Vol.2019-HPC-169 No.10
2019/5/10

GPU-FPGA 協調計算を記述するための プログラミング環境に関する研究

綱島隆太[1,a)]　小林諒平[2,1]　藤田典久[2]　中道安祐未[1]　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）分野におけるトップレベルのマシンには，アクセラレータを搭載した大規模計算クラスタが多く含まれている．高い演算性能とメモリバンド幅を有する Graphics Processing Unit（GPU）がアクセラレータとして主に用いられているが，条件分岐が頻出する処理や多数の演算コアが利用できないような並列性の小さい処理といった GPU の不得手する演算は依然として存在し，それが性能向上の妨げとなっている．このような問題に対し，任意の論理回路をプログラム可能な集積回路である Field Programmable Gate Array（FPGA）に，GPU が不得手とする処理を実行する回路を実装し，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アプローチを我々は試みている．しかしながら， GPU と FPGA の演算カーネルは，それぞれ CUDA と OpenCL といった異なるプログラミング言語で開発する必要があり，このようなマルチリンガルプログラミングは，ユーザーにとって多大な負担となる．そこで本研究では，GPU と FPGA が搭載された計算機システム上にて，両アクセラレータの統合的な制御を可能にする OpenACC を用いたプログラミング環境について検討する．本報告では，OpenACC により記述された別々の GPU 向け，FPGA 向けファイルをコンパイル時にリンクすることで両アクセラレータの連携が可能か検証を行った．その結果，OpenACC による記述のみで GPU-FPGA 協調計算が実現可能であることを確認した．

1. はじめに

大規模並列計算システムを用いて科学計算やシミュレーションなどの高速演算を行う HPC 分野では，CPU の他にアクセラレータを搭載することがある．世界的なスーパーコンピューターの性能ランキングである TOP500 では，上位 10 機のうち 6 機のシステムでアクセラレータが利用されている [1]．このアクセラレータには主に GPU が利用されている．

GPU は本来，画像を描画する際に必要な計算処理を行うプロセッサであるが，CPU に比べて非常に高い並列処理演算性能と電力効率を持っていることから（**表 1**），GPU を様々な汎用計算に用いる General Purpose GPU（GPGPU）がさかんに行われている．しかし，GPU には，条件分岐が頻出，あるいはデータレベルの並列性が低い処理では GPU の多数の演算コアを有効活用できない点や，データ交換やリダクションのような処理のためにノードを跨ぐ GPU 同士で通信が発生する場合では，CPU を介した複数回のメモリコピーが必要であるため通信レイテンシが増加したり、通信のたびにカーネルとホストのスイッチングが発生するといったデメリットが存在し，これらはアプリケーションの性能劣化の原因となる．

その一方で，近年はアクセラレータとして FPGA が注目されている．FPGA は，内部の論理回路を再構成可能なハードウェアであり，何度でも回路を書き換えられる．FPGA の利点として以下のようなものが挙げられる．

- アプリケーションに適した回路を自由に設計可能
- パイプライン処理によって高速化が可能
- 近年の高性能 FPGA 間は CPU を介さずに高速な通信が可能
- 実装依存であるが，GPU より省電力化が可能

デメリットとして，アプリケーションの実行時に，利用できるリソースが限られることが挙げられる．これは，FPGA がノイマン型アーキテクチャのようにメモリにプログラムを格納して，行える処理を動的に制御できるプログラミングモデルではなく，計算前に回路が固定されてしまうことによる．しかし，回路規模の大きな FPGA が登場してきているため，以前と比較して柔軟な回路設計が可能となっている．

GPU の不得意な計算を FPGA に行わせることで，実アプリケーションのさらなる高性能化が期待できる．そ

[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
[a)] tsunashima@hpcs.cs.tsukuba.ac.jp

表 1　CPU と GPU の性能比較 [2]

	CPU(Xeon Gold 6148)	GPU(V100)
電力効率 [GFLOPS/W]	4.5	15
演算性能 [TFLOPS]	1	7

こで，我々は新しいタイプの PC クラスタとして GPU と FPGA を組み合わせたマルチアクセラレータシステムを提案し，GPU-FPGA 協調計算の実現を目指して研究を進めている [3].

しかし，現状では GPU と FPGA のプログラムはそれぞれ異なるプログラミング環境で開発しなければならない．最も多用されている NVIDIA GPU のプログラミングには CUDA[4] が利用されるのが一般的である。．一方で FPGA プログラミングには，Verilog HDL，VHDL といったハードウェア記述言語（HDL: Hardware Description Language）を用いた開発に加え，C 言語や OpenCL[5] を用いて論理回路を生成する高位合成を行う環境が提供されている．ただし，アプリケーション開発者が記述することを前提とした場合，高位合成の方が難易度は低い．現在の開発環境で GPU-FPGA 協調計算を実現するためには，全く異なる複数の開発環境を組み合わせなければならず，ユーザーにとって非常に負担が大きい．

以上の問題から，本研究では，GPU と FPGA の両方を搭載した計算機において，両アクセラレータ上で実行する処理を，OpenACC により統一的に記述できるプログラミング環境について検討する．OpenACC は，ディレクティブ形式のプログラミング環境を提供するため，上記の開発環境に比べて記述が容易である．本研究ではコンパイラに，GPU は PGI Compiler，FPGA は OpenARC[6] をそれぞれ用いる．OpenARC は OpenACC から FPGA 向け OpenCL コードを出力する，ソース to ソースのコンパイラである．本研究を実現するうえで以下の課題が挙げられる．

- GPU と FPGA のコンパイル環境の組み合わせ
- オフロード部を処理するデバイスの明示手法
- FPGA を主体とした CPU を介さないデバイス間直接通信の記述手法

本報告では，GPU-FPGA 協調計算を OpenACC により記述された一つのプログラムにより実現するためのプログラミング環境を提案，議論する．これまでに，それぞれ別々に記述された GPU と FPGA 向けの OpenACC プログラムをコンパイル時にリンクし，連携動作の検証を行った．これにより，OpenACC を用いた GPU-FPGA 協調計算が実現可能であることを確認している．

2. 関連研究

[7] では，筑波大学計算科学研究センター（CCS）で GPU 間直接通信機構として提唱された TCA（Tightly Coupled Accelerators）アーキテクチャ [8] を実装した PEACH2（PCI Express Adaptive Communication Hub version 2）による通信処理のオフロードを行っている．PEACH2 は GPU に足りない通信処理を行う FPGA ボードであり，異なるノード間の GPU 同士を直接接続できるようになっている．これによって，ホストの CPU とメモリを介さずに通信を行えるため，転送速度が改善される．Tsuruta らは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構成し，ノード間通信を行う際に CPU で行う処理のオフロードを行った．N 体シミュレーションを対象に，GPU 間通信の前処理となるツリー法の枝刈り部分を FPGA 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．

HPC 分野において，GPU はアクセラレータとして最も用いられているが，並列処理演算性能が高い一方で通信性能が貧弱なことから，全体性能のボトルネックとなることがある．一方で，ノード間でも高速なデバイス間通信を可能としたハイエンドモデルの FPGA をアクセラレータとすることが，HPC 分野で増えている．近年では，FPGA の開発環境として，OpenCL を用いた高位合成を FPGA ベンダーが提供していることにより，従来であれば容易でなかったアプリケーション開発者による FPGA の論理回路の実装が現実的になってきている．[9] では，OpenCL を用いて FPGA 上での Authentic Radiation Transfer（ART）法の最適化を行っている．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ている．複数の FPGA を使用し並列化した場合は，GPU を超える性能を発揮すると考えられる．[10] では，OpenCL を用いて実装した疎行列数値計算の性能評価と最適化方法について評価している．この中で FPGA 向けのプログラミングでは GPU とは異なる最適化を行う必要があると述べられている．[11] では，FPGA 上に津波シミュレーションの専用回路を構築し，性能評価を行っている．これにはストリーム計算用高位合成コンパイラである SPGen を用いている．

小林らは，GPU-FPGA 複合システムにおけるアクセラレータ間の連携機構ついて検証を行った [12]．計算と通信処理を FPGA で行い協調計算を行わせるため，GPU-FPGA 間直接通信の性能を評価した．結果，従来この通信の起動は CPU が行わなくてはならなかったが，FPGA が主体的に通信を起動できることが確認された．

これらの研究では FPGA を GPU の通信補助手段，あるいは FPGA のみの演算と通信の融合を試みているが，GPU と FPGA の協調動作を統一して記述するためのプログラミング手法については検討されていない．

AiS（Accelerators in Switch）[13] は FPGA を演算と通信に積極的に利用するコンセプトであり，GPU による高速演算を組み合わせることで，理想的な高性能並列処理システムが構築できると考えられる．例えば，通信の遅延を隠

蔽するために通信中に FPGA 上で演算を行ったり，GPU の苦手な処理を FPGA が行うことで高速化を図ったりする連携が考えられる．しかし現状，FPGA と GPU の協調計算を行うためのプログラミングには以下のような課題がある．

- 開発に用いられるプログラミング言語がアクセラレータごとに別々
- 同一のプログラムで異なる複数のデバイスを動かすことができない
- どちらも独自の機能を使用していることにより移植性が低い
- メモリアクセスや並列化手法などコーディングにおいて考慮すべきことが多い

OpenACC[15] は複数のアクセラレータ向けプログラミング標準である．ディレクティブ形式の言語拡張規格であり，計算科学アプリケーションが標準的に利用する C，C++，Fortran の各言語に対応している．利点として以下が挙げられる．

- 指示文の挿入のみでアクセラレータでの演算が可能
- CUDA と比べ抽象化されており移植性が高い

OpenACC では，CUDA や OpenCL のようにホストコードとデバイスコードという区別は無い．オフロードさせたい部分にディレクティブの挿入を行うだけで済むため，処理を別々のコードとして分けて記述する手間を省くことができる．

本研究では，OpenACC により，GPU-FPGA 協調計算を実現する開発環境の実現を目指す．

OpenACC のコード例を図 **1** に示す．まず，`#pragma acc data` で転送するデータを指定する．`copyin()` でホストからアクセラレータへ転送するデータを，`copyout()` でアクセラレータからホストへ転送するデータを指定する．転送するタイミングはディレクティブを挿入した位置となる．次に，`#pragma acc kernels` でアクセラレータが処理を行う領域を指定する．なお，この指示文のみでも OpenACC のプログラムはアクセラレータでの演算実行を行えるが，指定されていない処理はコンパイラ依存となり，特にアクセラレータとホスト間のデータ転送でオーバーヘッドが発生する可能性があるため，通常は`#pragma acc data` も挿入する．`#pragma acc loop` は個々のループに対する指示を行う．`independent` は，ループにデータ依存性が無いことを示す節である．これを指定しない場合，コンパイラの判断により適切な並列化が行えない場合がある．

```
1．転送するデータの指定
   #pragma acc data copyin(in1[0:N], in2[0:N])
   copyout(out[0:N]) {
2．カーネル実行対象の指定
   #pragma acc kernels
3．個々のループに対する指定
   #pragma acc loop independent
       for (i = 0; i < N; i++) {
           out[i] = in1[i] * in2[i];
       }
   }
```

図 1　OpenACC のコード例

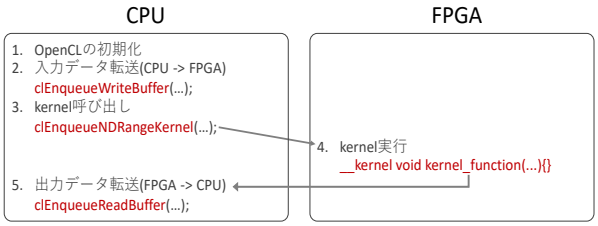

図 2　OpenCL プログラミングの流れ

3. 既存のプログラミング環境

3.1 FPGA

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．これらの 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．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く，HDL を用いた FPGA のプログラミングコストは高い．

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 を用いた高位合成による FPGA のプログラミング環境が提供されている．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．OpenCL は C 言語ベースでデバイスコードを実装できるため，HDL を用いた回路生成に比べ，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 **2** に示す．OpenCL プログラミングでは，CUDA と同様にホストコードとデバイスコードの 2 種類のプログラムが存在する．ホストコードは，CPU 上で動作し FPGA や GPU といったデバイスを制御するプログラムである．デバイスコードは，デバイス上で動作するプログラムである．しかし，OpenCL では，ホストコードとデバイスコードは別ファイルに記述し，別々にコンパイルする．ホストコードは gcc や Intel Compiler などの C コンパイラでコンパイルし，デバイスコード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．なお，OpenCL は GPU でも利用

できるものの，GPU の機能を全て利用することはできず，HPC アプリケーションにおいては，CUDA が GPU プログラミング環境のデファクトスタンダードとなっている．また，OpenCL のほうがホストコードの記述が複雑であるため，OpenCL による統一的な記述ではアプリケーション開発者にとってデメリットが多い．

3.2 GPU と FPGA のコンパイル環境の組み合わせ

我々は，GPU と FPGA の計算が混在したプログラムの実装，協調計算は可能であることを確認している [14]．これは，CUDA と OpenCL で記述したプログラムを分割コンパイルし，リンクすることで一つのプログラムとして実行できるようにしている．しかし，現状では GPU と FPGA で用いるプログラミング言語が異なるため，複数言語で記述しなければならず，実装が極めて難しい．CUDA と OpenCL ではプログラミングの作法が異なる部分が存在する．カーネルの並列度の指定方法が異なっており，また，OpenCL では clSetKernelArg() をカーネル引数毎に実行する必要があるため，同じように記述した場合バグを発生させる原因になる．さらに，2 で述べたとおり，GPU と FPGA ではプログラミングにおける最適化手法が異なる．FPGA ではリソースが限られているため，GPU のような非常に高い並列度を持ったプログラムは向いていない．したがって，これらのアクセラレータの処理を統一的に記述できるプログラミング環境を用意する必要がある．

3.3 各アクセラレータ向けの OpenACC コンパイラ

GPU 向けの OpenACC コンパイラには，PGI Compiler[16], GCC, Cray Compiler, OpenARC などが存在する．その中でも，PGI Compiler は NVIDIA 傘下の PGI 社が開発するコンパイラであり，NVIDIA 製 GPU に特に最適化されている．

一方で，FPGA に対応した OpenACC コンパイラは存在しない．しかし，研究開発中のものとして，FPGA に対応した OpanACC コンパイラとして OpenARC が存在する．. OpenARC は米国オークリッジ国立研究所（ORNL）が開発しているソース to ソースコンパイラであり，OpenACC を FPGA 向け OpenCL へ変換する機能を備える．本研究では，ORNL との共同研究により GPU-FPGA 協調計算環境の実現に OpenARC を利用する．OpenARC の OpenACC は，FPGA に限らず，GPU やメニーコア CPU 向けのプログラムも出力できる．NVIDIA 製 GPU 向けのコンパイルでは，OpenACC のコードを CUDA へ変換する．

しかし，現状では，異なる複数のアクセラレータを用いたプログラムの記述は不可能である．OpenACC では規格上，アクセラレータごとに異なる処理を記述可能になっているが，異なる複数のアクセラレータの利用を同時に行うことが想定されたコンパイル環境は存在しない．したがって，**図 3** (a) のように，単一のアクセラレータ向けのコードを記述することは可能だが，(b) のように一つのプログラム内で異なる複数のアクセラレータを使用する書き方はできない．協調計算を実現する場合，FPGA と GPU それぞれのプログラムの記述を行い，分割コンパイルした上でオブジェクトファイルをリンク，グルーコード（互換性の無い部分同士を結合するためだけに働くコード）で繋ぐ必要がある．これはユーザーにとって負担となるため，一つのプログラム内で協調計算を記述可能にし，ユーザの利便性を高める必要がある．

(a) 別々のプログラム　(b) 一つのプログラム

図 3 OpenACC コンパイラの記述対応状況

4. 統一的な記述を可能とするプログラミング環境の提案

3.3 で述べたとおり，複数のアクセラレータの処理を一つのプログラム上で実行させることに対応した OpenACC コンパイラは存在しない．しかし，一つのプログラム内に複数のアクセラレータの処理を記述可能にすることで，ユーザーの実装負担を軽減したい．よって，本研究では，**図 4** に示すコンパイル手法を提案する．まず，OpenACC により GPU，FPGA それぞれの処理を含めたプログラムを記述する．次に，記述したプログラムを既存のコンパイラでコンパイルできるように，それぞれのアクセラレータ向けにファイル分割する．このファイル分割の処理は OpenACC の入力から OpenACC のファイルを出力するソース to ソースの変換とする．この変換を行うために，理化学研究所計算科学研究センターと筑波大学 HPCS 研究室で共同研究中の OpenACC に対応したコンパイラである Omni Compiler[17] を拡張し，専用のトランスレーターを開発する．分割されたファイルはそれぞれのデバイスに対応する OpenACC コンパイラで分割コンパイルを行い，オブジェクトファイルをリンクすることで一つのプログラムとして実行できるようにする．

5. OpenACC による GPU+FPGA 複合プログラム記述の検証

5.1 オブジェクトファイルのリンクによる協調動作の確認

トランスレーターを開発する前段階として，GPU，FPGA それぞれの処理が記述された 2 つの OpenACC プログラムが協調動作可能か確認を行った．確認には，**トランスレーターの出力を想定して記述された GPU，FPGA 向けの 2**

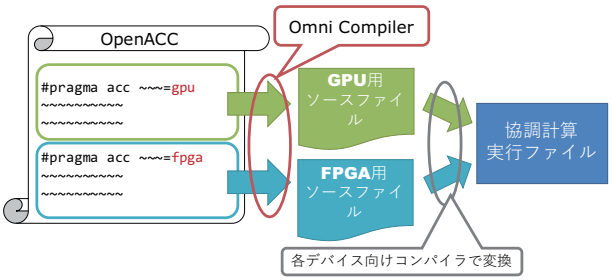

図 4　提案するプログラミング環境の概要

つの OpenACC プログラムを用いた．なお，OpenARC がサポートする OpenACC が C 言語のみであるため，今回の検証で用いたプログラムはすべて C 言語で記述されている．

5.2　コンパイラの組み合わせ

当初，本研究では GPU 向け OpenACC ファイル，FPGA 向け OpenACC ファイルの両方とも OpenARC でコンパイルすることを検討していた．しかし，OpenARC では，OpenACC のコードを変換する際に，独自のランタイムライブラリに変換を行っており，直接 CUDA 及び OpenCL のデバイスコードを呼び出していない．このランタイムライブラリによるデバイスコードの呼び出しは，どのデバイスでも共通の関数を使用していて複数のデバイスが 1 つのプロセスで同時に使用されることを想定していないため，ホストコードが書かれた GPU，FPGA それぞれ 2 つのオブジェクトファイルのシンボルが重複してしまい，リンクすることができない．そこで，FPGA 向け OpenACC ファイルには OpenARC を使用し，GPU 向け OpenACC ファイルにはシンボルが重複しないと予想される PGI Compiler を使用した．コンパイルの流れを図 **5** に示す．なお，OpenARC でコンパイル後に出力される FPGA 向け OpenCL プログラムは，Intel 社が提供している Intel FPGA SDK for OpenCL[18] に付属する，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 コマンドを用いてコンパイルした．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，実行を行う．

5.3　検証プログラムの実行

検証は，CCS で稼働している，Pre-PACS version X（PPX）（表 **2**）上で行った．AiS の技術実証機として PPX は開発されている．なお，ノードのうち図 **7** の CPU0 側の左側赤枠部分を用いた．

検証プログラムの処理の流れを図 **6** に示す．プログラムは大きく分けてホストで実行するコードブロック，GPU で実行するコードブロック，FPGA で実行するコードブロックに別れている．まず，ホストでデータの初期値をセットし，GPU で処理を実行する．その後，データを FPGA に

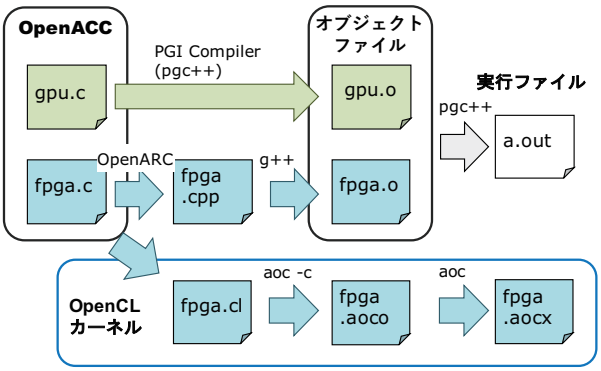

図 5　コンパイルの流れ

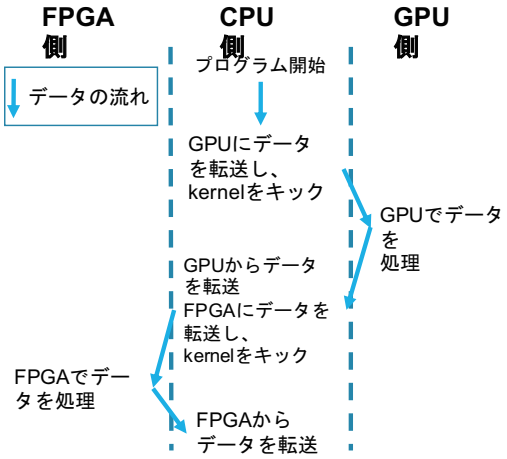

図 6　処理の流れ

表 2　評価環境

CPU	Intel(R) Xeon(R) CPU E5-2660 v4 x2
GPU	NVIDIA P100 x2 (PCIe Gen3 x16)
FPGA	Intel Arria 10 GX 1150 (BittWare A10PL4[19]) (PCIe Gen3 x8)
OS	CentOS 7.3
GPU compiler	PGI Compiler 18.10
FPGA compiler	OpenARC V0.11 (Jan, 2018)
OpenCL compiler	Intel FPGA SDK for OpenCL 17.1.2.304

渡して FPGA で処理をしたあと，ホストにデータを返し，結果を出力する．なお，データは GPU から FPGA へ直接転送されるわけではなく，GPU から一旦ホストのメモリに転送された後，FPGA に転送される．

GPU では，図 **8** のコードを実行した．ベクトルの和の計算を行っている．FPGA では，図 **9** のようにデータのコピーのみ実行した．ベクトル `a` は GPU で計算した結果を FPGA に転送したものであり，ベクトル `b` がホストに返される．なお，処理内容は実際に想定される処理を行うことも考えられるが，ここでは問題を単純化するため，複雑な処理はせずに検証しやすいコードとした．また，計算

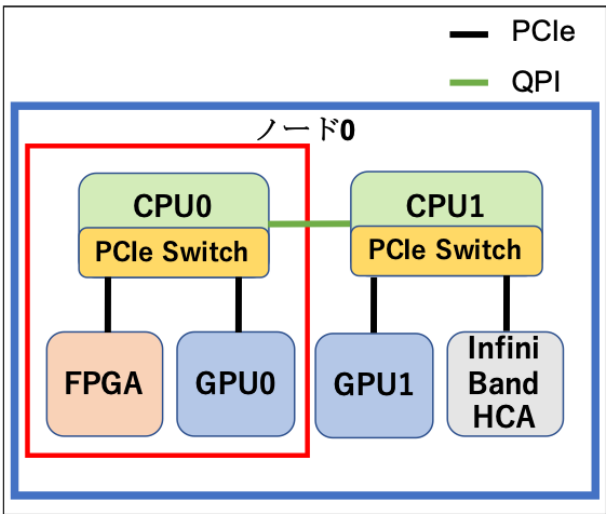

図 7　PPX のノード内構成

```
1  #pragma acc data copyin(A[0:size], B[0:size])
       copyout(D[0:size]) {
2  #pragma acc kernels loop independent gang worker
       (16)
3          for (i = 0; i < size; i++) {
4              D[i] = A[i] + B[i];
5          }
6      }
```

図 8　GPU での演算

```
1  #pragma acc data copyin(a[0:size]) copyout(b[0:size])
2  #pragma acc kernels
3  #pragma acc loop independent
4      for (j = 0; j < size; j++) {
5          b[j] = a[j];
6      }
```

図 9　FPGA での演算

結果は CPU 上で計算した結果と GPU，FPGA から返ってきた値が同じであるか，それぞれ確認を行った．

実行した結果，GPU，FPGA からホストへ返された値と CPU での演算結果は一致していることが確認できた．以上より，OpenACC による統一プログラミング環境から両アクセラレータを同時に正しく動作させ，GPU-FPGA 協調計算を実行可能であることが確認された．

5.4　言語間のプログラム記述量の比較

本検証で用いたプログラムは，FPGA 向けファイル，GPU 向けファイル，ヘッダーファイルを合わせて 85 行，1105 文字で記述されている．しかし，CUDA や OpenCL で記述した場合はさらに記述量が増える．特に，デバイスコードは OpenACC で記述されたオフロードするコードブロックとほぼ同じ記述量になるが，ホストコードは OpenACC では必要の無いデバイスの初期化やメモリの確保などの処理を大量に記述する必要がある．これらの処理はどのプログラムでも共通に，毎回書かなければならないため，この分の記述量が OpenACC のコードに追加されることになる．さらに，CUDA と OpenCL を組み合わせた場合，2 つのホストコードに必要な記述をどちらも書かなければならない．

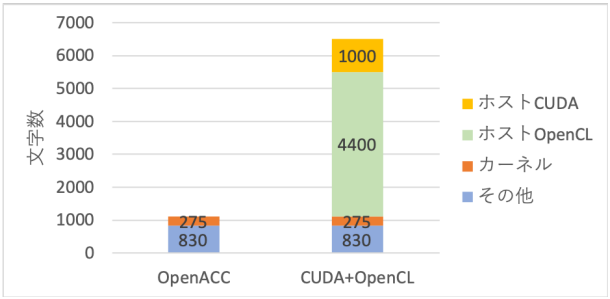

図 10　必要となる記述量のおおよその比較

我々が実行した CUDA+OpenCL による GPU-FPGA 協調計算プログラムでは，OpenACC で記述する必要の無いホストコードの処理に 240 行以上，5000 文字以上を要している．これを参考に本検証で用いたプログラムを CUDA+OpenCL で実装した場合のおおよその記述量の比較を図 **10** に示す．カーネルは CUDA と OpenCL のカーネルコードを合わせた記述量，その他はホストで行う変数の宣言，初期化，標準出力などの記述量である．これらは，CUDA+OpenCL で実装した場合も OpenACC の場合と同様に記述することになる．また，ホスト CUDA，ホスト OpenCL は OpenACC で記述する必要の無いホストコードのおおよその記述量である．CUDA で必要となるホストコードに対し，OpenCL で必要となるホストコードの方が多いということがわかる．本検証で用いたプログラムを CUDA+OpenCL で記述した場合は約 330 行，約 6500 文字の記述を要する．以上より，OpenACC では CUDA+OpenCL と比較して大幅に削減された約 1.5 割の記述量で，GPU-FPGA 協調計算が記述可能である．

6.　FPGA の言語間における性能比較

参考までに，OpenACC と OpenCL の演算速度を比較した．それぞれの言語で記述された行列積演算を行う FPGA 向けプログラム（図 **11**，図 **12**）を実行し，それぞれの Flops を測定した．図 11 のコードは OpenARC によって図 **13** のような OpenCL のカーネルコードに変換される．なお，プログラム中の N の値は 10000 として 10000 × 10000 の正方行列同士で掛け算を行い，変数は倍精度浮動小数点数を用いた．実行環境は 5.3 と同じである．

実行結果を図 **15** に示す．結果より，OpenACC のコードの演算速度は OpenCL のコードの 6 割程度であること

```
#pragma acc kernels loop independent gang worker
    collapse(2) copyout(a[0:(N*N)]), copyin(b[0:(N*N
    )],c[0:(N*N)])
  for (i=0; i<N; i++){
    for (j=0; j<N; j++)
      {
        double sum = 0.0 ;
#pragma acc loop seq
          for (k=0; k<N; k++) {
            sum += b[i*N+k]*c[k*N+j] ;
      }
          a[i*N+j] = sum ;
      }
  }
```

図 11　OpenACC の性能比較コード

```
__attribute__((reqd_work_group_size(1,1,1)))
__kernel void matMul(__global double *restrict a,
    __global const double *restrict b, __global
    const double *restrict c, const int N) {

  int i, j, k ;

  for (i=0; i<N; i++){
    for (j=0; j<N; j++)
      {
        double sum = 0.0 ;
        for (k=0; k<N; k++) {
          sum += b[i*N+k]*c[k*N+j] ;
      }
          a[i*N+j] = sum ;
      }
  }
}
```

図 12　OpenCL の性能比較コード

```
__kernel void __attribute__((reqd_work_group_size(32,
    1, 1))) MatrixMultiplication_openacc_kernel0(
    __global double * restrict a, __global double *
    restrict b, __global double * restrict c, int M,
    int N, int P)
{
double sum;
int lwpriv___ti_100_0;
int lwpriv__i;
int lwpriv__j;
int lwpriv__k;
lwpriv___ti_100_0=get_global_id(0);
if (lwpriv___ti_100_0<(M*N))
{
sum=0.0;
lwpriv__j=(lwpriv___ti_100_0%N);
lwpriv__i=(lwpriv___ti_100_0/N);
for (lwpriv__k=0; lwpriv__k<P; lwpriv__k ++ )
{
sum+=(b[((lwpriv__i*P)+lwpriv__k)]*c[((lwpriv__k*N)
    +lwpriv__j)]);
}
a[((lwpriv__i*N)+lwpriv__j)]=sum;
}
}
```

図 13　図 11 を OpenCL に変換したもの

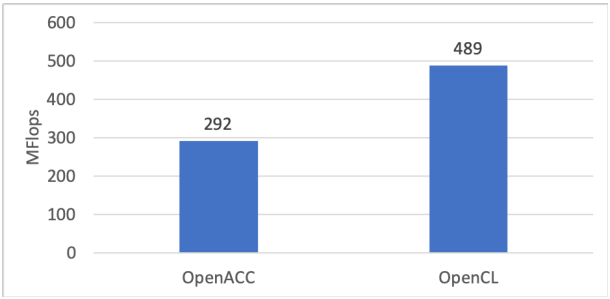

図 14　演算速度の比較

がわかった．しかし，動作周波数の比較（図 15）では，OpenACC のほうが高かった．動作周波数が高いほど演算速度も高くなるはずであり，行列のサイズを 512 × 512 で実行した場合の演算速度は，OpenACC のほうが高速だった．図 13 を参照すると，FPGA で推奨されているシングルワークアイテムによって処理されていないことがわかる．よって，計算を行うに連れてオーバーヘッドが累積して演算速度が下がったと考えられる．今回の OpenCL のコードはパフォーマンス・チューニングを行っていないため，プログラムを工夫することで更に性能差が大きくなる可能性があるが，そのためには FPGA に関する高度な知識が必要である．また，本研究で使用している OpenARC は開発中であるため，今後演算速度が改善される可能性がある．なお，プログラム全体のコード量で比較した場合は，OpenCL はホストコードとデバイスコードを合わせて約 260 行，約 4850 文字であるのに対し，OpenACC は約 110 行，約 2450 文字であり，およそ半分まで削減できる．

7. 今後の課題

本報告の検証により，OpenACC を用いた GPU-FPGA 協調計算が可能であることが確認できた．これは本研究の目標とする，GPU-FPGA 協調計算の統一開発環境の実現に向けた第一歩である．しかし，同一のプログラムファイル内に GPU と FPGA 両方のコードを含められるプログラ

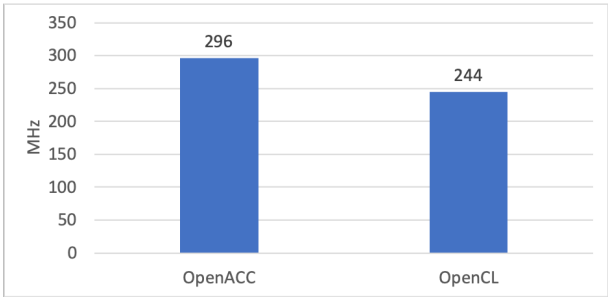

図 15　動作周波数の比較

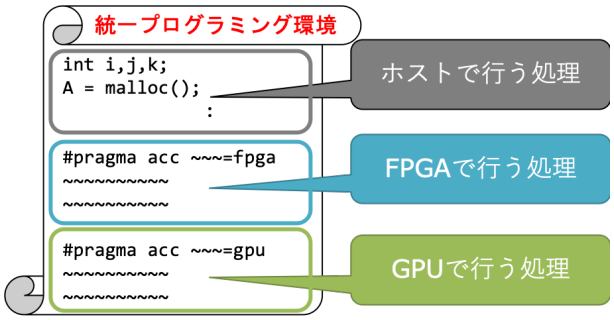

図 16　統一プログラミング環境のイメージ

ミング環境は実現できていない．本研究の目標とするプログラミング環境では，図 16 のようにプログラムの一連の処理を一つのプログラム内に含められるようにしたい．4 章で述べたとおり，今後は，ソース to ソースの変換により，両演算加速デバイスの処理が含まれた一つの OpenACC プログラムファイルを GPU 対応 OpenACC コンパイラで処理可能なプログラムファイルと，FPGA 対応 OpenACC コンパイラで処理可能なプログラムファイルに分割するトランスレーターの開発を行っていく．

この際に，単純にコードブロックだけをファイル分割するのではなく，最終的にこれらのファイルを分割コンパイルし，リンクしたときに正しく動作するように整合性が取れたコードに整形した上でファイル分割する必要がある．よって開発過程でこの検証についても行っていく予定である．

なお，トランスレーターとして利用する OmniCompiler は大規模並列計算機で利用することを想定した XcalableMP，XcalableACC，OpenACC といった指示文を含むコードを対象としたコンパイラである．現状 OmniCompiler で OpenACC をコンパイルした場合，CUDA または PEZY-SC プロセッサ向け OpenCL 拡張である PZCL を出力する．しかし，FPGA 向け OpenCL へのコンパイルには対応していない．また，最新の GPU アーキテクチャには対応しておらず，PGI Compiler と同程度に最適化されたコードを生成可能にすることは容易ではない．したがって，OmniCompiler ではソース to ソースの変換のみ行い，実行ファイルへの変換には各アクセラレータに対応したコンパイラを利用する手法をとる．このために，OpenACC の出力を可能にするための開発を進めていく．

また，本報告では，GPU-FPGA 間のデータ転送はホストのメモリを一旦介して行っていたが，我々は関連研究で述べた GPU-FPGA 間で直接通信を行う手法についても研究を進めている．これを利用することでさらなる高速化が可能なことから，OpenACC からこのフレームワークを利用可能にすることも課題として挙げられる．

8. まとめ

本報告では，GPU と FPGA の両方を搭載した計算機における両演算加速デバイスの利用を，OpenACC により統一的に記述できるプログラミング環境について検討を行った．その結果，GPU 向けには PGI Compiler，FPGA 向けには OpenARC で OpenACC のコードを分割コンパイルし，それぞれで出力されたオブジェクトファイルをリンクすることで，同一規格により GPU と FPGA で同時に演算を行わせるプログラムを記述できることを確認した．この結果を発展させ，一つのファイルに同一規格で GPU と FPGA を同時に利用する記述が可能なプログラミング環境を実現するには，GPU と FPGA の処理が混在する 1 つの OpenACC プログラムファイルを，それぞれのデバイス向けに 2 つのファイルへ分割するトランスレーターを開発し，分割されたファイルを本報告で検証した手法でコンパイルできるようにする必要がある．

筑波大学計算科学研究センターでは，2019 年 4 月より 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[20] が稼働している．本報告で示したプログラミング環境はこの資源を有効利用するためのユーザーインターフェイスとして重要な技術である．

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．OpenARC の開発に携わっている米国オークリッジ国立研究所の Seyong Lee 博士，Jeffrey S. Vetter 博士を始めとする諸氏に感謝する．OmniCompiler を共同研究している理化学研究所計算科学研究センターの村井均博士，中尾昌広博士，佐藤三久教授に感謝する．また，PPX の利用を始めとして日頃からお世話になっている筑波大学計算科学研究センターの皆様に感謝する．最後に，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の廣川祐太博士に深く感謝する．

参考文献

[1] June 2018 — TOP500 Supercomputer Sites
https://www.top500.org/lists/2018/06/
[2] June 2018 — TOP500 Supercomputer Sites
https://www.top500.org/green500/lists/2018/06/

[3] 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. 研究報告ハイパフォーマンスコンピューティング (HPC), 2018, 2018.11: 1-10.
[4] CUDA とは？ - nVIDIA https://www.nvidia.co.jp/object/cuda_whatis_jp.html
[5] OpenCL Overview - The Khronos Group Inc https://www.khronos.org/opencl/
[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] Tsuruta, C., Miki, Y., Kuhara, T., Amano, H., & Umemura, M. Off-loading let generation to peach2: A switching hub for high performance gpu clusters. ACM SIGARCH Computer Architecture News, 2016, 43.4: 3-8.
[8] Hanawa, T., Fujii, H., Fujita, N., Odajima, T., Matsumoto, K., & Boku, T. Evaluation of FFT for GPU cluster using tightly coupled accelerators architecture. In: Cluster Computing (CLUSTER), 2015 IEEE International Conference on. IEEE, 2015. p. 635-641.
[9] 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.
[10] 大島聡史, 塙敏博, 片桐孝洋, 中島研吾. FPGA を用いた疎行列数値計算の性能評価. 研究報告ハイパフォーマンスコンピューティング (HPC), 2016, 2016.1: 1-9.
[11] 佐野健太郎, 河野郁也, 中里直人, Alexander Vazhenin, Stanislav Sedukhin. FPGA による津波シミュレーションの専用ストリーム計算ハードウェアと性能評価. 研究報告ハイパフォーマンスコンピューティング (HPC), 2015, 2015.5: 1-7.
[12] 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アクセラレータ間連携機構. 研究報告ハイパフォーマンスコンピューティング (HPC), 2018, 2018.26: 1-8.
[13] 藤田典久, 小林諒平, 山口佳樹, 大畠佑真, 朴泰祐, 吉川耕司, 安部牧人, 梅村雅之. OpenCL を用いた FPGA による宇宙輻射輸送シミュレーションの演算加速. 研究報告ハイパフォーマンスコンピューティング (HPC), 2017, 2017.12: 1-9.
[14] 中道安祐未, 小林 諒平, 藤田 典久, 朴 泰祐. 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. 研究報告ハイパフォーマンスコンピューティング (HPC), 2019, 2019.10: 1-7.
[15] OpenACC, https://www.openacc.org
[16] PGI Compilers & Tools https://www.pgroup.com/
[17] Omni Compiler http://omni-compiler.org/
[18] Intel:Intel FPGA SDK for OpenCL, https://www.intel.co.jp/content/www/jp/ja/software/programmable/sdk-for-opencl/overview.html
[19] BittWare: A10PL4 PCIe FPGA Board, https://www.bittware.com/fpga/intel/boards/a10pl4/
[20] スーパーコンピュータ - 筑波大学 計算科学研究センター Center for Computational Sciences https://www.ccs.tsukuba.ac.jp/supercomputer/