Vol.2019-HPC-168 No.10
2019/3/6

GPU・FPGA混載ノードにおける ヘテロ演算加速プログラム環境に関する研究

中道安祐未[1]　小林諒平[2,1]　藤田典久[2]　朴泰祐[2,1]

概要：近年，高性能コンピューティング (HPC : High Performance Computing) の分野において，アクセラレータを搭載した大規模計算クラスタが主流の 1 つとなっている．アクセラレータには，主に Graphics Processing Unit (GPU) が用いられているが，HPC 分野では処理の柔軟性や電力効率の高さから Field Programmable Gate Array (FPGA) が注目されつつある．そこで，GPU が不得意な計算を FPGA に行わせる GPU+FPGA の複合システムにより実アプリケーションのさらなる高性能化を目指す．本研究では，GPU と FPGA の両方を搭載した計算機で GPU+FPGA のハイブリッドアクセラレーションを実現するプログラムの開発手法と環境について議論する．

1. はじめに

近年, HPC 分野では GPU などのアクセラレータを搭載した大規模計算クラスタが主流の 1 つとなっている．世界のスーパーコンピュータの性能ランキングである TOP500[1] を参照すると，上位 10 位のうち 6 つのシステムがアクセラレータを搭載している．

GPU は CPU よりも高い電力効率と演算性能を有しているが，条件分岐により性能が大幅に低下する，データレベルの並列性が低いと GPU の性能を発揮できない，並列化をしたときノードをまたぐ GPU 間の通信コストが高いなどのデメリットを抱えている．

その一方で, 近年, FPGA の HPC 分野へのアクセラレータとしての応用が注目されている．FPGA は，内部の論理回路の構造を何度も繰り返し再構成可能なハードウェアで回路規模が数百万ゲートの大規模なものも存在している．特に近年の高性能 FPGA 間は CPU を介さずにデバイスからのダイレクトな高速光通信を可能とするインタフェースを備えたものもあり，FPGA が主体的に高速通信を行うことができる．しかし，アプリケーションの実行時には，動的再構成を取り入れなければ計算リソースが限られてしまう．これらの問題を解決し，FPGA と GPU を相補的に活用するため，我々は図 1 のように GPU と FPGA が PCI express (PCIe) で接続された GPU+FPGA 複合システムに関する研究を進めている．

しかし，GPU と FPGA はデバイスにアクセスするための API や開発言語が異なる．現在最も多用されている NVIDIA 製 GPU では CUDA を用いたプログラミング環境が用意されているが，FPGA では Verilog HDL などのハードウェア記述言語による実装に加え，OpenCL や C 言語などの高級言語を用いた高位合成手法が提供されている．本研究では一般的なアプリケーション開発者が活用することを前提として，OpenCL による FPGA プログラミングを前提とする．GPU+FPGA 複合システムの活用に当たり，解決すべき問題は以下の通りである．

- GPU と FPGA のコンパイル環境の組み合わせ
- 開発言語 (CUDA と OpenCL) の協調的利用
- GPU と FPGA の連携動作

本研究では GPU と FPGA を同一プログラムから呼び出し，協調計算するための基本モデルを提案，議論する．これまでに, GPU と FPGA の連携計算での計算結果検証が完了し，本研究で提案する協調計算モデルにより GPU+FPGA 複合システムが活用可能である．本報告では，これらについて述べる．

本稿の構成を以下に示す．第 2 章では関連研究，第 3 章ではアクセラレータプログラミング，第 4 章では第 3 章のアクセラレータプログラミングをふまえて GPU・FPGA の協調計算のプログラミングの方法について述べる．第 5 章では第 4 章のプログラム環境の整合性を検証し，最後に，第 6 章にて本研究をまとめる．

2. 関連研究

筑波大学計算科学研究センターでは，CPU を介さない GPU 間直接通信機構として TCA (Tightly Coupled Accel-

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

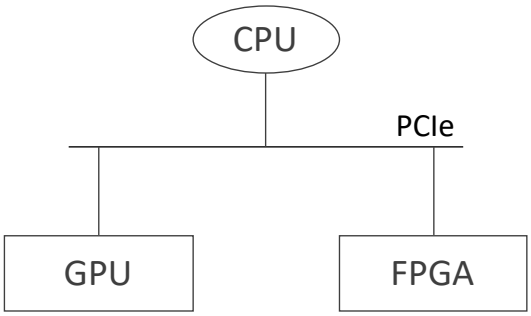

図 1 ハードウェアの構成

erators) アーキテクチャを提唱し，その実装として PEACH2 (PCI Express Adaptive Communication Hub version 2) を FPGA により開発した [2]．PEACH2 を経由して通信を行うことで，GPU 同士の通信を高速に行えるが，GPU 間の通信途中に CPU による演算が必要になると，PEACH2 の性能を生かすことができない．本来通信機構としてしか用いられていない PEACH2 に追加要素として演算機構を取り付け，ノード間通信を行う際に CPU で行う処理をオフロードする．[3] では，実際に PEACH2 を用いて N 体シミュレーションの高速に取り組んでいる．結果，GPU 間通信の前処理となるツリー法の枝刈り部分を PEACH2 で処理することにより，CPU の約 7.2 倍の高速化を達成した．

FPGA パフォーマンスの向上に加えて，OpenCL ベースの FPGA 開発環境が FPGA ベンダーによって提供されているため，従来に比べてプログラミングコストが低くなっている．このような改善によって，低レイテンシのデータ移動を実行しながら，CPU / GPU の FPGA へのパフォーマンスが低いオンザフライのオフロード計算を可能にするという概念を実現した．この概念を実現するために OpenCL と Verilog HDL の混合プログラミングを用いた高性能の GPU-FPGA データ通信を提案し，両者を円滑に連携させた [4]．その結果，toy program を用いた実験より，提案手法が GPU と FPGA 間で 0.6 [μ s] のレイテンシ，最大 6.9 [GB/s] を達成し，提案手法が高性能な GPU-FPGA 協調演算の実現に有効であることを示している．

GPU は HPC では最も一般的に使用されるアクセラレータであるが，場合によっては性能のボトルネックとなることもある．その一方で，回路設計が柔軟な FPGA を利用する機会が増えてきている．しかし，アプリケーション開発者がアプリケーションやアルゴリズム用に FPGA の論理回路を実装することは容易ではない．そのため，近年の FPGA の開発環境の進歩により，OpenCL 言語を用いた高位合成 (HLS) 開発環境が普及してきている．そこで，OpenCL を用いて FPGA 上での Authentic Radiation Transfer (ART) 法の最適化を行った [5]．その結果，OpenMP を使用した CPU 実装と比較して 6.9 倍高速な性能を達成した．複数の FPGA を使用し並列化された実装は，GPU よりも高い性能を達成すると考えらる．

OpenACC などのディレクティブベースのアクセラレータプログラミングモデルは，Scalable Heterogeneous Computing (SHC) プラットフォームをプログラミングするための代替ソリューションとして生まれたが，SHC システムの複雑さが増すと，移植性と生産性の点で課題点が生じる．そこで，OpenARC と呼ばれるオープンソースの OpenACC コンパイラを紹介する [6]．13 個の標準 OpenACC プログラムと 3 個の拡張 OpenACC プログラムを CUDA GPU に移植すると，OpenARC は商用の OpenACC コンパイラと同様に機能し，高レベルの研究フレームワークも提供する．また，OpenACC は様々なデバイスに対応しており，その 1 つとして FPGA が挙げられる [7]．

Axel クラスタは，各ノードに FPGA，GPU など複数の種類のアクセラレータを含んだシステムである [8]．ノード内・ノード間通信にはシステムバスが用いられている．Axel クラスタのための Map-Reduce フレームワークによって，異なるタイプの処理要素と通信チャネルを通して空間的および時間的局所性を利用した結果，N 体シミュレーションの性能が 4.4 倍から 22.7 倍向上した．

過去の研究において GPU と FPGA を演算と通信の両面で協調させる試みは報告されていない．PEACH2 に利用されている Stratix IV は浮動小数点演算を行う FP DSP(floating DSP) を搭載していないため，GPU と協調して演算を行うだけの演算性能を有していない．そのため，PEACH2 では FPGA を通信処理に限定して利用している．したがって，PEACH2 を利用して GPU と協調して演算を行うことはできない．本研究では，GPU と協調して演算を行う FPGA として FP DSP を搭載している Arria10 を利用し，GPU が不得意な処理を補完することでより高速な計算処理の実現を目指す．

3. アクセラレータプログラミング

3.1 GPU プログラミング

GPU プログラミングのための代表的な言語は CUDA[9] である．CUDA は NVIDIA 社が開発・提供している GPU のための言語である．CUDA には host プログラムと kernel プログラムの 2 種類のプログラムが存在する．host プログラムは，CPU 上で動作し kernel の実行を制御するプログラムである．kernel プログラムは，GPU 上で動作するプログラムである．

CUDA プログラミングの流れを図 2 に示す．図 2 のように書かれたプログラム test.cu を NVIDIA が提供する CUDA コンパイラ nvcc で `nvcc test.cu -o test` とコンパイルする．その結果，プログラムから CPU 向けに書かれた部分 (host コード) と GPU 向けに書かれた部分 (GPU kernel) が分離し，それぞれがコンパイルされて一つの実行ファイル test が生成される．

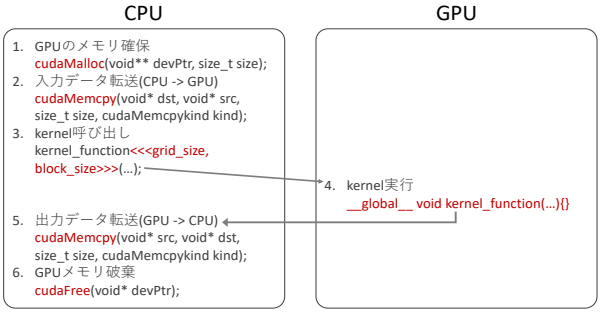

図 2　CUDA プログラミングの流れ

3.2　FPGA プログラミング

FPGA の一般的なプログラミングモデルは，Verilog HDL や VHDL のようなハードウェア記述言語 (HDL: Hardware Description Language) が挙げられる．一般的に FPGA のプログラミングは Verilog HDL や VHDL を用いて行われる．HDL は，論理回路の動作を定義するために設計された専用言語で，プログラミング言語に似た構文や表記法を用いて，回路に含まれる素子の構成やそれぞれの動作条件，素子間の配線などの記述が可能となっている．しかし，HDL を用いたプログラミングでは回路動作をクロック単位で定義する必要があるため，記述が複雑になり動作検証も困難となる．また，ハードウェアの素養を持たない開発者にとって HDL で目的の処理を記述することは難しい．したがって，HDL を用いた FPGA のプログラミングコストは高い．我々は，この問題を解決するために OpenCL[10] に注目している．

OpenCL は，ヘテロジニアスな並列計算機環境に適した並列プログラミングのための言語である．ヘテロジニアスとは，異なる種類のプロセッサを組み合わせて構築したシステムのことである．OpenCL はハードウェアに近いレベルで API を共通化しており，高度な抽象化が行われていないので，実際に利用する演算プロセッサの特徴に適したパフォーマンスチューニングが OpenCL 上で実施可能となっている．また，OpenCL では異なる種類のプロセッサのプログラミングを C 言語をベースとした言語である OpenCL C を用いて行うことができる．OpenCL C を利用することで，従来と比べて高水準な言語で FPGA のプログラミングを行うことができるため，実装コストを低減させることができる．OpenCL プログラミングの流れを図 3 に示す．OpenCL プログラミングでは，CUDA と同様に host プログラムと kernel プログラムの 2 種類のプログラムが存在する．host プログラムは，CPU 上で動作し kernel の実行を制御するプログラムである．kernel プログラムは，デバイス上で動作するプログラムである．しかし，OpenCL では，host プログラムと kernel プログラムは別々に記述し，別々にコンパイルする．host プログラム host.c は gcc や Intel Compiler などの C コンパイラでコンパイルし，kernel プログラム kernel.cl は専用コンパイラでコンパイルされ論理合成可能なファイルに変換される．

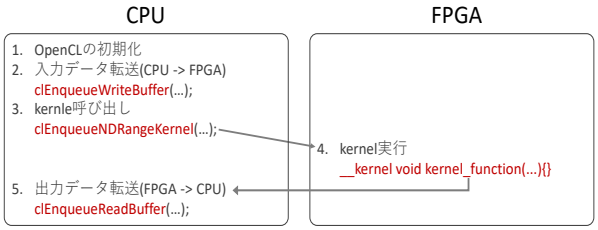

図 3　OpenCL プログラミングの流れ

近年では，高位合成と呼ばれる技術の発達により OpenCL を用いて FPGA のプログラミングを行うことができる開発環境が登場してきている．本研究では，OpenCL に対応している FPGA の開発環境として Intel 社が提供している Intel FPGA SDK for OpenCL[11] を利用する．Intel FPGA SDK for OpenCL には OpenCL C で記述されたコードを FPGA の回路に変換するコンパイラである aoc(Altera Offline Compiler) が付属している．ユーザは aoc を使って回路を生成した後にホストプログラムから生成した回路を FPGA に転送し，実行する．

4.　GPU・FPGA プログラミングの方法

OpenCL は，ヘテロジニアスな並列計算機環境に適した並列プログラミングのための言語であると，3.2 節で述べたが，CPU をはじめ，GPU，FPGA などさまざまなプロセッサ上で動かすことができる．今回の研究では，GPU プログラミングに CUDA，FPGA プログラミングに OpenCL を用いている．GPU プログラミングに OpenCL[10] ではなく，CUDA を使用する理由は，OpenCL よりも CUDA の方が実用的アプリケーションが多く開発されており，また本研究では NVIDIA の GPU を使用しているため，この GPU で使用することができる機能を全て有効利用するためには CUDA が適切と考えられるためである．

CPU，GPU，FPGA それぞれの挙動をそれぞれ，CUDA と OpenCL を用いて記述する．CUDA のコード内には 2 種類のプログラム上のブロックがある．host プログラムと kernel プログラムが 1 つのプログラムの中に記述されている．host プログラムは CPU で動き，kernel プログラムは GPU で動き host 側でキック (kernel プログラムの起動) される．代表的な GPU ベンダーである NVIDIA 社の GPU は CUDA に対応しており，コンパイルは nvcc で行う．

OpenCL にも CUDA 同様 2 種類のプログラムブロックがある．しかし，host プログラムと kernel プログラムは別々に記述する．host プログラムは CPU で動き，kernel プログラムは FPGA で動き host 側でキック (kernel プログラムの起動) される．コンパイルは host プログラムは gcc，kernel プログラムは aoc で行う．

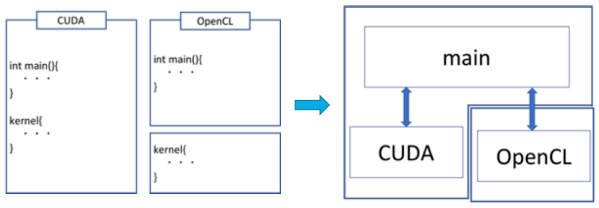

図 4　プログラムの構成

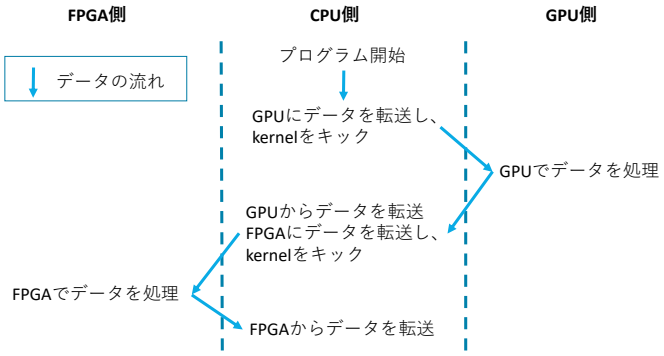

図 5　処理の流れ

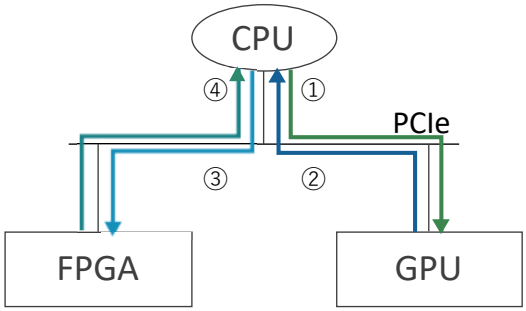

図 6　データの流れ

1 つの host プログラムで，GPU と FPGA を両方呼び出せるようにする．CUDA と OpenCL の host プログラムは別々であるため，main 関数は 1 つでなくてはならないのに，main 関数が 2 つある状態である．つまり，1 つの host プログラムから CUDA と OpenCL を呼び出す必要がある．

なお，GPU と FPGA の協調動作として，処理を分散し，非同期に同時並行的に両者を利用することも考えられるが，ここではまず全体のプログラミングの枠組みを検証するため，両デバイスは交互に利用し，それらの間でデータの移動を含めた単純なモデルを想定するものとする．

5. GPU+FPGA 複合プログラムの検証

本研究では，GPU+FPGA によるアプリケーション開発に先立ち，このようなプラットフォーム上で CUDA と OpenCL による複合プログラミングが可能であることを，toy program を例として検証する．

5.1 CUDA と OpenCL の協調

1 つのホストプログラムで，GPU と FPGA の両方を利用して協調計算を行う．GPU と FPGA は，それぞれ CUDA と OpenCL という異なる API でアクセスしなければならないため，1 つのホストプログラムで CUDA と OpenCL 両方の API を呼び出し協調実行する必要がある．このときのプログラム構成を図 4，ホストプログラム上で想定される処理の流れを図 5，データの流れを図 6 に示す．以上の想定から，どのように実装すれば，1 つのホストプログラムから CUDA と OpenCL の協調計算が可能なのかを議論する．

5.2 GPU+FPGA 複合プログラムの実現方法

前節の協調プログラムを実装するにあたり，一番重要になるのがコンパイル方法である．GPU と FPGA は異なる開発環境を提供するため，コンパイルの方法も異なる．そのため，どのように 2 つの開発環境を統合していくのかが論点となる．

GPU+FPGA 複合プログラムについて，コンパイルの流れを図 7 に示す．GPU アプリケーションは，CUDA が提供するコンパイラの “nvcc” でコンパイルするが，実体は GCC の C++コンパイラをバックエンドとして CUDA 固有の制御文やライブラリを付与するもので，あえて GCC で行う場合は `g++ main.cu -lcuda -lcudart` でコンパイルできる．FPGA アプリケーションは，`g++ -std=c++11 func.cc` ‘`aocl compile-config`‘ ‘`aocl link-config`‘でコンパイルする．Intel FPGA 用 OpenCL のコンパイルには “aoc” コンパイラが提供されており，高級言語で記述されたコードを FPGA の回路に変換する．‘aocl compile-config‘および ‘aocl link-config‘は，FPGA OpenCL のホストプログラムを作る際に必要な以下のコンパイルオプションを取得するコマンドである．

- compile-config：必要なコンパイルオプションを表示
- link-config：実行ランタイムライブラリのリンクオプションを表示

GPU+FPGA 複合プログラムでは，CUDA ライブラリおよび OpenCL ライブラリの両方をリンクしなければならない．したがって，CUDA API と OpenCL API にアクセスする関数をそれぞれ別のソースファイルとして作成し，それぞれを各コンパイラに渡してオブジェクトファイルを生成する．この関数をプロキシ関数と呼ぶ．エントリポイントではプロキシ関数を呼び出すだけにすれば，ホストプログラムのコンパイルは単純に GCC コンパイラで行うことができる．すべてのオブジェクトファイルをリンクして実行ファイルを生成する時に，CUDA と OpenCL のライブラリをリンクすれば，GPU+FPGA 複合プログラムを生成できる．

5.3 ベクトル内積計算による実行確認

ベクトルの内積計算を例として，GPU+FPGA 複合プロ

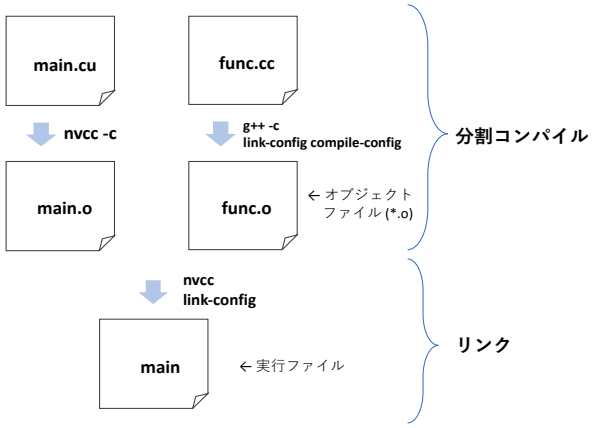

図 7　コンパイルの流れ

表 1　本研究の評価環境

CPU	Intel(R) Xeon(R) CPU E5-2660 v4 x2
GPU	NVIDIA P100 x2 (PCIe Gen3 x16)
FPGA	BittWare A10PL4 (PCIe Gen3 x8)
OS	CentOS 7.3
Host compiler	g++ 4.8.5
GPU compiler	CUDA 9.0.176
FPGA compiler	quartus 17.1.2.304 aocl 17.1.2

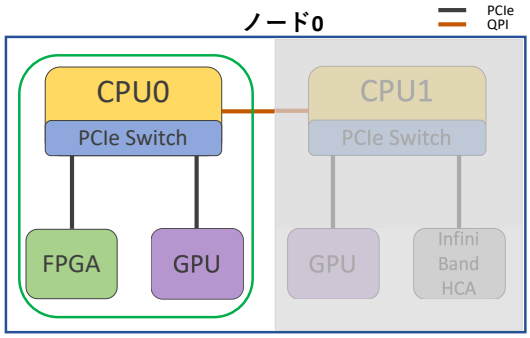

図 8　ノードの構成

グラムの検証を行う．

検証には，筑波大学計算科学研究センターの PPX (Pre-PACS version X) を用いた．AiS の技術実証器として PPX は開発されている．評価環境を表 1 に，ノード構成を図 8 に示す．本研究で用いたノード部分は緑の枠で囲まれている部分である．今回は GPU，FPGA の協調計算が可能かの検証であるため，グレーの部分は使用していない．

GPU と FPGA を複合した，ベクトルの内積計算のカーネル実装をソースコード 1，ソースコード 2 に示し，全体の計算の流れを図 6 に示す．GPU では内積の部分和計算を行い，FPGA では GPU で求めた内積の部分和の総和を行う．図 6 の説明を以下に示す．

(1) CPU で GPU と FPGA の初期化やメモリ確保，計算に使うベクトルの初期化を行う
(2) CPU から GPU に演算に用いるデータを転送する

ソースコード 1　GPU での演算

```
1 __global__ void calc(float *a, float *b, float *c, int n
      ){
2         int idx=blockIdx.x*blockDim.x+threadIdx.x;
3         float tmp=0.0;
4         for(int i=0; i<n; i++){
5                 c[idx*n+i] = a[i]*b[i*n+idx];
6                 tmp+=c[idx*n+i];
7         }
8 }
```

ソースコード 2　FPGA での演算

```
1 __kernel void calc(__global float *restrict A,
2                    __global float *restrict Z,
3                    const int NUMSTREAM){
4         int j=0;
5         for(int i=0; i<NUMSTREAM*
              NUMSTREAM; i++){
6                 Z[i/NUMSTREAM] = i\%
                      NUMSTREAM==0?A[i]:Z[i/
                      NUMSTREAM]+A[i];
7         }
8 }
```

(3) GPU の kernel を起動し内積の部分和を計算する
(4) GPU での演算結果 (部分和) は FPGA の演算で用いるため CPU に転送する
(5) GPU から CPU に転送された演算結果を FPGA に転送する
(6) FPGA で転送された部分和ベクトルの要素について総和を計算する
(7) FPGA の演算結果を CPU に転送する

GPU では，図 9 に示すように，行列 A の 1 行分と行列 B を全ての要素のデータを送り，それぞれの内積の部分和を各スレッドで求める．この内積の数は行列 B の全ての要素数と同じである．これを FPGA に送る．FPGA に送ったデータのイメージを図 10 に示す．FPGA では，送られてきた内積の部分和を行列 A の列の要素数分に区切り (図 10 の真ん中の黒枠)，内積の総和 (図 10 の右の黒枠) を求めることを行列 B の列の要素数回分行う．その結果，FPGA にデータを 1 度転送して求められる行列の積は行列 B の列の要素数分と同じである．

ベクトル内積の複合計算について，CPU で計算した結果と一致することを確認した．以上より，提案した CUDA と OpenCL の複合環境で，両アクセラレータを同時に正しく動作させ，かつ協調計算を実現可能であることが確認された．

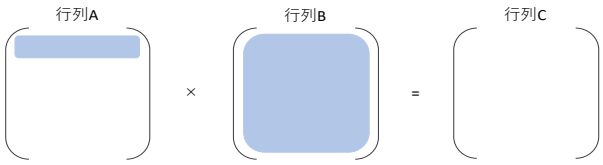

図 9　GPU へのデータ転送

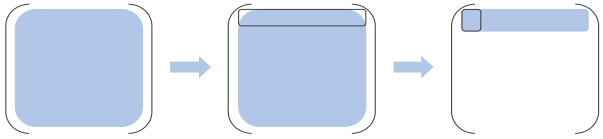

図 10　FPGA へのデータ転送

GPU における CUDA 記述での内積の計算を行い，また，そのデータを FPGA に渡し，OpenCL で記述した内積の加算を行った結果を，CPU 上で行った結果と一致することを確認した．以上により，今回作成した CUDA と OpenCL の統合プログラム環境で，両アクセラレータデバイスを正しく動作させ，かつデータを連携させることができることが確認された．

本報告では，GPU と FPGA の間のデータ交換は cudaMemcpy による GPU-CPU 間データ転送と clEnqueue による FPGA-CPU 間データ転送を連続して実行して行っているが，PCIe 上のデバイスである GPU と FPGA の間で DMA 転送を行う手法も研究を進めている [4]．今後は本研究のプログラミングフレームワークにこの手法を取り入れてより高速なデバイス連携研究に展開する予定である．

6. まとめ

GPU と FPGA という異なるデバイスが共存する環境上で，2 つの異なるコンパイル環境を使い分けて 2 つの異なる言語を協調させる方法を見つけ，動作を確認した．動作確認の結果，正しい結果が得られたので GPU と FPGA を連携させて演算した場合も得られる結果は正しくなることが確認できた．前述のとおり，GPU が不得手な処理を多く含むプログラムの高速化に有効であると考えられている．また，本研究では，GPU-FPGA 間のデータ転送を CPU を介して行っていたが，GPU-FPGA 間直接通信のフレームワークが完成しているので，それを用いてさらなる高速化を図る．

2019 年 4 月より，筑波大学計算科学研究センターに GPU+FPGA 混載ノードを備えた次世代ハイブリッドクラスタである Cygnus が導入される．それはヘテロジニアスなシステムであり，本稿で示されたプログラミング環境はこの資源を有効利用する第一歩である．

今回の検証ではは GPU と FPGA を連携させたときに正しく動作するかの確認だったが，実アプリケーションでも高速化を図ることが可能かということが今後の課題となる．今後の課題として，GPU+FPGA の複合システムを実アプリケーションに実装する．そのために，実際にコードを書いてデータのやりとりやカーネル呼び出しのオーバーヘッドなどを確認すると同時に，性能評価を行い，有用性を確かめる．現在，このプログラム環境の性能を発揮できる実アプリケーションとして想定されているのは，宇宙物理のアプリケーションである ARGOT[12] である．ARGOT は GRAVITY，ARGOT，ART から構成されるアプリケーションで現在は ARGOT と ART を GPU で行っている．しかし，ART 部分が全実行時間の大部分を占めているため，その部分を FPGA に行わせることで，アプリケーションの高速化を図る．

本研究では，CUDA と OpenCL という，両デバイスの高レベルプログラミングとして現実的に組み合わせ可能なプラットフォームを対象としている．しかし，アプリケーションユーザにとってはこの環境は非常に複雑であり，より統一的かつ機能的なプログラム記述ができることが理想である．このため，我々は統一的な言語プラットフォームとして，OpenACC によって GPU と FPGA のそれぞれのコードを記述し，これを Cygnus のような環境で利用する枠組みの開発も目指している．

謝辞　本研究は，文部科学省「次世代領域研究開発」（高性能汎用計算機高度利用事業費補助金） 次世代演算通信融合型スーパーコンピュータの開発の一環として実施したものである．また，本稿の執筆に当たりご協力頂いた筑波大学計算科学研究センター廣川祐太氏に深く感謝する．

参考文献

[1] June 2018 — TOP500 Supercomputer Sites `https://www.top500.org/lists/2018/06/`

[2] Y. kodama, T. Hanawa, T. Boku, M. Sato, ”PEACH2: FPGA based PCIe network device for Tightly Coupled Accelerators”, Proc. of HEART2014, Sendai, 2014. (receiving Best Paper Award of HEART2014)

[3] Chiharu Tsuruta, Yohei Miki, Takuya Kuhara, Masayuki Umemura and Hideharu Amano, “Off-loading LET generation to PEACH2: A switching hub for high performance GPU clusters”, Proc. of the International Symposium on Highly-Efficient Accelerators and Reconfigureable Technologies (HEART), May 2015.

[4] Ryohei Kobayashi, Norihisa Fujita, Yoshiki Yamaguchi, Taisuke Boku, “OpenCL-enabled high performance direct memory access for GPU-FPGA cooperative computation”, Proc. of IXPUG Workshop Asia 2019 (in HPC Asia 2019), Guangzhou, Jan. 14th, 2019.

[5] Norihisa Fujita, Ryohei Kobayashi, Taisuke Boku, Yuma Oobata, Yoshiki Yamaguchi, Kohji Yoshikawa, Makino Abe, Masayuki Umemura, ”Accelerating Space Radiative Transfer on FPGA using OpenCL”, Proc. of HEART2018 (Int. Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies), Toronto, Jun. 21st 2018.

[6] Seyong Lee, Rudolf Eigenmann, ”OpenMPC: Extended OpenMP programming and tuning for GPUs”, Proceedings of the 2010 ACM/IEEE International Conference

for High Performance Computing, Networking, Storage and Analysis
[7] Seyong Lee, Jungwon Kim, Jeffrey S Vetter, "OpenACC to FPGA: A Framework for Directive-based High-Performance Reconfigurable Computing", 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)
[8] Kuen Hung Tsoi, Wayne Luk: Axel: a heterogeneous cluster with FPGAs and GPUs. FPGA 2010: 115-124
[9] CUDA とは？ - nVIDIA https://www.nvidia.co.jp/object/cuda_whatis_jp.html
[10] OpenCL Overview - The Khronos Group Inc https://www.khronos.org/opencl/
[11] Intel FPGA SDK for OpenCL Pro Edition: Programming Guide https://www.intel.com/content/www/us/en/programmable/documentation/mwh1391807965224.html
[12] Okamoto T., Yoshikawa K., Umemura M., "ARGOT: Accelerated radiative transfer on grids using oct-tree", Monthly Notices of the Royal Astronomical Society/419/p.2855-2866, 2012-01