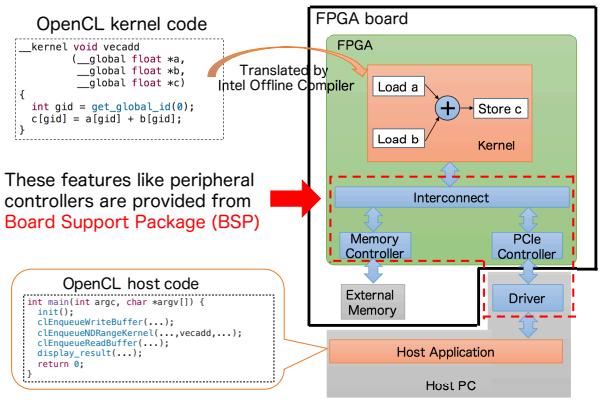


図 3　Intel FPGA SDK for OpenCL プラットフォームの構成図．

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 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

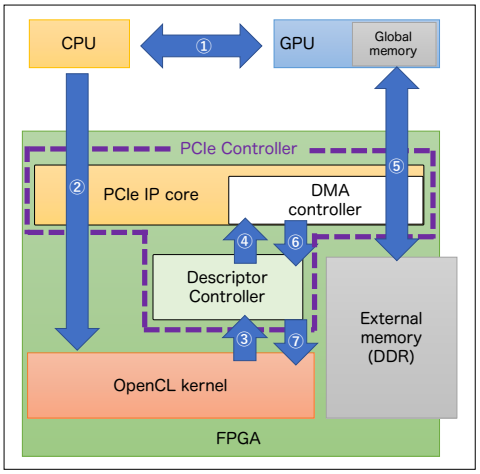


図 4　提案手法の概略図.

```
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．

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，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 に組み込まなければならない．我々はこれまでに，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 カーネルから制御する研究を行っている [5]．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．すなわち本稿では，提案する GPU-FPGA 間データ転送を実行する機能を BSP 中の PCIe コントローラに追加し，その機能を I/O Channel API を介して OpenCL カーネルから制御する手法について述べる．

3. OpenCL による GPU-FPGA 間データ転送

図 4 に，本提案手法の概要を示す．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

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

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，我々の提案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 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．我々の提案手法は，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

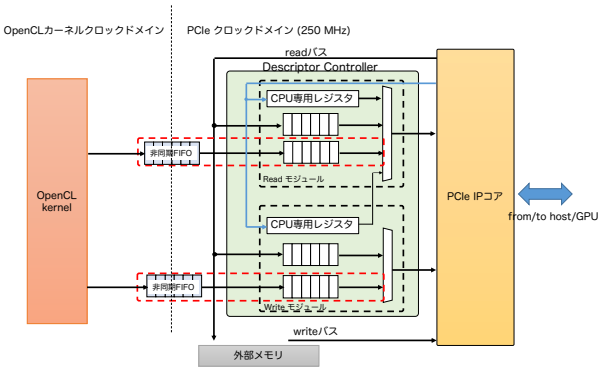


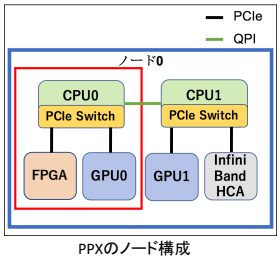
図 6　ディスクリプタコントローラの構成図．赤色の破線で囲まれたコンポーネントを加えることにより OpenCL カーネルからディスクリプタコントローラを操作し，ディスクリプタを DMAC に書き込むことができる．

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 (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．

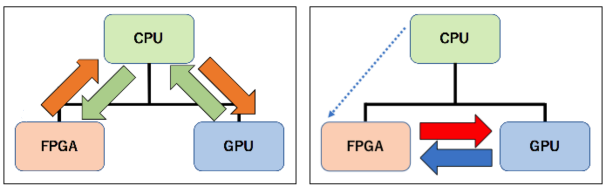
これらの動作を妨げることなく OpenCL カーネルコードから GPU-FPGA 間 DMA データ転送を実行するためには，Read モジュール，Write モジュール内に GPU-FPGA 間 DMA データ転送を実行するためのディスクリプタを格納する FIFO を用意し，プライオリティエンコーダによってそれぞれのモジュールからのディスクリプタの発行を適切に排他制御すれば良い．それらを実行するために図の赤色の破線で囲まれたコンポーネントを Verilog HDL で実装し，ディスクリプタコントローラに付け加えた．なお，OpenCL カーネルとディスクリプタコントローラのクロックドメインは異なるため，OpenCL カーネル

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



PPXのノード構成

図 7　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．



(a) 従来手法での計測区間　(b) 本研究による手法での計測区間

図 8　通信時間の計測区間．

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 [5] 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．

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 7 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．図 7 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 8 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-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 C++11 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 DMAC に書き込むことによって，FPGA が自律的に GPU-FPGA 間のメモリコピーを実行する．これら一連の処理 (図 4 の③から⑦までの処理) に要するサイクル数を取得する OpenCL ヘルパー関数 (カーネルコードで呼び出される) を実装し，それによって取得したサイクル数と FPGA の動作周波数から GPU-FPGA 間データ転送に要する時間を計測した．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めない．前述したように，この処理は一般的には計算開始時に一度実行すれば良く，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，破線部分の時間コストは無視できる．

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
提案手法	1.44 μsec
FPGA → GPU の通信	
従来手法	20 μsec
提案手法	0.6 μsec

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 DMAC 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 1.44 μsec となり，11.8 倍の性能向上が確認された．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 0.6 μsec となり，33.3 倍の性能向上が確認された．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，NVIDIA Kernel API を用いて 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 9 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 9 の横軸は，通信バンド幅の

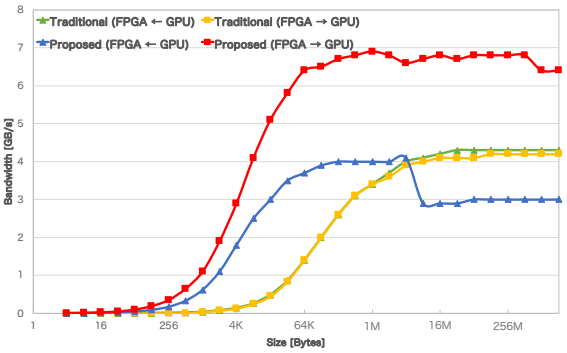


図 9　GPU-FPGA 間通信バンド幅の比較.

測定におけるデータサイズを対数軸で表しており，範囲は 4 ～ 2 GiB 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それぞれ最大 4.3 GB/s，4.2 GB/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．

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

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 間データ転送はそれぞれ 80.7 %，78.8 % 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 であり，それぞれ 1.03 倍，2.0 倍の性能差が確認された．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 51.3 %，後者は 86.3 % 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 DMAC にメモリリクエストを送信し，GPU 内の DMAC 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．また，8 MiB 以上のデータを GPU から FPGA に転送する場合，通信バンド幅が低下してしまう原因については未だ判明しておらず現在調査中である．

図 9 より，我々の提案手法は転送するデータサイズが 4 MiB 以下であれば，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．前述したように，我々の提案手法は通信レイテンシが小さいため，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．また，データ長が長い場合 (8 MiB 以上) であっても，FPGA から GPU への通信は従来手法よりも優れており，データサイズが 2GiB の場合では，従来手法の 1.5 倍の通信バンド幅が確認された．

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．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 DMAC 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 11.8 倍の性能差，FPGA から GPU の通信では 33.3 の性能差が確認された．通信バンド幅の面では，提案手法は，データサイズが 4 MiB 以下である場合は常に従来手法より性能が高く，データ長が長い場合 (8 MiB 以上) であっても，FPGA から GPU への通信では 2GiB のデータ転送において 1.5 倍の性能差が確認された．そして，提案手法は低レイテンシ通信によって通信バンド幅の立ち上がりが優れているため，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価値を発揮することが明らかとなった．

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．

文　献

[1] 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.

[2] T. Kuhara, C. Tsuruta, T. Hanawa, and H. Amano. Reduction calculator in an fpga based switching hub for high performance clusters. In *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–4, Sept 2015.

[3] Chiharu Tsuruta, Yohei Miki, Takuya Kuhara, Hideharu Amano, and Masayuki Umemura. 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, April 2016.

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

[5] Ryohei Kobayashi, Yuma Oobata, Norihisa Fujita, Yoshiki Yamaguchi, and Taisuke Boku. Opencl-ready high speed fpga network for reconfigurable high performance computing. In *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, HPC Asia 2018, pp. 192–201, New York, NY, USA, 2018. ACM.