

GPU-FPGA 複合システムにおけるデバイス間連携機構

小林 諒平^{1,2} 阿部 昂之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トを 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 デバイスの内蔵メモリ間で 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 8.4 倍の性能差、通信バンド幅の面では最大で 3.7 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氣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 デバイスのグローバルメモリと FPGA デバイスの内蔵メモリ間で CPU を介さ

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

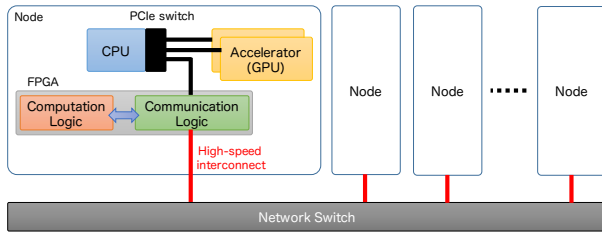


図 1: AiS コンセプトの概要. GPU では粗粒度並列処理を担当する計算カーネルが実行され, FPGA では GPU が不得手とする演算や集団通信を含む高速ノード間通信を担当するカーネルが実行される. CPU はこれらのカーネルの起動および全計算デバイスの調停を行う.

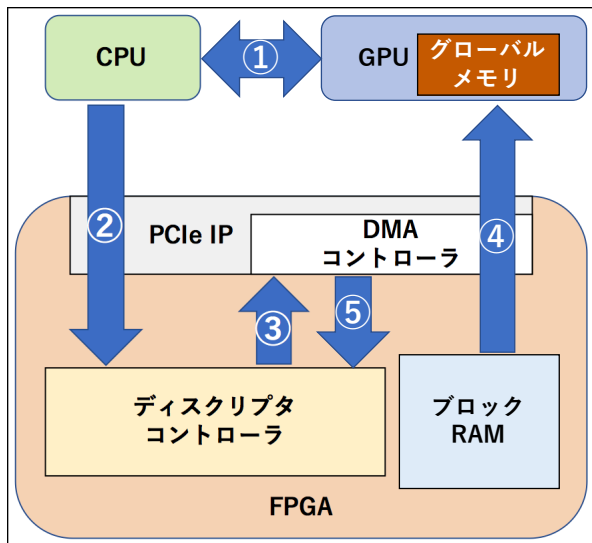


図 2: 開発した GPU-FPGA 間通信機能の概要. PCIe DMA 転送用の IP コアを用いて, デバイス間データ転送を行う.

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, その通信性能を評価した. 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で 8.4 倍の性能差, 通信バンド幅の面では最大で 3.7 倍の性能差が確認された. 本稿では, ユーザレベルで CPU-FPGA 間 DMA 転送を実現するためのアプローチについて述べる.

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, 通信レイテンシと通信バンド幅の評価を第 3 章にて述べる. そして, 第 4 章で, GPU-FPGA 間通信を可能な限りユーザから隠蔽しつつアプリケーションからの利用を容易にする機能について議論し, 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

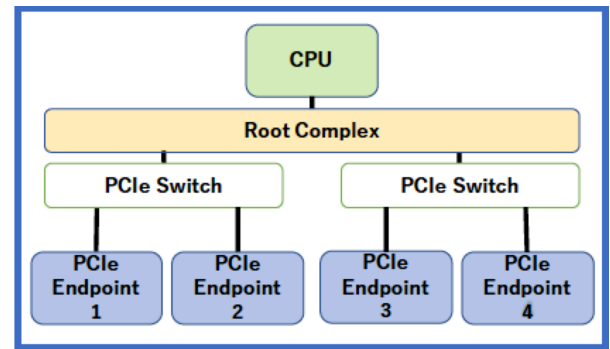


図 3: CPU と PCIe デバイスの接続.

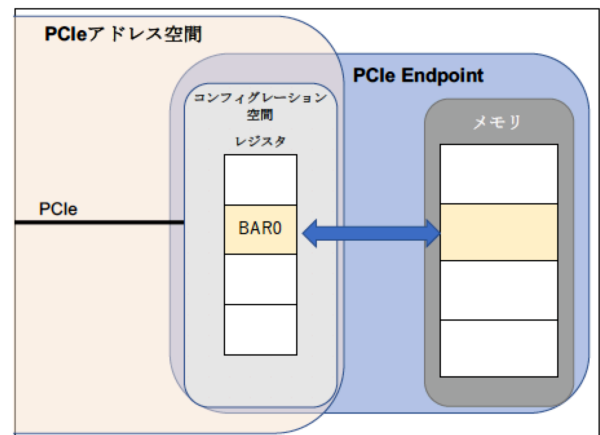


図 4: PCIe アドレス空間

バイスの内蔵メモリを 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, **FPGA が主体的に DMA 転送を起動する**点が既存技術と異なる. FPGA から GPU に対しての DMA 転送は以下の手順で実行され, これらの詳細を次節より述べる.

- 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) ディスクリプタをホスト CPU から FPGA に送信
- FPGA 側での設定
 - (3) ホストから受け取ったディスクリプタを 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

```

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     for (int i = 0; i < SIZE/4; i++) data[i] = i;
15
16     cudaMemcpy(ptr, data, SIZE,
   cudaMemcpyHostToDevice);
17     printf("paddr=%0x%016llx\n", paddr);
18     sleep(50);
19     cudaMemcpy(data, ptr, SIZE,
   cudaMemcpyDeviceToHost);
20
21     return 0;
22 }

```

図 5: PCIe アドレス空間への GPU メモリのマップ

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。

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

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

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、変更されているマクロ等が存在した。

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、本稿ではその機能を流用する。

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。ディスクリプタは特定の形式に従ってデータが格納されている。ディスクリプタの形式を表 3.1 に示す。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 (バイト)、DMA Descriptor ID は転送完了時にどの転送が完

```

1  const off_t ONCHIP_MEMORY_OFFSET = 0x00000000;
2  const off_t RD_AST_RX_OFFSET      = 0x40000000;
3
4  inline void wr(void* p, unsigned i, uint32_t v) {
5      reinterpret_cast<uint32_t volatile*>(p)[i] = v;
6  }
7
8  void rd_ast_rx(void* ctrl, unsigned long src,
9      unsigned long dest, unsigned len, unsigned id)
10     {
11         uint32_t x0 = src & 0xfffffffful;
12         uint32_t x1 = (src >> 32) & 0xfffffffful;
13         uint32_t x2 = dest & 0xfffffffful;
14         uint32_t x3 = (dest >> 32) & 0xfffffffful;
15         uint32_t x4 = ((id & 0xff) << 18) | ((len >> 2)
16             & 0x3ffff);
17
18         wr(ctrl, 0, x0);
19         wr(ctrl, 0, x1);
20         wr(ctrl, 0, x2);
21         wr(ctrl, 0, x3);
22         wr(ctrl, 0, x4);
23     }
24
25 int random_id() {
26     std::random_device seed;
27     std::mt19937 g(seed());
28     std::uniform_int_distribution<int> d(0, 0xff);
29     return d(g);
30 }
31
32 int main(int argc, char** argv) {
33     mapbar_device dev;
34     void* ctrl = dev.mmap(2, RD_AST_RX_OFFSET);
35     unsigned long const addr = dev.get_buffer();
36     int id = random_id();
37     rd_ast_rx(ctrl, addr, ONCHIP_MEMORY_OFFSET, size
38         *4, id[i]);
39 }

```

図 6: ホストによる FPGA へのディスクリプタの書き込み

了したかを判別するために用いる ID である。ディスクリプタ内の Source や Destination の Address を変更することにより、DMA 通信の相手を CPU や GPU に変更可能である。

CPU から FPGA のメモリに対してディスクリプタを書き込むために、FPGA のメモリ空間をホストメモリにマップするカーネルモジュールを作成した。このモジュールは、FPGA の PCIe アドレス空間を mmap システムコールによってユーザ空間にマップできる機能を持ち、これを用いることで、CPU 上のユーザ空間で実行されているプログラムから FPGA のメモリ空間にアクセスすることが可能となる。Mmap された FPGA メモリ領域に対してデータを書き込むと、FPGA の PCIe DMA コントローラにディスクリプタが書き込まれる仕組みとなっている。

CPU から FPGA に対してディスクリプタを書き込み、

```

1  always @(posedge clk) begin
2      if (write == 0) begin
3          count <= count + 32'b1;
4      end
5      if (!rstn) begin
6          data <= 160'b0;
7      end else begin
8          if (avm_write & ~avm_waitrequest) begin
9              case (s)
10                 3'd1: begin
11                     data[ 31: 0] <= avm_writedata;
12                     count <= 0;
13                     state <= 3'd1;
14                 end
15                 3'd2: begin
16                     data[ 63: 32] <= avm_writedata;
17                 end
18                 3'd3: begin
19                     data[ 95: 64] <= avm_writedata;
20                 end
21                 3'd4: begin
22                     data[127: 96] <= avm_writedata;
23                 end
24                 3'd5: begin
25                     data[159:128] <= avm_writedata;
26                     id <= avm_writedata[25:18];
27                 end
28             endcase
29         end
30         case (state)
31             3'd1: begin
32                 if (avalon_streaming_sink_valid == 1)
33                     begin
34                         if (avalon_streaming_sink_data == {2'h01, id} &
35                             avalon_streaming_source_1_ready)
36                             begin
37                                 write <= 1;
38                                 state <= state + 3'd1;
39                             end
40                         end
41                     end
42                 3'd2: begin
43                     write <= 0;
44                     count <= 32'd0;
45                     state <= 3'd0;
46                 end
47             endcase
48         end
49     end

```

図 7: Verilog による自作モジュールのディスクリプタ受け取り及び時間計測部分

READ を実行させるためのコードを図 6 に示す。コード中の mapbar_device dev は、キャラクタデバイスとして登録された FPGA を open している。rd_ast_rx() 関数によって、ディスクリプタの形式に則り、FPGA 側に 32 ビットずつに分けて自作モジュールにデータを書き込む。ディス

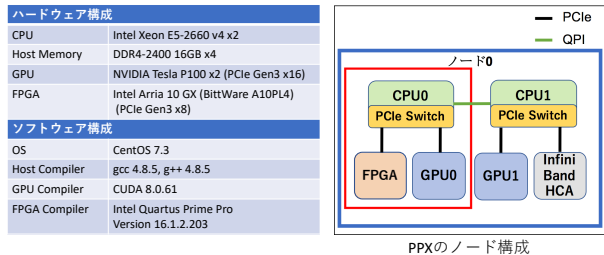


図 8: 評価環境および PPX システムの計算ノードの構成図。評価には赤色の枠線で囲まれたデバイスを用いた。

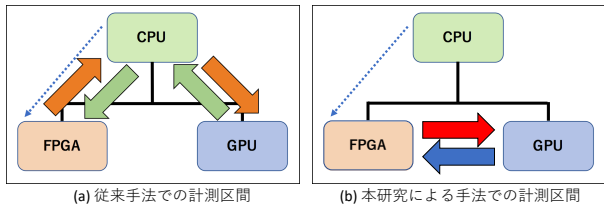


図 9: 通信時間の計測区間。

クリプタのデータは PIO (Programable IO) を用いて CPU から FPGA の BAR 2 空間に書き込まれるが、FPGA の PCIe コントローラの仕様により 32bit 単位でしか PIO アクセスできないため、ディスクリプタを 32bit 毎に分解して書き込む必要がある。

FPGA 内の自作モジュールにおいて、ディスクリプタの受け取り及び時間計測を行っている部分を図 7 に示す。まず、CPU から 32 ビット単位でバラバラに書き込まれるディスクリプタのデータを再構築し、レジスタ data に格納する。そして、そのディスクリプタを Avalon-ST というプロトコルを用いて、DMA コントローラに対してディスクリプタを書き込む。Avalon-ST は Intel FPGA で標準的に用いられるストリーミング用インターフェイスであり、1:1 の接続に用いられるものである。時間の計測は 1 クロック毎に count に 1 を加算することにより行う。このクロックは PCIe IP から供給されており、周波数は設定によって変動するが、本稿で用いる PCIe Gen.3 x8 レーン設定時では 250 MHz で駆動され、4 ns の時間解像度を持つ。これを、Avalon-ST を用いて FPGA 内のメモリマップされたブロック RAM (BRAM) に書き込む。この BRAM は FPGA の PCIe メモリ空間に接続されており、CPU 側からその値を認識することができるようになっている。

3. 評価

3.1 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・

表 2: GPU-FPGA 間通信レイテンシの比較。

FPGA ← GPU の通信	
従来手法	3.92 μ sec
本研究による手法	1.45 μ sec
FPGA → GPU の通信	
従来手法	3.63 μ sec
本研究による手法	0.43 μ sec

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 8 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 8 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 9 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。評価では、1 と 2 の通信に要した時間をそれぞれ計測し、合計している。すなわち、1 では、FPGA 内の PCIe DMA コントローラに対してホストから送信された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、2 では、cudaMemcpy に要した時間をそれぞれ計測し、れらを合計した。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、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 に対してのディスクリプタの送信を表しており、評価ではディスクリプタ書き込み時間は含まないことに留意して頂きたい。

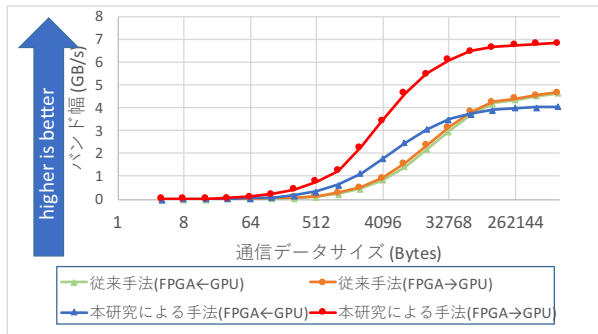


図 10: GPU-FPGA 間通信バンド幅の比較.

3.2 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 3.92 μ sec, 提案手法を用いた場合では 1.45 μ sec となり、2.7 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 3.63 μ sec, 提案手法を用いた場合では 0.43 μ sec となり、8.4 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。

3.3 通信バンド幅

図 10 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 10 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 ~ 1MByte である。従来手法を用いた場合では、GPU から FPGA へのデータ転送における通信バンド幅は最大 4.64 GB/s, FPGA から GPU へは最大 4.65 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。

$$\frac{N}{\frac{N}{8 \text{ GB/s}} + \frac{N}{16 \text{ GB/s}}} = 5.33 \text{ GB/s} \quad (1)$$

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 87% 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 4.48 GB/s, FPGA から GPU へは最大 6.83 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 56%, 後者は 85% の実効性能であることが分かる。

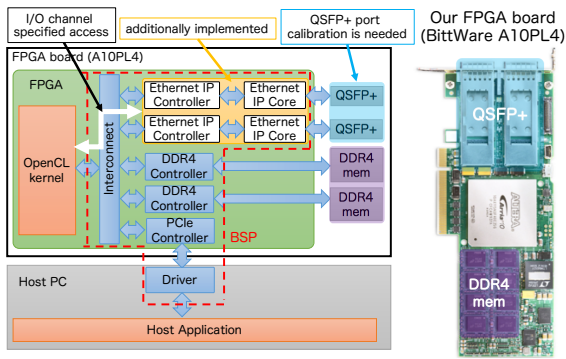
FPGA から GPU への通信では常に従来手法より性能が高く、最大で 3.7 倍の性能差、またデータ長が長い場合 (256KB 以上) で約 1.4 倍の性能差が確認された。一方、GPU から FPGA への通信では 64KB データサイズまでは本研究の手法が勝っているが、これを超えると従来手法のほうが性能が良い結果となった。この原因は現在解析中であるが、特にデータサイズが小さくなる細粒度並列処理を行う場合、提案手法のほうが有利であることは確認された。また、提案手法を用いる場合、従来手法と比べ、バンド幅の立ち上がりが優れている。これは、前節で示した提案手法によって実現される低レイテンシな通信によってもたらされている。

4. Next Step: アプリケーションプログラムからの GPU-FPGA 間データ転送

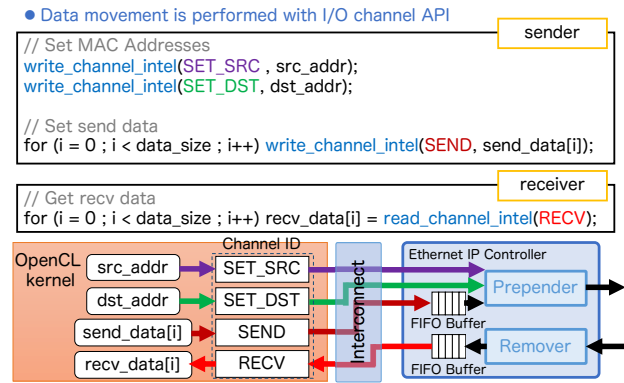
本稿では、GPU プログラミングと FPGA プログラミングの連携を効率的に行うためのデバイス間データ転送について提案したが、それを可能な限りユーザから隠蔽しつつアプリケーションからの利用を容易にする機能が実際の運用には求められる。我々は、OpenCL を代表とする高位言語による設計手法をサポートする FPGA 開発環境を活用して、これを実現することを検討している。

Intel FPGA SDK for OpenCL では、ユーザーが自作したハードウェアモジュールを OpenCL 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 を提供しており、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 [4], [5], [6]. 例えば、[6] 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 11 に示す。

Intel FPGA SDK for OpenCL を利用するためには、BSP (Board Support Package) 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 11 (a) に示す内蔵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



(a) ハードウェア実装の概略図



(b) Ping-pong通信のOpenCLコードの一部

図 11: QSPF+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

通信ポートの使用を望む場合、QSPF+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 11 (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。

Intel FPGA SDK for OpenCL に関するこれらの仕組みを利用し、今後の研究では、提案したデバイス間データ転送を実行する機能を BSP に組み込み、本稿で述べた FPGA 主体の GPU-FPGA 間 DMA 転送が I/O Channel API を介して OpenCL から制御可能であることを明らかにする。

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 データ転送手法について提案した。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、最大で 8.4 倍の性能差が確認された。通信バンド幅の面で

は、FPGA から GPU への通信では常に従来手法より性能が高く、最大で 3.7 倍の性能差、またデータ長が長い場合 (256KB 以上) で約 1.4 倍の性能差が確認された。一方、GPU から FPGA への通信では 64KB データサイズまでは本研究の手法が勝っているが、これを超えると従来手法のほうが性能が良い結果となった。この原因は現在解析中であるが、特にデータサイズが小さくなる細粒度並列処理を行う場合、提案手法のほうが有利であることは確認された。また、提案手法を用いる場合、従来手法と比べ、バンド幅の立ち上がりが優れている。これは、提案手法によって実現される低レイテンシな通信によってもたらされている。

今後の研究においては、本稿で提案したデバイス間データ転送をユーザーレベルで制御するための機能を、OpenCL を代表とする高位言語による設計手法をサポートする FPGA 開発環境を活用しながら開発していく。そして、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] 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).
 - [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).