Vol.2022-HPC-183 No.12
2022/3/17

oneAPIを用いたGPU・FPGA混載ノードにおける宇宙物理シミュレーションコードARGOTの実装

柏野隆太[1]　小林諒平[2,1]　藤田典久[2,1]　朴泰祐[2,1]

概要：GPU（Graphics Processing Unit）は，HPC 分野において最も広く用いられているアクセラレータの一つである．しかし，マルチフィジックスに基づく科学計算では単一のシミュレーションの中に多様なワークロードが出現し，GPU のみを用いた高速化では不十分である．我々は，このような複雑な物理シミュレーションを対象として，GPU と FPGA（Field Programmable Gate Array）の併用による高速化を目指し，CHARM (Cooperative Heterogeneous Acceleration by Reconfigurable Multidevices) というコンセプトの下，ハードウェア，プログラミングシステム，そしてアプリケーション開発をおこなっている．ここでの大きな課題は，これら複数のデバイスをどのようにプログラムするかである．近年注目されている Intel 社によって提案された oneAPI は，SYCL をベースにした DPC++による単一言語プラットフォームを提供し，複数のデバイス間における連携プログラミングが可能である．本稿では，GPU と FPGA を用いた宇宙物理シミュレーションコード ARGOT を oneAPI によって実装し，その性能評価について報告する．本研究の特徴は，oneAPI をその一般的な利用方法とは異なり，DPC++のみを用いた開発ではなく既存の CUDA や OpenCL によるプログラム部分コードを組み合わせるためのフレームワークとして用いている点である．結果として，oneAPI を用いることで，DPC++によるプログラミングだけでなく，CUDA や OpenCL など他の言語で記述された既存のソースコードを再利用して，複数のデバイスが協調するプログラムを実装することができることがわかった．

1. はじめに

高性能計算における対消費電力性能比の向上のため，アクセラレータは現代の HPC システムを牽引する最も重要な技術の一つである．特に，TOP500 リスト [1] の上位マシンのような多数の計算ノードを持つ大規模システムにおいて重要となっている．最も代表的なアクセラレータは GPU（Graphics Processing Unit）であり，複数かつ同一種類の演算を一度に計算することで高い演算性能を実現する SIMD 演算を基本とし，HBM2 などの高性能メモリとの親和性が高いという特徴をもつ．最新の GPU である NVIDIA Tesla A100[2] は，倍精度で最大 9.7 TFLOPS の理論ピーク性能と 2.039 TB/s の理論メモリバンド幅を実現する．近年の GPU は，主に AI やディープラーニングのアプリケーションに注目が集まっているが，気候・生物医学・天体物理学・量子物理学のモデリングなど，従来の HPC アプリケーションで最も利用されるアクセラレータであることには変わりがない．このように広く用いられている一方で，GPU は大量の演算を SIMD 演算により同時実行するという比較的単純なモデルに基づいて計算を行うため，その適用可能性に限界がある．

一般的に，以下のような条件をもつ実アプリケーションでは，GPU の性能を発揮することが困難である．

- 計算に含まれる演算量が不十分である場合
- 分岐条件による不規則な計算パターンとなる場合
- 頻繁に計算ノード間通信が発生し，CPU と GPU の実行を切り替える必要がある場合

一つのプログラム全体がこのような制約を持つことは稀であるが，プログラムコード上のある一部でこういった制約が発生することは十分考えられ，結果的にその部分での GPU 演算効率が落ちることで，その部分がアムダール則に基づく性能ボトルネックとなりやすい．これらは，GPU を用いた高速化において大きな課題となっている．これに対し，FPGA（Field Programmable Gate Array）は，用途に応じて内部回路の書き換えが可能なアクセラレータであり，HPC 分野における新しいアクセラレータとして注目されている．このデバイスの利点は以下の通りである．

- パイプライン並列による時間方向の並列化が可能である
- 消費電力あたりの性能において GPU を上回ることが

[1] 筑波大学 情報理工学位プログラム
[2] 筑波大学 計算科学研究センター

表 1　GPU および FPGA の特徴比較

アクセラレータ	GPU	FPGA
並列性	SIMD+空間並列	パイプライン並列
メモリ性能	強い (大容量 HBM2)	弱い (DDR4 or 小容量 HBM2)
条件分岐	弱い	強い
細粒度処理	弱い	強い
ノード間通信	弱い	強い (光リンクを利用)

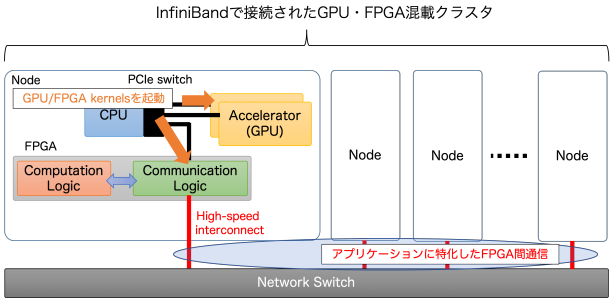

図 1　CHARM コンセプト

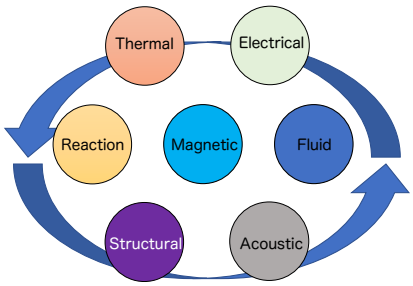

図 2　マルチフィジックスシミュレーション

可能である

- 近年のハイエンド FPGA では，高速な光リンクを用いた FPGA デバイス間の直接通信が可能である

しかし，FPGA には以下のような欠点がある．

- FPGA における最も代表的な開発言語である HDL（ハードウェア記述言語）は HPC アプリケーションユーザーにとってプログラミングが困難である
- 特にハイエンド FPGA においてはコンパイル時間が長い（数時間から 10 時間以上）ため，開発におけるターンアラウンドタイムが CPU や GPU に比べ非常に長い
- 通常の CPU や GPU と異なり，プログラムがハードウェア回路として展開されるため，ハードウェアリソース（論理素子やメモリデバイス）量による計算規模の制約が厳しい

しかし，Intel Stratix10[3] のような近年のハイエンド FPGA では，ハードウェアリソース（Logic Element）量やメモリ容量が格段に大きくなっている．また，高位合成技術の発達により，開発コストの問題も緩和されつつある．

我々は，この GPU と FPGA の相補的な特徴に着目している (表 1)．両デバイスを計算ノード上で結合し利用することにより，各デバイスが互いを補い合いながら，様々な HPC ワークロードに対応できると考えている．我々はこのコンセプトを CHARM（Cooperative Heterogeneous Acceleration with Reconfigurable Multidevices）と呼んでいる．その概念図を図 1 に示す．計算ノード内に GPU および FPGA が存在し，各ノードの FPGA 間は高性能なインターコネクトによって相互に接続されている．この環境において，演算量が多く単純な並列性が高い計算部分には GPU を利用し，ノード間通信が多発したり GPU による演算加速が効果的でない計算部分には FPGA を利用する．このコンセプトは，特にマルチフィジックス現象を題材とした複雑な実アプリケーションで有効であると考えている．図 2 に示すように先端的な計算科学応用では，解くべき問題が単純ではなく，複数の物理現象を組み合わせて問題を解く必要がある．そのため，問題を解く過程で多様な HPC ワークロードが出現し，単一の種類のアクセラレータでは高速化が困難になる．そこで CHARM コンセプトを適用し，各ワークロードに適切なアクセラレータを利用することにより，高いパフォーマンスを達成することが期待できる．

PCIe の汎用性により，CPU に対して GPU および FPGA を物理的に接続することは比較的容易であるが，それらの複合的プログラミングは HPC アプリケーションユーザーにとって非常に難易度が高い．我々は，OpenACC[4] を対象として，複数のバックエンドコンパイラ [5][6] を利用し，GPU および FPGA を統一して扱うことを可能にするプログラミング環境に関する研究を行っている [7][8]．また，最近では，Intel 社が提供する oneAPI[9] を用いて，複数のアクセラレータを単一のプログラミング言語 DPC++[10] で実装し，複数のデバイスに対する演算制御と，それらのデバイス間のデータ転送を行うというアプローチもある．

一方，GPU 向けには CUDA（C/C++），FPGA には OpenCL などを用いた既存のコード資産が多く存在する場合も多い．これらのコード資産を DPC++のような異なる言語で再実装することはアプリケーションユーザーにとって大きな手間となる．oneAPI には，DPC++の下で CUDA や OpenCL で記述されたコードを吸収し，一つのアプリケーションとして実装するというプログラム方法が提供されている．我々はこの機能に着目し，複数のデバイス用に書かれた従来のコードを再利用しつつ，複雑なコーディングとなり得る計算カーネル実行・イベントキューイング・データ移動・全体制御などを DPC++を用いて統一的に記述するという手法で oneAPI を CHARM に適用することを検討する．

本稿では，このような手法で oneAPI を CHARM のために用い，既に開発されている宇宙物理コードに適用し，実機上で性能評価を行う．

2. 関連研究

CHARM コンセプトでは，GPU と FPGA をアプリケーションの部分的特性に応じて使い分ける．ここでは，GPU・FPGA 協調計算をどのようにプログラミングするかが重要になる．GPU や FPGA など異なる種類のアクセラレータが協調動作するようなヘテロジニアスシステムにおけるプログラミングに関する先行研究を以下に紹介する．

[11] では，複数のアクセラレータを搭載できるノードからなるヘテロジニアスクラスタ Axel を提案し，N 体シミュレーションを対象として，FPGA・GPU・CPU が協調動作したと報告している．当時，CPU・GPU・FPGA を包括的に扱える商用およびオープンソースのフレームワークは存在しなかったため，著者らは MapReduce をベースに Axel クラスタ用のフレームワークを独自に開発した．このため，既存のアプリケーションを Axel クラスタ上で動作させるためには，MapReduce フレームワークに合わせてアプリケーションを再実装する必要がある．加えて，当時は高位合成技術が発達しておらず，FPGA の実装にハードウェア記述言語を利用する必要があった．そのため，FPGA 開発のコストが高いという課題があった．

近年，高位合成技術は飛躍的に進歩し，それらを活用したフレームワークも多く提案されている．[12] では，OpenCL をベースとした高水準フレームワークである EngineCL に対して，FPGA に対応するように拡張を行っている．単一のプログラミング言語で CPU・GPU・FPGA を扱うことが可能になり，このフレームワークを用いることで CPU・GPU・FPGA 協調計算を実現したと報告されている．

しかし，アクセラレータを利用する既存の HPC アプリケーションの多くは NVIDIA 社製の GPU を対象とした CUDA ベースの実装が行われており，その場合，EngineCL との互換性のためにすべてのコードを OpenCL に書き換える必要がある．これはアプリケーションユーザーにとって負担が大きい．

[13] では，GPU・FPGA 混載ノードを用いて初期宇宙における天体形成のシミュレーションを行う実アプリケーションである ARGOT の高速化をしている．このシミュレーションは，ARGOT 法と ART 法の 2 つの処理からなる．特に ART 法は，並列性に乏しく GPU による高速化に不向きであるという特徴をもつ．この処理を FPGA によるパイプライン並列を用いて高速化することで，計算全体の高速化を行っている．この結果，GPU のみの実装と比較して，最大 17.4 倍の高速化を達成した．この研究では CUDA と OpenCL という異なる言語を用いた複合的プログラミングが行われている．しかし，GPU・FPGA 協調計算を複合的プログラミングによって記述することは，アプリケーションユーザーにとって難易度が高い．

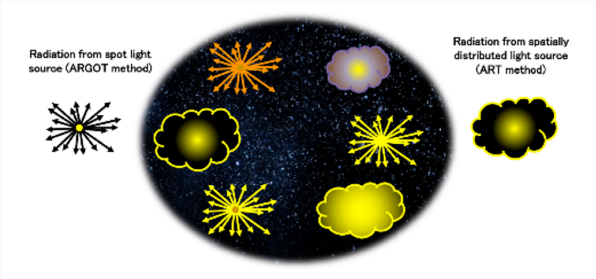

図 3 ARGOT プログラム概要

一方で，oneAPI は複数のデバイスを統一的に扱えるようにするプログラミング言語 DPC++を提供している．この DPC++プログラムは，既存の OpenCL や CUDA コードを直接利用することが可能であり，アプリケーションユーザーの負担を軽減することができる．また，GPU・FPGA 協調計算を統一的な記述で実装することができるため，ソースコードの保守性を高めることができる．したがって，本稿では oneAPI を CHARM コンセプト実現に向けて，積極的に活用する．

3. 宇宙輻射輸送コード: ARGOT

ARGOT (Accelerated Radiative transfer on Grids using Oct-Tree) は，宇宙輻射輸送を解くアプリケーションプログラムであり，筑波大学計算科学研究センターによって開発されている．宇宙輻射輸送問題とは，宇宙初期の天体形成を解明する上で重要な研究対象であり，星・ブラックホール・銀河などの形成過程を解明する手がかりとなる．

ARGOT コードは，ARGOT 法 [14] と ART 法 [15] の 2 つの部分計算を行うことで，宇宙輻射輸送問題を解く．ここで，ARGOT コードはプログラムコード全体を指し，ARGOT 法及び ART 法はシミュレーションの中心となる 2 種類の異なる輻射輸送問題を解くアルゴリズムである．ARGOT コードと ARGOT 法を混同しないよう注意されたい．前者では，点光源からの輻射輸送を計算し，後者では空間に広がる光源からの輻射輸送を計算する．次の節では，2 つの部分計算について説明する．

3.1 ARGOT 法

ARGOT 法は，点光源からの輻射輸送を計算する部分計算である．ARGOT 法は点光源の数に比例して計算量が増加する．計算量を抑えるために，AROGT 法では光源の分布を表すデータ構造として八分木を利用している．八分木の利用により，遠方に位置するツリーノード内における全ての点光源を単一の光源とみなすことができる．その結果，計算に使用する点光源の数を N とすると，計算オーダーを $O(N^2)$ から $O(NlogN)$ に減少させ，全体の計算量を抑える事ができる．

あるメッシュグリッドを対象とした，輻射輸送によって

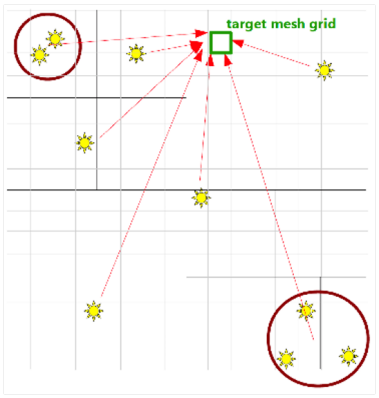

図 4 ARGOT 法

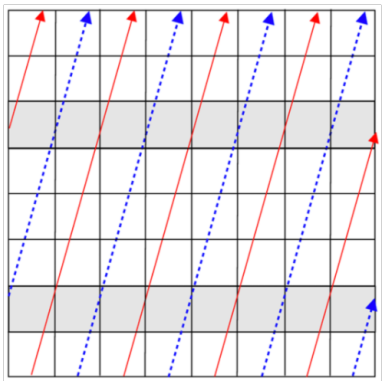

図 5 ART 法

各点光源から得られる光子束は以下の式により求まる．

$$f(\nu) = \frac{L(\nu)e^{-\tau(\nu)}}{4\pi r^2} \tag{1}$$

$L(\nu), \tau(\nu)$ はそれぞれ，振動数 ν における光の速度・光学的距離である．また，$\tau(\nu)$ は以下の式で求められる．

$$\tau(\nu) = \sigma(\nu) \int n(\boldsymbol{x}) dl \simeq \sigma(\nu) \sum_i n\left(\boldsymbol{x}_i\right) \Delta l \tag{2}$$

ここで，$n(x)$ は，光を吸収するガス分子の数密度である．

3.2 ART 法

ART 法は，空間に広がる光源からの輻射輸送を計算する部分計算である．与えられた問題空間を 3 次元メッシュに分割し，レイトレーシングを行うことで輻射輸送の計算を行う．たたし，それぞれのレイは境界から発射され，平行に直進し，反射や屈折はしない．

ART 法において，レイがメッシュを通過するたびに計算する式は以下の通りである．

$$I_\nu^{out}(\hat{\boldsymbol{n}}) = I_\nu^{in}(\hat{\boldsymbol{n}}) e^{-\Delta\tau_\nu} + S_p\left(1 - e^{-\Delta\tau_\nu}\right) \tag{3}$$

ここで，$\nu, I_\nu^{in}, I_\nu^{\text{out}}, \hat{n}, \Delta\tau, S_p$ はそれぞれ，周波数・入力放射・出力放射強度・レイの方向・メッシュにおける光学的厚み・メッシュの source function を表す．

GPU に代表される SIMD アーキテクチャ上で ART 法を実装するとき，高速化の障害となり得る 2 つの問題がある．1 つ目は，レイの方向に応じて読み込むメッシュデータが非連続となり得るという問題である．メモリアクセスパターンがランダムアクセスとなり，キャッシュヒット率の低下が発生する可能性がある．特に GPU ではコアレッシングアクセスができず，大幅な性能低下の恐れがある．2 つ目は，メッシュにおける積分計算が衝突する恐れがあるという問題である．同じメッシュ内を複数のレイが通過した場合，そのメッシュに対して複数のレイの効果を重ね合わせる必要がある．この場合の計算を正しく行うために，メッシュ内のレイの計算を逐次的に行うか，atomic 演算などの排他処理を用いて並列に処理する必要がある．排他処理には同期が必要なため，一般に性能低下を引き起こす可能性がある．また，ART 法は各空間セルにおける計算の並列性が低く，計算中に含まれる演算数が不足しているため，SIMD アーキテクチャの性能を十分に引き出すことができない．以上より，ART 法においては GPU による実装では性能が十分に引き出せない場合が多い．

そこで，FPGA による ART 法の実装が検討された [16]．FPGA ではオンチップの BRAM が利用可能であり，ランダムアクセスを低レイテンシかつ高バンド幅で行うことができる．また，ART 法専用の回路を FPGA 上で実装することで SIMD アーキテクチャにおける並列処理の制約を回避することができる．すなわち，SIMD アーキテクチャで課題であった 2 つの問題を FPGA により回避することが期待できる．以上より，ART 法は FPGA による実装が適している．

3.3 GPU・FPGA 加速 ARGOT コード

GPU・FPGA 加速 ARGOT コード実装の概要を図 6 に示す．前節で述べたとおり，ARGOT 法は GPU による高速化が効果的であるため GPU を用いる．一方で，ART 法は GPU が不得意とする計算であるため，FPGA を用いて高速化を行う．ART 法では，GPU 上でデータを生成し FPGA に送信，FPGA 上で ART 法による計算を実施し，計算結果を GPU に書き戻すという流れで処理が行われる．そのため，GPU-FPGA 間で適切なデータ転送が必要になる．つまり，GPU・FPGA の連携処理の実装が必要になる．[13] における GPU・FPGA 加速 ARGOT コード実装では，2 つの言語，CUDA および OpenCL を用いて実装されている．しかし，このような実装では，各プログラミング言語 API を同時に利用するため，連携処理のソースコードが煩雑となり保守性を損なう可能性が高い．そこで，本研究では GPU・FPGA 連携処理の実装において oneAPI を用いることで統一的な記述による実装を行う．本稿では oneAPI を用いた GPU・FPGA 加速 ARGOT の実装と性能評価について報告する．この手法は，HPC アプリケー

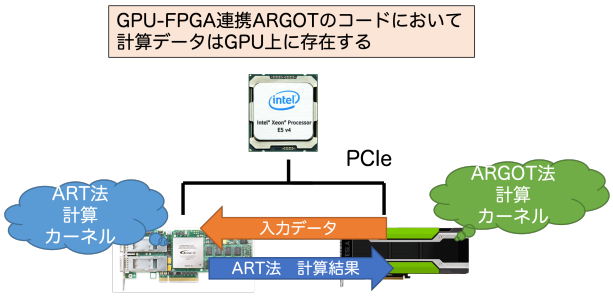

図 6 GPU・FPGA 加速 ARGOT コード実装

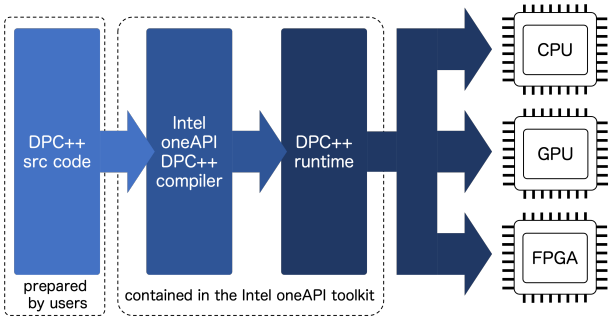

図 7 oneAPI 概要図

ションユーザーの生産性を向上させる GPU-FPGA 協調計算を実現法の一つとなり得る．

4. oneAPI を用いた GPU・FPGA 加速 ARGOT コード

4.1 oneAPI 概要

oneAPI[9] は，Intel 社により提唱されたクロスアーキテクチャプログラミングフレームワークである．oneAPI の目的は，異なるアーキテクチャ間における開発を簡素化することにある．oneAPI には，DPC++プログラミング言語とライブラリ群が含まれており，単一の言語や API での開発が可能になる (図 7)．

現在，oneAPI が公式にサポートしているデバイスは [19] の通りである．また，現時点において Experimental Support であるが，CUDA デバイスも利用可能 [20] である．

4.2 DPC++概要

DPC++ (Data Parallel C++) は，oneAPI の中核をなすプログラミング言語であり，異種アーキテクチャ間において統一の言語を用いた開発を可能にする．DPC++は，C++および Khronos Group により策定された SYCL 仕様 [21] を基本としている．

DPC++では，Queue をインターフェースとして，アクセラレータデバイス（以下，単にデバイスと呼ぶ）への操作を行う．この操作のことをアクションとよび，代表的なものとしてカーネルの起動 (`parallel_for/single_task`) や明示的なデータ転送 (`copy/memcpy`) などがある．Queue

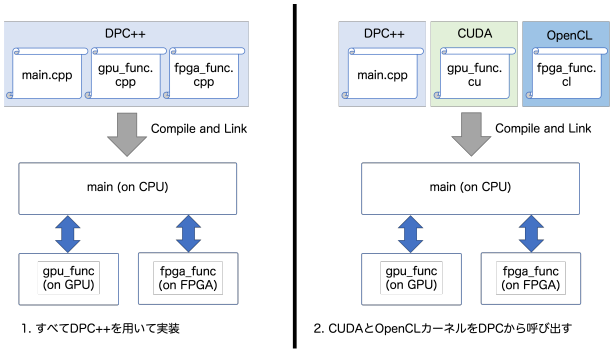

図 8 2 通りの実装方法

は作成時に必ず 1 つのデバイスと紐付けられ，その Queue に投入したアクションは紐付けられたデバイス上で実行される．つまり，GPU と FPGA を同時に扱いたい場合は，GPU 用の Queue と FPGA 用の Queue を用意し，それぞれの Queue にアクションを投入すればよい．投入先の Queue を変えることにより，実行デバイスを切り替える事ができる．

現在，DPC++のコンパイラには 2 つの種類が存在する．oneAPI Base Toolkit に付属する公式リリース版 [22] とオープンソースで公開されている OSS 版 [23] である．NVIDIA GPU に対応しているのは OSS 版のみであるため，本稿では OSS 版 DPC++コンパイラを利用している．

4.3 oneAPI を用いた CHARM 実装

oneAPI を利用して，CHARM コンセプトを実現する方法として，以下の 2 通りのプログラミング方法がある．

(1) 各デバイス側のコードをすべて DPC++により記述する方法
(2) 既存の CUDA カーネルや OpenCL カーネルを DPC++プログラミングフレームワークから呼び出す方法

前者は Intel 社が推奨している方法であり，単一のコードで CHARM を実現することができる．我々は，この方法で NVIDIA GPU・Intel FPGA を連携させることが可能であることを確認している [17]．

しかし，この方法で既存のアプリケーションを実行するためには，ソースコード全体を DPC++で再実装する必要がある．アクセラレータを利用する既存の HPC アプリケーションの多くは，NVIDIA GPU での実行を想定しており，CUDA により記述されている．このような異なる言語への再実装は，アプリケーションユーザーには大きな負担となり得る．本稿の対象である実アプリケーションも例外でなく，CUDA および OpenCL で実装されている．これらのソースコードを全て DPC++に置き換えるには多くの工数を要する．

一方で，DPC++には，CUDA および OpenCL で記述されたカーネルを呼び出す機能が存在する．これらの機能を

```
cuda_mem[idev].segment_dev = sycl::malloc_device<struct
ray_segment>(nseg_to_dev[idev], sycl_gpu->strm_queue[idev]);
...
sycl_gpu->strm_queue[idev].submit([&](sycl::handler& h) {
  h.memcpy(cuda_mem[idev].segment_dev, seg+offset_seg[idev],
sizeof(struct ray_segment)*nseg_to_dev[idev]);
});
sycl_gpu->strm_queue[idev].wait();          memcpy (CPU -> GPU)
...
while(nseg_waiting > 0) {
  ...
  dim3 nthrd(nthread, 1, 1);
  dim3 nblck(nblock, 1, 1);

  sycl_gpu->strm_queue[idev].submit([&](sycl::handler& h) {
  h.interop_task([=](sycl::interop_handler ih) {
    // Call the CUDA kernel directly from SYCL
    calc_optical_depth_kernel<<<nblck, nthrd, 0>>>
      (cuda_mem[idev].mesh_dev, cuda_mem[idev].segment_dev,
                cuda_mem[idev].this_run_dev, offset);
  });
  });
  sycl_gpu->strm_queue[idev].wait();        CUDA kernel launched
  ...
}
```

図 9 DPC++における CUDA カーネルの呼び出し

利用することにより，後者の方法，すなわち，DPC++によって協調動作をする CUDA および OpenCL カーネルの実現ができる．そこで，本稿では後者の方法を採用する．既存の CUDA や OpenCL のコードをそのまま利用することで，oneAPI による CHARM 実装に要する工数を可能な限り削減することが期待できる．

4.4 DCP++を用いた ARGOT 法の呼び出し

ARGOT 法は，CUDA カーネルによって実装され，GPU を利用して計算が行われる．本稿における oneAPI による CHARM 実装では，DPC++からこれらの CUDA カーネルを呼び出すことで，ARGOT 法の計算を行う．DPC++から CUDA カーネルを呼び出すには，NVIDIA GPU をサポートする `oneAPI for CUDA`[20] を利用する．実際に ARGOT 法の実装の中で，CUDA カーネルの呼び出し部分コードの例を図 9 に示す．

これは，ARGOT 法の式 (1) を計算するためのコードの一部である． まず，`oneAPI for CUDA` を利用するためには，`<CL/sycl/backend/cuda.hpp>`というヘッダーファイルをインクルードする必要がある．このヘッダーファイルを読み込むことで，oneAPI から CUDA API と同等の操作を行うことが可能になる．図 9 では，上段のオレンジ色の線が，CPU と GPU のデータ転送に相当する．下部のオレンジ色の線が，CUDA カーネルの呼び出しである．CUDA カーネルを呼び出すために，interop_task という関数を利用している．この関数を使うことで，既存の CUDA カーネルを直接呼び出すことができる．

4.5 DCP++を用いた ART 法の呼び出し

図 10 は，OpenCL による ART 法の実装を図示したものである．図 10 に示すように，FPGA に実装された ART 法のハードウェアアクセラレータは，ART 法の計算を実行するエンジンと，外部メモリからエンジンへのメッシュ

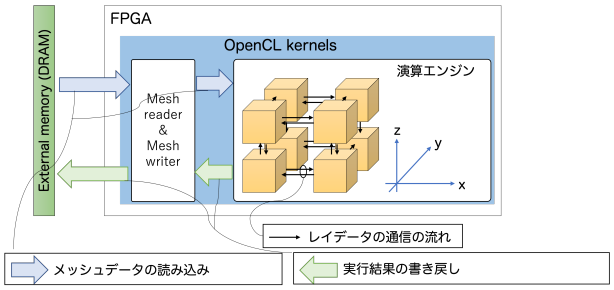

図 10 OpenCL による ART 実装

データの読み込みと外部メモリへの計算結果の書き込みを担当するハンドラからなり，これらはすべて OpenCL で実装されている．

演算エンジンは，演算コアを 3 次元 ($2 \times 2 \times 2$) 状に接続した PE（Processing Elements）で構成されている．つまり，1 つの FPGA が担当する問題空間をより小さなブロックに分割し，各 PE に割り当てて並列計算を行う．分割された問題空間は，各 PE のブロック BRAM（Block RAM）を用いて実装されたスクラッチパッドメモリに格納され，各 PE は 16^3 個の問題サイズを保持する．FPGA 全体には 32^3 個の問題サイズが保持される．そして，各 PE は，レイデータを他の PE へ通信する．そして，レイデータを受け取った PE は，ART 法の演算カーネルを実行し，演算カーネルの結果を反映したレイデータをレイ方向に位置する次の問題空間を担当する PE に送ることで，ART 法のレイトレーシングを計算している．そして，ART 法の実行結果は，ハンドラを介して外部メモリに書き戻される．

DPC++プログラムから，この OpenCL カーネルを呼び出すことで，ART 法の計算を行う．図 11 は，この ART 法の OpenCL 実装を oneAPI で利用するために使用した DPC++の部分コードである．DPC++は，`<CL/sycl.hpp>`と`<CL/opencl.h>`のヘッダを記述することで，OpenCL のオブジェクトを DPC++のオブジェクトとして扱えるようになる．そのための操作，つまり，OpenCL オブジェクト（`cl_context`，`cl_kernel`，`cl_queue`）から DPC++オブジェクトへの変換は，12〜27 行目に対応している．

初期データであるメッシュデータをホスト CPU から FPGA に送るために，30 行目で FPGA のデバイスメモリ領域を確保し，31〜33 行目で CPU から FPGA へのメモリコピーを実行する．その後，35〜57 行目の操作ですべての OpenCL カーネルを起動して，FPGA ベースの ART 法を実行する．ART 法が終了すると，67〜69 行目の操作で，FPGA の外部メモリから CPU のメモリに実行結果が書き戻される．

4.6 コンパイルフロー

図 12 に oneAPI による GPU・FPGA 加速 ARGOT コードのコンパイルフローを示す．ARGOT 法のソースコード

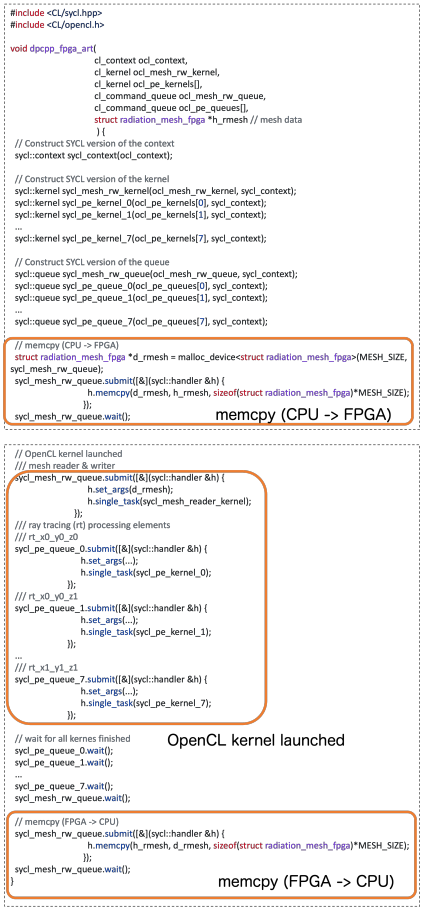

図 11 DPC++における OpenCL カーネルの呼び出し

と ART 法のソースコードを別々にコンパイルした後，リンクしている．このコンパイラフローで使用した clang++ コンパイラは，oneAPI DPC++コンパイラの OSS 版で，DPC++や OpenCL のソースコードだけでなく，CUDA コードも入力可能である．本稿では，CUDA カーネルおよび OpenCL カーネルのソースコードをそのまま使用し，カーネルを呼び出す部分 (図 12 の赤色) に DPC++のコードを追加した．このコンパイルフローに従ってコンパイルし，最終的に GPU と FPGA の両方を制御できるホストバイナリが生成される．なお，FPGA にロードするための aocx は，Intel FPGA SDK for the OpenCL[18] を用いたオフライン・コンパイルにより生成した．

5. 性能評価

5.1 評価環境

本稿では，筑波大学計算科学研究センターで運用している CHARM コンセプト実験クラスタ Pre-PACS-X (PPX) を評価環境に用いた．評価環境の構成を表 2 に示す．この評価環境は，3 種類のデバイスからなるヘテロジニアス環境であり，2 台の Intel Xeon Gold 6242 CPU，1 台の NVIDIA V100 GPU(Gen3 ×16)，および 1 台の Intel FPGA PAC D5005 ボードから構成される．各アクセラレータは PCI Gen3 ×16 を介して，CPU に接続されている．

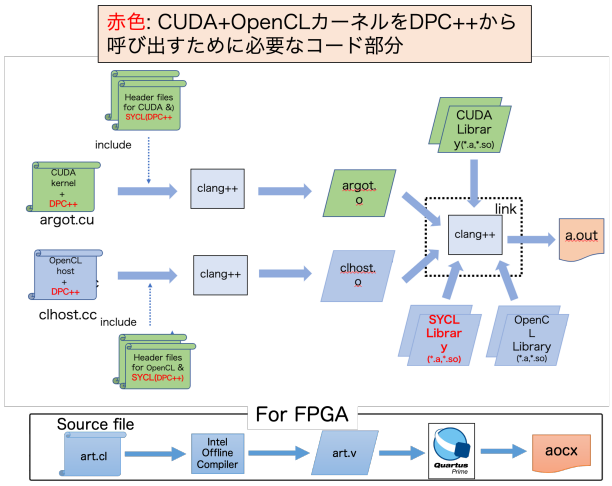

図 12 コンパイルフロー

表 2 評価環境

ハードウェア構成	
CPU	Intel® Xeon® Gold 6242 × 2
GPU	NVIDIA Tesla V100 (PCIe Gen3 × 16 card version)
FPGA	Intel FPGA PAC D5005 (Intel Stratix 10 SX)
ソフトウェア構成	
Host OS	CentOS 7.9
Linux Kernel Version	3.10.0-1160.15.2.el7.x86_64
Compiler	oneAPI data parallel C++ compiler [23] (commitID: 6e9ddb6)
MPI	Open MPI 4.0.3
Accelerator Platforms	CUDA Toolkit 10.2.89 Intel FPGA SDK for OpenCL 2021.2.0 Build 268.1 Pro edition

評価環境の OS には CentOS 7.9 を用い，oneAPI を用いた ARGOT コードは oneAPI Data Parallel C++コンパイラを用いてコンパイルした．バックエンドには CUDA Toolkit 10.2.89 を使用した．ART 法の FPGA 実装は OpenCL を用いて実装され，Intel FPGA SDK for OpenCL, version 2021.2.0 Build 268.1 Pro edition が提供するオフラインコンパイラを使用してコンパイルした．また，今回は単一ノードのみの評価であるが，ARGOT コードは MPI を使用して複数ノードで動作することを前提に開発してあるため，本稿における評価では OpenMPI 4.0.3 を使用し，単一プロセスで動作するようにした．

5.2 ARGOT コードの性能評価

図 13 は，問題サイズ 32^3 を適用した場合の CPU，GPU，FPGA を利用した各実装における性能評価である．縦軸は，各シミュレーションステップの実行時間を示している．

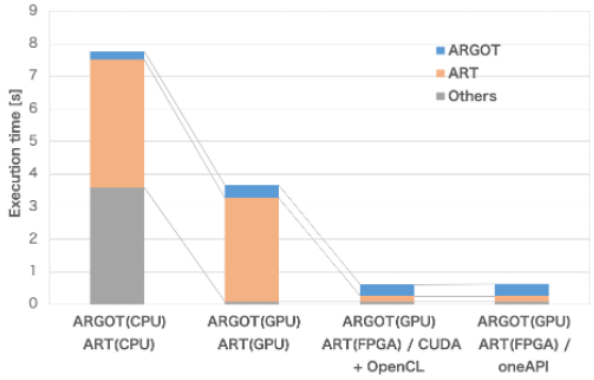

図 13 問題サイズ 32^3 における CPU，GPU，FPGA を利用した各実装における ARGOT コード性能評価

表 3 問題サイズ 32^3 における CUDA+OpenCL 実装および oneAPI 実装による ARGOT コード性能比較

実装	ARGOT [s]	ART [s]	Others [s]
CUDA + OpenCL	約 0.362	約 0.170	約 0.0782
oneAPI	約 0.375	約 0.170	約 0.0734

各項目の意味は以下の通りである．

- ARGOT(CPU) ART(CPU)：ARGOT 法および ART 法が共に CPU 実装である．この CPU 実装では，1 つの Xeon CPU 上で 16 スレッド実行を行っている．
- ARGOT(GPU) ART(GPU)：ARGOT 法および ART 法が共に GPU 実装である．この GPU 実装では，CUDA で記述され，NVIDIA V100 上で実行される．
- ARGOT(GPU) ART(FPGA)/CUDA+OpenCL：ARGOT 法が GPU 実装であり，ART 法が FPGA 実装である．ベースラインとなる CHARM コンセプト実装である．
- ARGOT(GPU) ART(FPGA)/oneAPI：ARGOT 法が GPU 実装であり，ART 法が FPGA 実装である．oneAPI を用いた CHARM コンセプト実装である．

また，CUDA+OpenCL 実装および oneAPI 実装における各シミュレーションステップの実行時間を表 3 に示す．

図 13 から，両方共に CPU 実装の場合において ART 法だけでなく，Others も支配的であることがわかる．Others は主に，ARGOT 法と ART 法の実行結果をもとに，各メッシュにおける化学反応などの計算を行っている．各メッシュの計算は独立に行うことができるため，GPU による並列化が効果的である．実際に，GPU を利用した実装では，全体の計算時間に占める Others の割合が非常に小さくなっている．

Others は GPU による高速化が効果的に働いたが，ART 法の場合は GPU による高速化が十分に働かない．図 13 における ART 法の CPU 実装と ART 法の GPU 実装を比較すると，約 1.2 倍の高速化にとどまっている．これは，3.2 節で述べたとおり，ART 法は，GPU に代表される SIMD

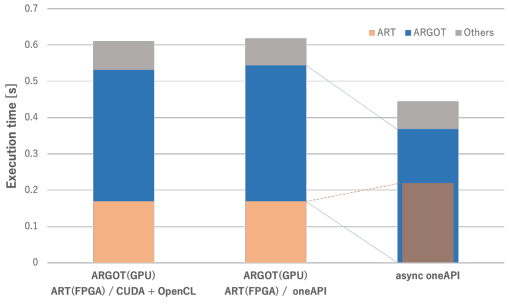

図 14 問題サイズ 32^3 における ARGOT コード非同期実行版の性能評価

アーキテクチャとの相性が合わないためである．また，問題サイズが小さく，GPU 内部にある CUDA コアを十分に活用できるだけの並列性が得られないことも原因の一つだと考えられる．一方で，図 13 から FPGA を利用した ART 法の実装は，GPU 実装版よりも性能が大幅に優れていることがわかる．FPGA 内におけるパイプライン化された ART 法の実装により，ART 法の計算が高速にできたためと考えられる．また，パイプライン並列化により，問題サイズが小さい場合でも計算を高速化できることがわかる．今回の評価における問題サイズが 32^3 の場合，FPGA 実装版 ART は，GPU 実装版 ART と比較して 18.7 倍の性能向上を達成した．

既存の CHARM 実装，すなわち CUDA+OpenCL による実装と oneAPI 実装を比較したところ，oneAPI 実装の実行時間が 1.5%だけ増加した．つまり，DPC++によって協調動作をする CUDA および OpenCL カーネルを呼び出すという実装でも，オリジナル実装と同程度の性能を実現できることが確認できた．

ARGOT コードにおける ARGOT 法と ART 法は非同期に実行することができる．また，oneAPI の各 Queue は非同期で実行される．したがって，GPU および FPGA の各 Queue の操作を OpenMP により並列実行することで，各処理を非同期に実行させる事ができる．問題サイズ 32^3 を適用した場合の，非同期実行の性能評価が図 14 である．縦軸は，各シミュレーションステップの実行時間を示している．図 14 における，左 2 本のグラフは図 13 と同等であり，右端のグラフが新たに追加した非同期実行版である．ART 法の実行時間が ARGOT 法の実行時間で隠蔽されており，全体として GPU 実行（ARGOT 法）と FPGA 実行（ART 法）を逐次的に処理する場合に比べ 1.38 倍の高速化を実現した．

6. まとめ

本稿では，oneAPI を用いた GPU・FPGA 加速 ARGOT の実装と性能評価について報告した．アプリケーションユーザーのプログラミング工数を可能な限り削減するた

め，既存の CUDA および OpenCL コードをそのまま利用し，DPC++を用いて協調動作させるアプローチに着目した．oneAPI で実装した GPU・FPGA 加速 ARGOT コードの性能を評価したところ，CUDA+OpenCL で実装した ARGOT コードと同程度の性能が得られることがわかった．つまり，既存のアプリケーションコードを oneAPI により，性能低下をほぼ気にすることなく利用することができる．また，GPU と FPGA を非同期に実行させることで，さらなる性能向上ができることがわかった．

以上より，oneAPI の下で GPU と FPGA を協調動作させることで，各アクセラレータのカーネル起動やアクセラレータ間の同期・仲介を統一的な API (Queue) で行うことができ，アプリケーションコードの高い保守性を維持することが可能である．既存の HPC コードに対して，GPU と FPGA による高速化を目指す CHARM コンセプトを実現する一つの手段として，本稿のアプローチは有用である．

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」による．本研究の一部は，JSPS 科研費 21H04869 の助成を受けたものである．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．

参考文献

[1] November 2021 — TOP500 Supercomputer Sites https://www.top500.org/lists/top500/2021/11/
[2] NVIDIA A100 GPU. https://www.nvidia.com/en-us/data-center/a100/
[3] Intel Stratix 10 FPGA. https://www.intel.com/content/www/us/en/products/details/fpga/stratix/10.html
[4] OpenACC. https://www.openacc.org/
[5] Omni Compiler Project. https://omni-compiler.org/index.html
[6] S. Lee, J. Kim and J. S. Vetter, "OpenACC to FPGA: A Framework for Directive-Based High-Performance Reconfigurable Computing," 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2016, pp. 544-554, doi: 10.1109/IPDPS.2016.28.
[7] Ryuta Tsunashima, Ryohei Kobayashi, Norihisa Fujita, Taisuke Boku, Seyong Lee, Jeffrey S. Vetter, Hitoshi Murai, Masahiro Nakao, Mitsuhisa Sato: OpenACC unified programming environment for GPU and FPGA multi-hybrid acceleration, HLPP 2020 Proceedings, HLPP 2020, (2020)
[8] 綱島隆太，小林諒平，藤田典久，中道安祐未，朴泰祐，Lee Seyong，Vetter Jeffrey，村井均，佐藤三久：GPU-FPGA 協調プログラミングを実現するコンパイラの開発，情報処理学会研究報告，2019-HPC-172.
[9] oneAPI. https://www.oneapi.com/
[10] Data Parallel C++ (DPC++). https://www.intel.com/content/www/us/en/develop/documentation/oneapi-programming-guide/top/oneapi-programming-model/data-parallel-c-dpc.html
[11] Kuen Hung Tsoi and Wayne Luk. 2010. Axel: A Heterogeneous Cluster with FPGAs and GPUs. In Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays (Monterey, California, USA) (FPGA '10). Association for Computing Machinery, New York, NY, USA, 115–124. https://doi.org/10.1145/1723112.1723134
[12] María Angélica Dávila Guzmán, Raúl Nozal, Rubén Gran Tejero, María Villarroya-Gaudó, Darío Suárez Gracia, and Jose Luis Bosque. 2019. Cooperative CPU, GPU, and FPGA Heterogeneous Execution with EngineCL. J. Supercomput. 75, 3 (March 2019), 1732–1746. https://doi.org/10.1007/s11227-019-02768-y
[13] Kobayashi Ryohei, Fujita Norihisa, Yamaguchi Yoshiki, Boku Taisuke, Yoshikawa, K., Abe, M. and Umemura, M.: Accelerating Radiative Transfer Simulation with GPU-FPGA Cooperative Computation, 2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP), pp. 9–16 (online), DOI: 10.1109/ASAP49362.2020.00011 (2020)
[14] Takashi Okamoto, Kohji Yoshikawa, Masayuki Umemura, ARGOT: accelerated radiative transfer on grids using oct-tree, Monthly Notices of the Royal Astronomical Society, Volume 419, Issue 4, February 2012, Pages 2855–2866, https://doi.org/10.1111/j.1365-2966.2011.19927.x
[15] Satoshi Tanaka, Kohji Yoshikawa, Takashi Okamoto, Kenji Hasegawa, A new ray-tracing scheme for 3D diffuse radiation transfer on highly parallel architectures, Publications of the Astronomical Society of Japan, Volume 67, Issue 4, August 2015, 62, https://doi.org/10.1093/pasj/psv027
[16] Norihisa Fujita, Ryohei Kobayashi, Yoshiki Yamaguchi, Yuma Oobata, Taisuke Boku, Makito Abe, Kohji Yoshikawa, and Masayuki Umemura. 2018. Accelerating Space Radiative Transfer on FPGA using OpenCL. In Proceedings of the 9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART 2018). Association for Computing Machinery, New York, NY, USA, Article 6, 1–7. DOI:https://doi.org/10.1145/3241793.3241799
[17] 柏野 隆太, 小林 諒平, 藤田 典久, 朴 泰祐：oneAPI を用いた GPU・FPGA 混載ノードにおけるヘテロ演算加速プログラム開発, 情報処理学会研究報告, 2021-HPC-180.
[18] Overview: Intel FPGA SDK for OpenCL. https://www.intel.com/content/www/us/en/software/programmable/sdk-for-opencl/overview.html
[19] Intel oneAPI Base Toolkit System Requirements, https://software.intel.com/content/www/us/en/develop/articles/intel-oneapi-base-toolkit-system-requirements.html
[20] oneAPI for CUDA, https://codeplay.com/solutions/oneapi/for-cuda/
[21] SYCL Specification. https://www.khronos.org/sycl/
[22] oneAPI Data Parallel C++ compiler (公式リリース版), https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/dpc-compiler.html
[23] oneAPI Data Parallel C++ compiler (OSS 版), https://github.com/intel/llvm