Vol.2022-HPC-183 No.11
2022/3/17

OpenACCによる宇宙物理シミュレーションのGPU＋FPGA協調計算の実装

綱島隆太[1,4,a)]　小林諒平[2,1]　藤田典久[2,1]　朴泰祐[2,1]　Lee, Seyong[3]　Vetter, Jeffrey S.[3]　村井均[4]
中尾昌広[4]　辻美和子[4]　佐藤三久[4,1]

概要：近年 HPC 分野では，アクセラレータとして GPU や FPGA が注目されている．特に FPGA は GPU の苦手な処理でも性能を発揮することが期待されており，我々は両者を統合した次世代スーパーコンピュータの研究を行っている．しかし，GPU と FPGA を組み合わせたプログラミングでは，標準的な手法や言語が存在していない．HPC における GPU のシェアは現状では NVIDIA 社によるものが支配的であるため，主に GPU の処理は CUDA で記述されている．一方で，FPGA では高位合成技術により，ハードウェア記述言語に代わって，OpenCL の使用が可能になっている．これら二つを組み合わせてプログラミングを行うことはアプリケーションプログラマーにとって多大な負担となる．また，OpenCL では GPU のプログラミングも行うことができるが，既存のアプリケーションの多くはすでに CUDA で書かれているか，CPU 版しか存在しないため，OpenCL に書き直すには相当の負担が掛かる．仮にコードを別の言語で書き直すのであれば，より汎用性や抽象度の高い記述を行うことが理想である．そこで，我々は CAMP（Cooperative Acceleration by Multi-device Programming）というコンセプトの下，ディレクティブ形式の API である OpenACC を用いて両アクセラレータのプログラミングを統一的に行う環境である MHOAT (Multi-Hybrid OpenACC Translator) を開発している．本稿では，宇宙物理分野の実アプリケーションである ARGOT コードを対象に，MHOAT による GPU と FPGA の混合演算加速の実装について述べる．

1. 近年の HPC 分野の動向

近年，HPC 分野ではアクセラレータの利用が広まっている．アクセラレータとは，演算加速を目的として，PCI Express（PCIe）などで CPU の周辺機器として外付けして利用されるハードウェアデバイスである．世界のスーパーコンピューターの性能ランキング TOP500 の最新版（2021 年 11 月）[1] では，上位 10 機中 8 機がアクセラレータを搭載している．特に，GPU（Graphics Processing Unit）が最も利用されているアクセラレータであり，8 機中 7 機のアクセラレータが GPU である．

本来 GPU はその名の通り画像処理に利用されることを目的としたプロセッサであったが，CPU に対し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，GPGPU（General Purpose GPU）と呼ばれる汎用演算への利用が盛んに行われている．しかし，GPU ではアプリケーションに以下の処理が含まれている場合，当該箇所において理論最高性能に対して高性能化が難しい．

- GPU の演算コア数に対してデータ並列度が低い場合
- 条件分岐が頻出する場合（Warp Divergence が発生するため）
- ノード間通信が頻発する場合（CPU での処理のために GPU カーネルが中断されるため）

一方で，現在では FPGA（Field Programmable Gate Array）もアクセラレータとして注目されている．FPGA は内部の論理回路を再構成可能で，何度でも回路を書き換えられるハードウェアデバイスである．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特化した演算加速回路を設計可能
- パイプライン処理によって SIMD 並列に囚われない高速化が可能
- ハイエンド FPGA では光リンクでの直接接続による高速なノード間の FPGA 通信が可能
- （実装依存ではあるが）GPU より省電力化が可能

欠点として，Verilog HDL 等のハードウェア記述言語でのコーディングが一般ユーザにとって困難であること，またコンパイル時間が極めて長い（数時間から数日かかる場合

1 筑波大学 理工情報生命学術院 システム情報工学研究群 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
3 Oak Ridge National Laboratory
4 理化学研究所 計算科学研究センター
a) tsunashima@hpcs.cs.tsukuba.ac.jp

ⓒ 2022 Information Processing Society of Japan

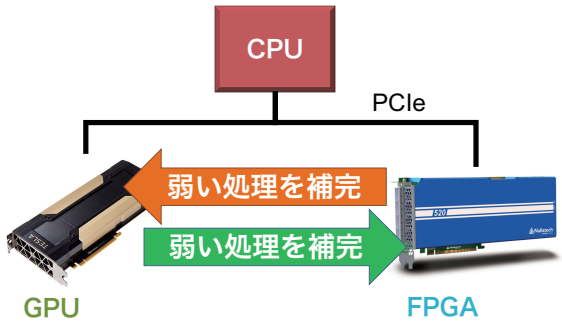

図 1　CHARM コンセプトの概略図

すらある）ことが挙げられる．さらに，アプリケーションに利用できるハードウェアリソースが限られることが HPC への適用上の大きな問題である．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，処理内容を動的に制御できる実行モデルではなく，計算に必要な全回路が展開され，大きな論理回路空間を必要とすることによる．しかし，回路規模の大きなハイエンド FPGA が登場してきているため，以前と比較して HPC 向けの大規模かつ複雑なアプリケーションの実装も可能となりつつある．

このように GPU と FPGA は異なる性質を持つアクセラレータである．我々は両アクセラレータの相互補完による演算加速が可能であると考え，このコンセプトを CHARM（Cooperative Heterogeneous Acceleration with Reconfigurable Multidevices）と呼んでいる（**図 1**）．CHARM コンセプトでは，GPU は単純なデータ並列性の高い処理に用い，FPGA では GPU では高速化できない処理を行うことで，相補的に演算加速を行う．また，これをノード並列に行うことで，大きな演算加速性能を得られるように，ポストムーア時代の高性能計算を可能とすることを目指している．

2.　CHARM におけるプログラミング環境

2.1　課題

CHARM では一つのプログラムの中で GPU と FPGA を同時に使うことで協調計算を行うことを想定している．しかし，このようなプログラムを記述するためのプログラミング環境の整備はあまり進んでいない．OpenCL はヘテロジニアスコンピューティングに対応した言語であり，GPU と FPGA を両方使ったプログラミングが可能であるが，CPU プログラミングと比べて非常に複雑であり，CUDA よりも多くのことを考慮して記述する必要がある．CUDA は NVIDIA 社が提供している GPU プログラミング環境であり，他社の GPU や，FPGA のような他のデバイス用に記述することはできない．現状，多くの実アプリケーションは CPU コードしか存在しないか，GPU 版が存在する場合は殆どが CUDA で書かれている．したがって，OpenCL を利用するためには書き換えの手間が発生し，アプリケーションを開発しているプログラマーに多大な負担を負わせることになる．さらに、科学計算のアプリケーションをプログラミングするのは計算機科学の素養を持たない物理、化学、数学などの科学者であり、CUDA や OpenCL のようにメモリ階層やスレッド数など低レイヤーの知識が要求されるプログラミングは、彼らにとって非常にハードルが高い。FPGA は元々ハードウェア記述言語（HDL）によってプログラミングされており，高度なハードウェアの知識を必要としていたが，近年になり高位合成（HLS）技術が発達したことで，OpenCL や C++を始めとした高位言語で記述できるようになってきている．以前，我々の研究では，GPU+FPGA にそれぞれ CUDA+OpenCL を用いたプログラミングによる宇宙物理シミュレーション ARGOT の実装を行ったが [2]，GPU 側を CUDA のままにしても，FPGA 向けの OpenCL への書き換えの手間は非常に大きく，さらに 2 つのプログラミング言語を混合したプログラミングは非常に複雑で，コードの保守性も悪いものであった．

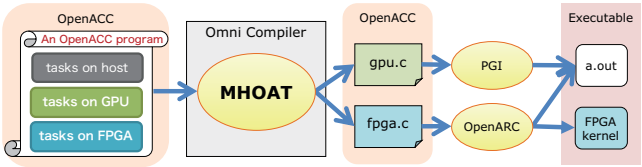

図 2　MHOAT のコンパイルフロー

2.2　我々の提案手法：CAMP

本研究では，GPU+FPGA 協調計算を統一したプログラミングフレームワークにより一つのプログラムとして簡潔に記述できる環境を目指している．このコンセプトを CAMP（Cooperative Acceleration by Multi-device Programming）と呼んでいる．本研究では，CAMP を実現するプログラミング環境として，OpenMP のようにディレクティブ形式でアクセラレータ上での計算（オフロード領域）を指示できる API である OpenACC を用いた環境を開発している．OpenACC は CUDA や OpenCL に比べ，抽象度が高く，既存の CPU コードにディレクティブを追加することで、より簡潔にアクセラレータへのオフロードを記述することができる．よって、OpenCL に書き換える場合に比べて必要なコーディング量を減らすことができ、抽象化により低レイヤーについて考慮する必要がある記述を大きく減らせることから、計算機科学の素養を持たないアプリケーションプログラマーのコーディングの負担を大幅に軽減することができると考えられる。

これまでの研究で，我々は CAMP を実現するための OpenACC source-to-source コンパイラのプロトタイプである MHOAT（Multi-Hybrid OpenACC Translator）を開

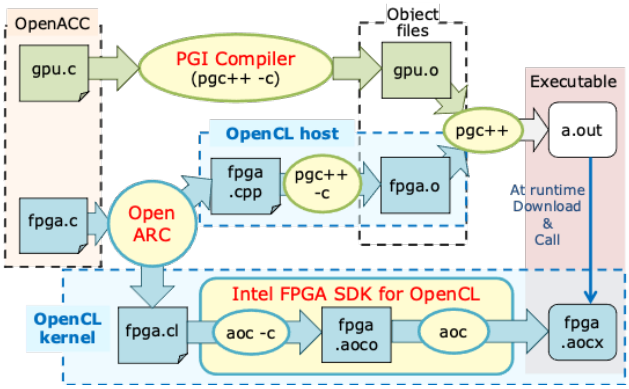

図 3 MHOAT のバックエンドのコンパイルフロー

発した [3][4]．**図 2** にコンパイルフローを示す．このコンパイラは 1 つの OpenACC プログラムを GPU 向け，FPGA 向けの 2 つのファイルに変換，分割する．分割されたファイルはそれぞれのアクセラレータ向け OpenACC コンパイラ（バックエンドコンパイラ）に通す．**図 3** に MHOAT のバックエンドコンパイルのフローを示す．バックエンドコンパイラは，GPU 向けには NVIDIA 社の PGI コンパイラを用いており，FPGA 向けには米国 Oak Ridge National Laboratory（ORNL）で開発されている研究用コンパイラ OpenARC（Open Accelerator Research Compiler）[5] を用いている．OpenARC は OpenACC C プログラムを入力ファイルとして受付ける．FPGA を対象としたコンパイルでは OpenCL ホストコード（C++）と OpenCL カーネルコードを生成する．生成された OpenCL カーネルコードは，Intel 社が提供している Intel FPGA SDK for OpenCL[6] に付属する，aoc コマンドを用いてコンパイルされる．aoc は OpenCL C で記述されたコードを FPGA の回路情報（ビットストリーム）に変換する OpenCL オフラインコンパイラである．PGI コンパイラでは，OpenACC は直接バイナリ形式へ変換される．MHOAT は理化学研究所計算科学研究センター（R-CCS）と筑波大学計算科学研究センター（CCS: Center for Computational Sciences）が共同研究している Omni コンパイラ基盤を用いて実装している．[3][4] では，トイプログラムを MHOAT でコンパイルし，OpenACC による GPU ＋ FPGA 協調計算が可能であることを確認した．

3. 宇宙輻射輸送コード：ARGOT

ARGOT（Accelerated Radiative transfer on Grids using Oct-Tree）は筑波大学 CCS で開発されている宇宙輻射輸送を解くプログラムである．輻射輸送問題は宇宙初期の星や銀河のような天体形成の研究において本質的な要素である．

図 4 に示すように，ARGOT は 2 つのアルゴリズム ARGOT 法 [7] [*1]と Authentic Radiative Transfer (ART) 法 [8] を組み合わせて輻射輸送問題を解く．ARGOT 法のアルゴリズムは点光源からの輻射輸送を計算し，ART 法のアルゴリズムは空間に広がる光源からの輻射輸送を計算する．ART 法は ARGOT コードの中で 90%以上の計算時間を占める重要なアルゴリズムである．本研究では ARGOT 法の計算に GPU を用い，ART 法の演算に FPGA を用いることで，主要演算部分を最適なアクセラレータで計算させることで性能向上を目指す．

[*1] 本論文では，対象とするプログラム名を「ARGOT コード」と表記し，その一部であるアルゴリズムを「ARGOT 法」と表記する．

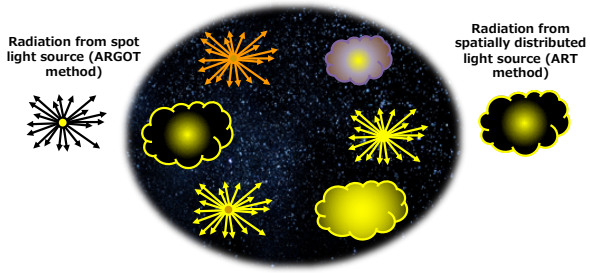

図 4 宇宙輻射輸送コード ARGOT の概観

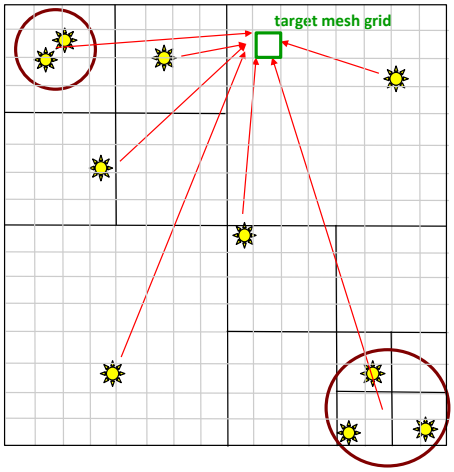

図 5 ARGOT 法の概観

3.1 ARGOT 法

ARGOT 法は，**図 5** に示すように点光源からの輻射輸送を計算するアルゴリズムであり，点光源の数に比例して計算量は増加する．そこで ARGOT 法では，光源の分布を八分木のデータ構造で扱う．これによって，離れたツリーノード内の光源は単一の光源として扱うことができるため，計算を行う光源の数を N から $logN$ に減らすことができる．あるメッシュグリッドを対象とした，各点光源からの輻射輸送による光子束は以下の式で求めることができる．

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

このとき，$L(\nu)$，$\tau(\nu)$，$n(x)$ は振動数 ν での光源の光度，振動数 ν での光学的距離，光を吸収するガス分子の数密度をそれぞれ表し，$\tau(\nu)$ は以下の式で求められる．

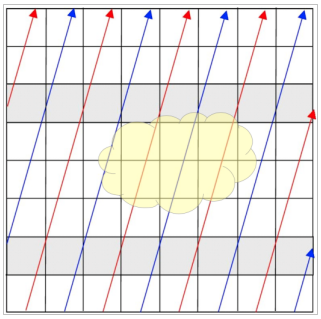

図 6 ART 法で用いられているレイトレーシングの概念図．矢印はレイを表し，黄色の雲は反応を計算するガスを表す．

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

3.2 ART 法

ART 法では問題空間を 3 次元のメッシュに分割し，その中でレイトレーシングを行うことで輻射輸送の計算を行う．**図 6** に示すように，レイは境界から発射され，それぞれのレイが平行に直進し，反射や屈折はしない．

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

式 (3) は ART 法の演算を表し，この式をレイがメッシュを通過する度に計算する．式における ν, I_ν^{in}, I_ν^{out}, $\hat{n}$, $\Delta\tau$, S_ν はそれぞれ周波数，入力放射強度，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 を表し，ART 法の計算は全て単精度浮動小数点数を用いて行われる．レイの方向 (角度) は HEALPix アルゴリズム [9] によって求められる．典型的な問題サイズでは，メッシュ数は 100^3 から 1000^3 の規模になり，レイの種類 ((ϕ, θ) の組み合わせの数) は少なくとも 768 方向になる (HEALPix における解像度パラメータ $N_{side} = 8$ の場合)．式 (3) において，ART 法における演算ボトルネックは指数関数である．周波数 ν 毎に 1 回の指数関数の計算が必要であり，周波数の数は問題の設定に依存するが $1 \leq \nu \leq 6$ であり，1 メッシュ通過毎に複数回の指数関数呼び出しを行わなければならない．

ART 法はレイトレーシングを用いているため，ある 1 つのレイに関する計算は進路に応じて順序通りに計算しなければならないが，異なるレイの間には計算の依存関係がなく並列に計算できる．しかしながら，ART 法を SIMD-like (CPU, GPU など) なアーキテクチャで実装する際には 2 つの問題がある．1 つ目は，メッシュデータに対するメモリアクセスパターンがレイの方向によって様々 (数百〜数千パターン) になることである．複数のレイの計算を SIMD 方式で計算する際に，メッシュデータがメモリ上で連続しない場合があり得る．したがって，キャッシュヒット率の低下や GPU においてメモリアクセスがコアレッシングされないことが問題になる．2 つ目に，メッシュに対する積分計算が衝突する可能性があることである．同じメッシュを隣接した複数のレイが通過する (図 6 の灰色のメッシュ部分) 可能性があるため，メッシュ上の複数のレイの効果を重ね合わせる必要があり，これを同時処理するためには，問題を回避するために atomic 演算を用いるか，隣接するレイを同時に計算しない (例えば，図 6 では，赤色のレイと青色のレイに分けて計算している) といった方法が必要となる．ただし，前者の方法では atomic 演算によるオーバーヘッドがあり，後者の方法ではメモリアクセスがより飛び飛びになるオーバーヘッドがある．

こうした ART 法の性質から，我々は CPU や GPU のような SIMD-like のアーキテクチャでは ART 法の処理が十分高速化できないことを経験的に把握している．一方で，FPGA はオンチップの内蔵メモリを持ち，低レイテンシ・高バンド幅なランダムアクセスが可能である．したがって，我々は FPGA が ART 法の計算により適していると考えている．

4. MHOAT 用 ARGOT コードの実装

2.1 節でも述べた通り，我々は CUDA+OpenCL 版の ARGOT コードを実装したが [2]，その後，GPU のみをターゲットとした OpenACC C 版 ARGOT コードについても実装を行った [10]．このコードは，MHOAT を用いたプログラミングを見据えた予備評価として，ARGOT コードを OpenACC で新たに実装し直したものである．今回は，このコードを MHOAT でコンパイル可能とするために書き換えを行った．言語は C である．なお，GPU のみをターゲットとした OpenACC 版 ARGOT では，PGI コンパイラを用いていた．PGI コンパイラは MHOAT で GPU 向けバックエンドとして用いられているため，今回の変更は主に ART 法の部分のみを FPGA で実行させるための書き換えと，FPGA 向けバックエンドコンパイラである OpenARC への対応である．変更点を要約すると，以下のようになる．

- オフロード領域のターゲットアクセラレータを指定するディレクティブの追加
- ホストとデバイス間のデータ転送の見直し
- FPGA 回路合成時のリソースを削減するためのコード修正
- OpenARC で未サポートの記述へのワークアラウンド対応

4.1 ターゲットデバイスの指定

現状の OpenACC は，一つのプログラムで複数種類のアクセラレータを使用することが論理的には可能であるものの，実際には仕様上は想定されておらず，アプリケーションプログラマーによって CAMP コンセプトにおけるプログラミングが行われる際のユーザビリティが不十分である．具体的には任意の場所でデバイスを指定して切り替え

```
1 void fuga() {
2 #pragma accomn target_dev(FPGA)
3
4 #pragma acc data copy(a, b)
5 #pragma acc kernels loop independent
6         for(i=0; i < N; i++)
7                 a[i] = a[i] + b[i];
8 }
```

図 7 オフロード先指定ディレクティブの使用例

ることになるが，オフロード領域ごとに指定ができないため，現在どのデバイスで実行することになっているかの確認が不便であり，可読性，保守性に問題がある[*2]．そこで，我々は OpenACC に独自拡張ディレクティブ

`#pragma accomn target_dev(` *DEVICE* `)`

を追加する．”accomn”は OpenACC 向けのディレクティブである”acc”に，Omni Compiler での独自拡張であることを示す ”omn” をつけたものである．**図 7** に，このディレクティブの使用例を示す．このディレクティブは，main 関数以外の関数の先頭に一つだけ指定することができる．現在の実装では，異なるデバイスをターゲットとするオフローディングは関数単位のみに制限される．このディレクティブに enum 型の *GPU* または *FPGA* を指定することで，指定された関数に含まれるオフロード領域をどちらのアクセラレータ上で実行するか指定することができる．言い換えると，このディレクティブを指定する関数には，GPU，FPGA どちらかがアクセラレータとして利用されることを想定した処理（kernels ディレクティブ，data ディレクティブ，routine ディレクティブなど）を記述する必要がある．このディレクティブを記述することで，MHOAT では GPU と FPGA，それぞれのオフロード領域が一つの OpenACC プログラムに含められているコードをコンパイルできるようになっている．

4.2 複数デバイスを考慮したデータ転送

MHOAT 使ったプログラミングを行うと，プログラム中で演算加速に使用するデバイスの種類が 2 つ（またはそれ以上）存在することになる．しかし，現状の OpenACC は，デバイス間のデータ転送には対応していない．したがって，データ転送はホスト CPU のメモリを経由して行うことになり，デバイス間のデータ転送を行う際には，一旦あるデバイスからホストにデータを転送してから，さらにホストから別のデバイスへデータを転送するという作業を手動で記述する必要がある．したがって，ホストがどちらのアクセラレータとの間でデータ転送を行うかを意識して記述する必要がある．具体的には 4.1 節で説明している通り，目的のアクレラレータが指定された `accomn target_dev` ディレクティブの存在する関数の中に，データ転送を記述する．ARGOT コードでは，ARGOT 法と ART 法の両方で使われる変数が存在する．このような変数は元々の GPU 用 OpenACC コードでは一度だけデータ転送（Host to Device）を行うようになっていたが，今回の修正では，ART 法でのデータ転送（FPGA への転送）も新たに追記した．

ここで，FPGA の向けのコードをコンパイルする OpenARC が正式にサポートしているのは，OpenACC 1.0 までである（2.0 以降のディレクティブには部分的にしか対応していない）．しかし，元の GPU 向け OpenACC 版 ARGOT コードは PGI の対応する上位の OpenACC バージョンに合わせて記述されていた．したがって，データ転送のディレクティブを OpenARC でコンパイル可能なように書き換えを行った．例えば，同じ変数を別々のデバイスへコピーする場合，安全のため，もしくは同じコードで一つのデバイスのみでも演算加速させられるように，その変数が既にデバイスに転送されていてもエラーにならない記述にすべきである．しかし，OpenACC 1.0 ではデータ転送ディレクティブの clause に `copy` ではなく，`present_copy`（または `pcopy`）のように present_〜を指定しなければこのような動作はしない（OpenACC 2.5 以降は present_を付けない場合も，付けた場合と同様の動作をする）．したがって，`pcopyin` などに書き換えを行った．なお，異種デバイス間のデータ直接転送記述への対応は MHOAT における今後の課題である．

4.3 FPGA でのリソース削減を考慮した修正

MHOAT におけるプログラミングを行う場合，FPGA の回路コンパイル（OpenCL カーネルのコンパイル）についても考慮する必要がある．特に，実アプリケーションの場合，FPGA 上での回路リソースの使用率は大きくなることが予想される．また，FPGA 回路のコンパイルは非常に長時間を要することから，事前にリソース増加の要因になりそうな箇所については修正しておくのが望ましい．OpenARC の FPGA 向けコンパイルでは，OpenACC を OpenCL へ変換する．しかし，OpenACC のオフロード部分のコードは比較的大きな変更なく OpenCL のカーネルコードとして出力される．これは，OpenARC での FPGA プログラミングは，OpenARC のバックエンドで使用される Intel FPGA SDK for OpenCL のプログラミング解説 [11] で推奨されているシングルワークアイテムでの FPGA 実行を想定しており，今回もシングルワークアイテムとしてコンパイルされるようにコードを記述しているためである．こ

*2 さらに，現状の NVIDIA 社製コンパイラでは OpenACC 3.0 以上には対応していないため，デバイスを切り替える方法がランタイム関数しかない．ランタイム関数ではコンパイルオプションによって有効化が切替可能なディレクティブと異なり可搬性が低くなるため，OpenACC で記述するメリットが少なくなってしまう．

```
1  lambda=(2.0*631515.0)/t; // 修正前
2
3  lambda=(2.0F*631515.0F)/t; // 修正後
```

図 8 数値リテラルの修正例

の解説ページに記載されている通り，シングルワークアイテムの場合，この Intel の SDK 下では，OpenCL カーネルを CPU の逐次プログラムと同様のコードで記述するようになっている．したがって，MHOAT でコンパイルする前の OpenACC のオフロード領域に，OpenCL で知られている FPGA リソースを削減するためのテクニックを適用することが概ね可能である．今回のコンパイルでも要求リソース過多のため，回路が合成できないという問題が発生した．しかし，ART 法では先述の通り，単精度浮動小数点数演算しか存在しないことから，ART 法のコードの数値リテラルをデフォルトの double 型から float 型に明示的に書き換える変更（**図 8**）などを行うことで，解決することができた．

4.4 OpenARC が未サポートの記述の修正

OpenARC は研究コンパイラであるため，実装が完全に OpenACC 仕様に準じていない部分がある．これについて対応するために，ART 法のコードの修正を行った．特にデータ転送のディレクティブにおける構造体の扱いは複雑である．OpenARC では SoA（Structure of Arrays）の構造体のメンバ配列を動的確保した場合，その構造体はデータ転送を行うことができない．ただし，この点については PGI コンパイラにおいても．いくつかの制限が存在する．ART 法のコードにはこれに当てはまる構造体が存在するため，AoS（Array of Structures）に修正を行うか，SoA のメンバを静的確保に変更する必要があった．AoS への修正は ARGOT コードのような実アプリケーションの場合，大幅に修正コストが掛かってしまう可能性がある．今回は静的確保を行っても問題ない配列サイズであったため，静的確保へ修正を行った．また，OpenARC では変換の際に，C の入力ソースコードのうち，オフロードに関係しない部分はそのまま変換せずに C++のランタイムライブラリをリンクして，プログラム全体を C++としてコンパイルする．その際，C と C++で整合性の取れないコードが存在した場合にエラーとなってしまう．したがって，この部分を回避するように書き換えを行える必要がある．具体的には，**図 9** のように，単に C++コンパイラの場合に `extern "C"` が有効になるように記述するだけではうまくいかない．MHOAT を実装している Omni コンパイラ基盤は C 言語のコンパイルのみに対応しているため，ソースコードを C コンパイラでコンパイルすることになる．しかし，ifdef は C++でコンパイルする場合に有効になるため，MHOAT でのコンパイル時には `extern "C"`が無視され，出力コードには `extern "C"`が生成されない．また，`extern "C"`は C 言語としてコンパイルすることはできない．ただし，MHOAT ではヘッダーは解析対象ではあるものの，変換対象ではないため，ヘッダーに図 9 のような記述をした場合，MHOAT では無視されるが，バックエンドコンパイラでも同じコードがもう一度読み込まれて解析されるため，正常にコンパイルすることが可能である．したがって，ヘッダーを追加で用意してそこに記述することが必要である．

```
1  #ifdef __cplusplus
2  extern "C" {
3  #endif
4  ...
5  #ifdef __cplusplus
6  }
7  #endif
```

図 9 extern ”C”の記述例

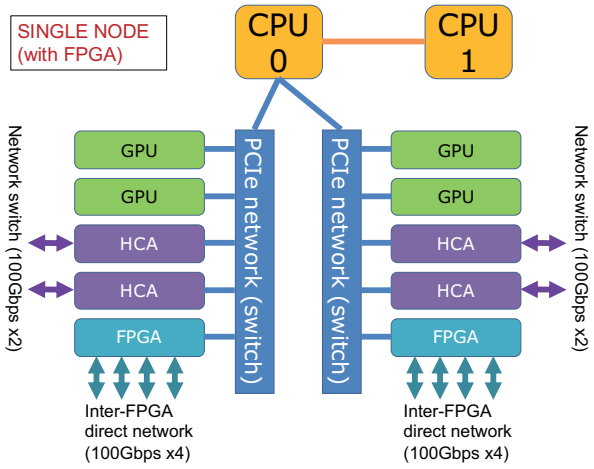

図 10 Cygnus の GPU + FPGA 混載ノードの構成

表 1 Cygnus のノード環境

CPU	Intel Xeon Gold 6126 (12C / 2.6GHz) x2
Host Memory	DDR4-2666 16GiB x12
GPU	NVIDIA Tesla V100 (32GiB HBM2 PCIe 3.0 x16) x4
FPGA	Intel Stratix 10 GX 2800 (BittWare 520N[13] PCIe Gen3 x16) x2
OS	CentOS 7
GPU compiler	PGI Compiler 20.4, CUDA11.0, OpenMPI GDR 4.0.3
FPGA compiler	OpenARC V0.63
OpenCL compiler	Intel FPGA SDK for OpenCL 19.4

5. コンパイルの確認と実行テスト

コンパイル及び実行テストは，筑波大学 CCS で稼働しているスーパーコンピュータ，Cygnus[12]（**図 10**，**表 1**）上で行った．なお，Cygnus には GPU のみが搭載されているノードと GPU と FPGA が混載されているノードが存在するが，今回は GPU と FPGA が両方搭載されている 1 ノードのうち，図 10 中の CPU0 及び CPU0 に接続されている GPU，FPGA をそれぞれ 1 台ずつ用いた．

コンパイルについては，先述の通り，MHOAT で生成された FPGA 向けコードのコンパイルのための各種工夫を施した結果，コンパイルが可能となった．これにより，実アプリケーションで本プログラミング環境を用いたコンパイルが可能なことを確認できた．

実行テストの結果，現状では計算結果の誤差，及び計算時間の大幅な増加が確認されている．計算時間については ART 法の部分で延びており，FPGA の最適化が不十分であると考えられる．現在，性能評価に向けて，これらの修正を進めている．

6. 今後の展望

CAMP コンセプトに基づくマルチデバイスプログラミングに関し，MHOAT の研究と並行して，OpenACC を GPU，FPGA それぞれのアクセラレータ用のコードに変換する際に，GPU 向けコードと FPGA 向けコードをどちらも OpenCL へ変換することも，R-CCS と CCS の共同研究の下で計画している．OpenCL はヘテロジニアスコンピューティングに対応した言語であることから，中間コードとしての利用を考えている．現在の MHOAT には入力するコードの記述に制限が存在する．バックエンドコンパイラが複数に分かれていることから，両デバイス向けコンパイラ双方の影響を考慮したコードを記述する必要がある．OpenARC では C++へ変換する際の問題があり，より制限を複雑化させていた．そこで，現在 MHOAT の実装に用いている Omni コンパイラ基盤を用いて，OpenACC to OpenCL コンパイラの開発を計画中である．**図 11** にコンパイルフローの構想を示す．一つのコンパイラに閉じて，GPU と FPGA 双方の OpenCL コードを生成，コンパイルすることから，現在の MHOAT が持つ複雑さを解決できると考えられる．

4.2 節で述べた通り，CAMP コンセプトにおけるデータ転送に関しても GPU to FPGA のような，デバイス間通信が可能となるような API を提案する必要がある．今後，仕様と実装を議論していく予定である．また，我々は 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 [14]．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

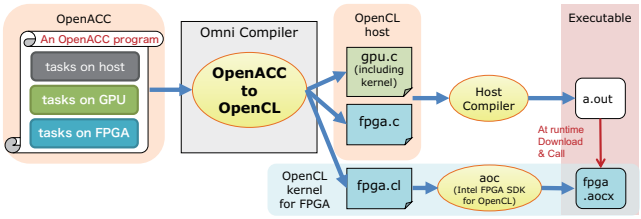

図 11 CAMP 対応 OpenACC to OpenCL コンパイラの構想

可能にすることも課題として挙げられる．

さらに，本研究の最終目標は，マルチノードでの GPU+FPGA 協調計算に対応したプログラミング環境へと進化させることである．今回のコードはシングルプロセス，シングルスレッド実行ではあるものの，OpenMP，MPI をコード中に含んでいる．よって，これまでは OpenACC 単体でのコンパイルしか確認していなかったが，本プログラミング環境と OpenMP，MPI を混ぜたコードをコンパイルできることが今回の実験により確認できた．今後は，この組み合わせでまずマルチノード演算が行えることを確認する．最終的には XcalableACC（XACC）[15] に本手法を組み入れて，XACC によるマルチノード GPU+FPGA 協調計算の実現を目指す．

7. 関連研究

我々の提案している CAMP コンセプトに近い研究は，いくつか存在する．

oneAPI[16] は Intel が提唱している，DPC++を内包したクロスアーキテクチャな API である．DPC++は C++拡張であり，SYCL が組み込まれている．単一なプログラミング言語により，CPU，GPU，FPGA 間でポータブルなコードを目指している．しかし，GPU と FPGA では異なるアーキテクチャであるため，同じコードをオフロードすることは性能面から効果的でないと考えられる．我々は，oneAPI を使用した ARGOT コードの GPU+FPGA 協調計算を試みた [17]．これ以前に，我々は DPC++のみで GPU+FPGA 協調計算が可能であることを確認している [18]．しかし，実アプリケーションの全てのコードを DPC++で再実装することは非常に負担が大きいことから，既存の CUDA カーネルと 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して実験を行った．性能面では，CUDA ＋ OpenCL の場合とほとんど性能差がなかったものの，今回のような実験は CUDA 及び OpenCL で既に記述された実アプリケーションコードが存在する場合にのみ可能である．特に FPGA での実行を想定したプログラムは少なく，多くのアクセラレータ用実アプリケーションコードの実装は CUDA であることから，このようなケースは少ないことが予想される．よって，実用面では学習コストも大きく，実際にユーザビリティを大きく向上させることは難しいと考えられる．

OmpSs[19] は OpenMP と同様のタスク並列処理を基本としており，OpenMP のオフロード機能を実現するためのプロトタイピング環境として利用されている．ディレクティブベースの比較的高レベルの抽象化による GPU や FPGA へのオフローディングを提供し，プログラマーの負担を軽減しようとしている．しかし，我々の提案手法と異なり，C/C++から CUDA/OpenCL コードへのトランスレーターは無いため，バックエンドである CUDA や OpenCL の記法にしたがって，カーネル関数を記述しなければならない．2 つの記法が混ざることは複雑であり，ユーザーの混乱を招く恐れがある．

[20] では，EngineCL による CPU と GPU 及び FPGA 協調実行について述べている．この研究の主眼はロードバランシングによる資源の有効活用である．EngineCL は OpenCL ベースであるため，OpenMP のようなディレクティブベースの API のようにコードを追加するだけでなく，CPU コードを書き換える必要がある．これはユーザーにとっての負担となる．本研究では，よりユーザーの負担を減らせるようにディレクティブベースで抽象度の高い API を目指している．StarPU[21] はノード内の CPU とアクセラレータにデータを割り当て，ワークシェアリングを行う，C，C++に対応したランタイムライブラリのシステムである．対応しているアクセラレータは GPU，メニーコア CPU などであり，現状 FPGA には非対応となっている．SkePU[22] は上記と共通点も多いが，アルゴリズミックスケルトンを採用した，C++テンプレートライブラリである．[23] では，既存の OpenMP や OpenACC では，単一のアクセラレータを使用することのみが想定されており，複数アクセラレータを用いた場合のアクセラレータ間のデータ交換，データと計算の分解，それらの同期の処理のサポートなどが不十分であると指摘している．そこで，ノード内の複数アクセラレータのサポートのために OpenMP の言語拡張を設計し，ランタイムやコンパイラのプロトタイプ実装を評価している．ただし，ここでの実装は複数 GPU 対応のみであり，FPGA や他のアクセラレータの実装や異なるアクセラレータを組み合わせた場合の実装はされていない．また，それぞれのループごとにデータやイテレーション分割をしてデバイス間作業分散することを想定しており，異なる計算特性のコードを異なるデバイスで処理することは想定されていない．

しかし，本研究の目的はこのような負荷分散ではない．それぞれのアクセラレータで高パフォーマンスが得られる処理が異なることから，計算の特性を考慮せずにオフロードした場合，[24] のように FPGA 複数台でも GPU のパフォーマンスより低くなる場合がある．本研究はユーザーがそれぞれのアクセラレータでアドバンテージのある処理を行うように記述し，高パフォーマンスを得ることを目的としている．

謝辞　本研究の一部は，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）における研究課題「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 JSPS 科研費 21H04869 の助成，及び筑波大学計算科学研究センターの学際共同利用プログラム（Cygnus），及び理研の大学院生リサーチ・アソシエイト制度，及び JST 次世代研究者挑戦的研究プログラム JPMJSP2124 の支援による．また，ARGOT コードの提供及び本研究に関する議論において，筑波大学計算科学研究センター宇宙物理研究部門の梅村雅之教授，吉川耕司准教授，安部牧人研究員に深く感謝する．

参考文献

[1] November 2021 — TOP500, <https://www.top500.org/lists/top500/2021/11/>

[2] Kobayashi, R., Fujita, N., Yamaguchi, Y., Boku, T., 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).

[3] 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) .

[4] 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，(2019)．

[5] Lee, S., Lambert, J., Kim, J., Vetter, J. S., & Malony, A. D: OpenACC to FPGA: A Directive-Based High-Level Programming Framework for High-Performance Reconfigurable Computing, Poster summary of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC18), (2018).

[6] Intel: Intel FPGA SDK for OpenCL, <https://www.intel.co.jp/content/www/jp/ja/software/programmable/sdk-for-opencl/overview.html>

[7] Okamoto, T., Yoshikawa, K. and Umemura, M.: argot: accelerated radiative transfer on grids using oct-tree, Monthly Notices of the Royal Astronomical Society, Vol. 419, No. 4, pp. 2855–2866 (online), DOI: 10.1111/j.1365-2966.2011.19927.x (2012).

[8] Tanaka, S., Yoshikawa, K., Okamoto, T. and Hasegawa, K.: A new ray-tracing scheme for 3D diffuse radiation transfer on highly parallel architectures, Publications of the Astronomical Society of Japan, Vol. 67, No. 4 (online), DOI: 10.1093/pasj/psv027 (2015).

[9] Gorski, K. M., Hivon, E., Banday, A. J., Wandelt, B. D., Hansen, F. K., Reinecke, M. and Bartelmann, M.: HEALPix: A Framework for High-Resolution Discretization and Fast Analysis of Data Distributed on the Sphere, The Astrophysical Journal, Vol. 622, No. 2, pp. 759–771 (online), DOI: 10.1086/427976 (2005).

[10] 小林諒平，藤田典久，山口佳樹，朴泰祐，吉川耕司，安部牧

人, 梅村雅之：宇宙幅射輸送コード ARGOT の OpenACC による GPU 実装, 情報処理学会研究報告, 2020-HPC-175, (2020).

[11] Intel(R) FPGA SDK for OpenCL™: ベスト・プラクティス・ガイド
<https://www.intel.co.jp/content/www/jp/ja/docs/programmable/683521/17-1/single-work-item-kernel-versus-ndrange.html>

[12] 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/#Cygnus>

[13] 520N - BittWare FPGA Acceleration,
<https://www.bittware.com/fpga/520n/>

[14] 小林諒平，藤田典久，朴泰祐：OpenACC と OpenCL の混合記述による GPU-FPGA デバイス間連携，情報処理学会研究報告，2020-HPC-177，(2020).

[15] Masahiro Nakao, Tetsuya Odajima, Hitoshi Murai, Akihiro Tabuchi, Norihisa Fujita, Toshihiro Hanawa, Taisuke Boku, Mitsuhisa Sato: Evaluation of XcalableACC with Tightly Coupled Accelerators/InfiniBand Hybrid Communication on Accelerated Cluster, International Journal of High Performance Computing Applications, (2019).

[16] Intel oneAPI: A Unified X-Architecture Programming Model.
<https://software.intel.com/content/www/us/en/develop/tools/oneapi.html >

[17] Kashino, R., Kobayashi, R., Fujita, N., Boku, T.: Multi-hetero Acceleration by GPU and FPGA for Astrophysics Simulation on oneAPI Environment, In International Conference on High Performance Computing in Asia-Pacific Region (HPCAsia2022), Association for Computing Machinery, pp. 84–93. DOI:https://doi.org/10.1145/3492805.3492817 (2022).

[18] 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，2020-HPC-180，(2021).

[19] Bosch, J., Filgueras, A., Vidal, M., Jimenez-Gonzalez, D., Alvarez, C., and Martorell, X.: Exploiting Parallelism on GPUs and FPGAs with OmpSs, In Proceedings of the 1st Workshop on AutotuniNg and aDaptivity AppRoaches for Energy efficient HPC Systems (ANDARE '17), Association for Computing Machinery, Article 4, pp. 1-5 (2017). 入手先 ⟨https://doi.org/10.1145/3152821.3152880⟩

[20] Dávila Guzmán, M.A., Nozal, R., Gran Tejero, R. et al.: Cooperative CPU, GPU, and FPGA heterogeneous execution with EngineCL, The Journal of Supercomputing, Vol. 75, pp. 1732-1746 (2019).

[21] Augonnet, C., Thibault, S., Namyst, R., and Wacrenier, P. A.: StarPU: a unified platform for task scheduling on heterogeneous multicore architectures, Concurrency and Computation: Practice and Experience, Vol. 23, No. 2, pp. 187-198 (2011). 入手先 ⟨https://doi.org/10.1002/cpe.1631⟩

[22] Öhberg, T., Ernstsson, A. and Kessler, C.: Hybrid CPU-GPU execution support in the skeleton programming framework SkePU, The Journal of Supercomputing, Vol. 76, pp. 5038-5056 (2020). 入手先 ⟨https://doi.org/10.1007/s11227-019-02824-7⟩

[23] Yan, Y., Lin, P. H., Liao, C., de Supinski, B. R., and Quinlan, D. J.: Supporting multiple accelerators in high-level programming models, In Proceedings of the Sixth International Workshop on Programming Models and Applications for Multicores and Manycores (PMAM '15), Association for Computing Machinery, pp. 170-180 (2015). 入手先 ⟨https://doi.org/10.1145/2712386.2712405⟩

[24] Cadenelli, N., Jaksić, Z., Polo, J. and Carrera, D.: Considerations in using OpenCL on GPUs and FPGAs for throughput-oriented genomics workloads, Future Generation Computer Systems, Vol. 94, pp. 148-159 (2019).