Vol.2018-HPC-165 No.27
2018/7/31

並列FPGAシステムにおける OpenCLを用いた宇宙輻射輸送コードの演算加速

藤田 典久[1]　小林 諒平[1,2]　山口 佳樹[2,1]　朴 泰祐[1,2]　吉川 耕司[1,3]　安部 牧人[4]　梅村 雅之[1,3]

概要：近年注目されている High Performance Computing (HPC) における挑戦の一つに，どのようにして Field Programable Gate Array (FPGA) 技術を用いて，高い性能と低い低消費電力を次世代スーパーコンピュータシステムで達成するかというものがある．従来手法ではソフトウェアの開発者が Hardware Description Language (HDL) を用いて FPGA 回路を開発することは困難であったが，近年の FPGA における開発環境の進歩により，高位合成の利用が一般的になりつつあり，HDL の記述なしに FPGA 開発が可能になりつつある．本研究では，初期宇宙の研究に重要な輻射輸送を解くプログラム Accelerated Radiative transfer on Grids using Oct-Tree (ARGOT) で用いられているアルゴリズムである Authentic Radiation Transfer (ART) 法を OpenCL で記述して FPGA 向けに最適化を行い，また，今後の展望として，ART 法の計算をどのようにして複数 FPGA で並列計算を行うかについて述べる．これまでの研究では，FPGA 内の Block RAM (BRAM) に収まる大きさの問題しか解けず，ARGOT で実際に計算したい問題サイズに対応できなかったが，大容量の DDR メモリを併用することで実用的な問題サイズを FPGA で解けるようになった．CPU, GPU, FPGA 間での性能比較を行い，CPU と比べて 6.9 倍の高速化が達成され，GPU との比較では GPU と同程度の性能を達成した．FPGA 実装の性能は GPU と同程度であるが，自ら通信機構を操作できる FPGA の方が通信オーバーヘッドは GPU と比べると小さく，並列計算を行う際の性能は GPU の性能を超えられると考えられ，今後，並列 FPGA 計算の実装を行う予定である．

1. はじめに

近年，HPC の分野では GPU や Intel Xeon Phi といったアクセラレータをノード内に持つヘテロジニアスなシステムが広く用いられている．アクセラレータは多数のコアと広い演算幅の Single Instruction Multiple Data (SIMD) 演算器を持つため，ヘテロジニアスなシステムでは CPU はレイテンシコア，アクセラレータはスループットコアとして用いられる．それらのアクセラレータに加え，FPGA が CPU とアクセラレータのトレードオフの中間にあるデバイスとして注目されている．しかしながら，ソフトウェアの開発者にとっては，Verilog HDL や VHDL に代表される伝統的な HDL を用いた FPGA ロジックの開発は困難である．一方で，HPC において絶対性能に加えて電力効率は重要な問題となってきており，アクセラレータによって提供される高いレベルの並列演算を利用するだけでは十分ではない．

近年の FPGA における開発環境の進歩により，高位合成 (High Level Synthesis; HLS) の利用が一般的になりつつある．HLS は C, C++, OpenCL といったソフトウェア向けの言語から FPGA の回路を開発する手法であり，HDL を用いることなく FPGA 回路の開発できる．例えば， Intel と Xilinx は OpenCL 言語を用いる Software Development Kit (SDK) を HLS 開発環境の 1 つとして提供しており，OpenCL による FPGA 回路の開発が行える．

また，アプリケーションのどの処理を FPGA にオフロードするかを決定することは FPGA を用いてアプリケーションを加速する際に重要であり，計算科学の研究者との “co-design” が重要となる．HLS を用いることで計算科学の研究者が直接 FPGA のプログラミングが行える可能性があり，HLS が HPC における FPGA 利用で重要な役割になると考えている．

我々は，これまでの研究で OpenCL の基本的な性能 [1] や，OpenCL からネットワーク機構を利用する方法について評価を行なった [2], [3]．また，初期宇宙の研究に重要な輻射輸送を解くプログラム ARGOT で用いられているアルゴリズムである ART 法について OpenCL を用いた FPGA 最適化 [4] を行い，CPU と FPGA を用いた性能評価で 14.6 倍の高速化が達成できたが，FPGA 内蔵の

[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
[3] 筑波大学 数理物質科学研究科
[4] 東北大学 天文学教室

SRAM のみを用いて計算していたため問題サイズに制限があり，実用的な問題サイズを解くことができなかった．また，Intel と Xilinx というプラットフォームの違いはあるものの，OpenCL 実装と Verilog HDL 実装による比較を行い [5]，両実装が同程度の性能が達成できることを示したものの，OpenCL 実装の方が回路リソースの消費量が多く，リソース使用量の最適化が OpenCL の課題であることがわかっている．

本研究の目的は [4] で実装した ART on FPGA の実装の最適化および，DDR メモリを用いてより大きな問題サイズを解けるようにし，性能評価を行うことである．また，今後の展望として，ART 法の計算をどのようにして複数 FPGA で並列計算を行うかについて述べる．

2. 関連研究

OpenCL を FPGA で用いてアプリケーションやベンチマークの性能評価を行った論文はいくつか報告されている．[6] では，元々 GPU 向けに作成されたコードをそのまま FPGA 向けに用いても性能が悪く，OpenCL コードが FPGA 向けに最適化されている必要があると述べられている．[7] では，VHDL と OpenCL で同じアルゴリズムを記述し，性能とリソース使用量を比較している．OpenCL 実装は VHDL 実装と同等の性能を得られるものの，OpenCL 実装の方が多くの回路リソースを使用すると報告している．[8] では，XSBench を用いてイレギュラーなメモリアクセスを FPGA と OpenCL を用いて行う場合の性能が評価されており，Intel Arria10 FPGA の性能は Intel Xeon 8-core CPU と比べて 35%悪いものの，FPGA の電力効率が CPU と比べて 50% 良いと報告されている．HPC 研究会においても，[9] や [10] で FPGA と OpenCL を用いた研究報告がなされているが，どちらでも OpenCL の最適化の困難さ，すなわち，CPU や GPU と異なる記述スタイルが必要であると述べられている．

FPGA の絶対性能は GPU など他のアクセラレータと比べると低く，どのような種類の処理を FPGA にオフロードするのかが重要となる．本研究では，OpenCL を用いて宇宙輻射輸送コードの最適化を行う．対象のアルゴリズムは複雑なメモリアクセスパターンを持ち，FPGA に適するアルゴリズムであると考えられる．

3. 宇宙輻射輸送コード: ARGOT

ARGOT は筑波大学 計算科学研究センター (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．ARGOT は 2 つのアルゴリズム ARGOT [11] と ART [12] を組み合わせて輻射輸送問題を解く．ARGOT アルゴリズムは点光源からの輻射輸送を計算し，ART 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 プログラムの中で 90%以上の計算時間を占める重要なアルゴリズムであり，本研究では ART 法の部分に注目し最適化を行う．

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 1** 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．

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

式 (1) 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}, $\hat{\boldsymbol{n}}$, $\Delta\tau$, 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 [13] 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，レイの方向は少なくとも 768 方向になる (HEALPix における解像度パラメータ $N_{side} = 8$)．式 (1) にあるように，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 $1 \leq \nu \leq 6$ である．

ART 法はレイトレーシングを用いているため，ある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．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 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える．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レイテンシの大きさが問題になる．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可能性があり，それらのレイを同時に計算する場合，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といった方法が必要となる．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 といった SIMD スタイルのアーキテクチャは ART 法に適さないと考えている．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・高バンド幅にランダムアクセスが可能である．それに加えて，FPGA であれば ART 法に最適化したメモリアクセス回路をハードウェアに組み込るため，ART 法は FPGA での実装に適したアルゴリズムであると考えている．

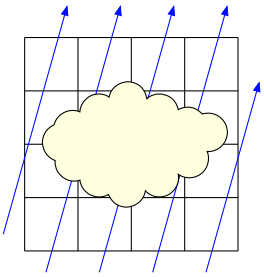

図 1: ART 法で用いられているレイトレーシングの概念図．青い矢印はレイを表し，黄色の雲は反応を計算するガスを表す．

4. ART on FPGA 実装

本章では ART on FPGA の実装について述べる．ただし，ART 法をどのようにして FPGA 上で OpenCL を用いて実装・並列化するのかついては，既に [4] で述べているため，重複する箇所については本原稿では概要を述べるに留める．

4.1 実装の概要

本研究では FPGA の回路実装に Intel 社が自社 FPGA 向けに開発している Intel FPGA SDK for OpenCL を高位合成の処理系として用いる．その SDK は OpenCL 言語にいくつかの独自拡張を加えており，FPGA に特化した処理を記述できる．ART on FPGA では独自拡張の中から Channel と autorun の 2 つ拡張を用いる．Channel はカーネル間のデータ交換を外部メモリではなく FPGA チップ内部で直接行う言語拡張であり，autorun はカーネルに対する属性の拡張で，そのカーネルがホストからの指示なしに自動で起動されるようにする属性である．

図 2 に本実装の概要を示す．図にあるように，本実装は FPGA の中に複数のカーネルを実装し，それぞれを Channel で接続して構成されている．図 2 にある “PE Array” は ART 法の演算を実装しているカーネル群であり，**図 3** にあるように Processing Element (PE), Boundary Element (BE) から構成され，PE と BE が相互にレイのデータを channel 経由で通信することで ART 法の計算を行う．

PE は ART 法の演算カーネルを担当するカーネルである．各 PE は**図 4** にあるように，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 PE に割り当てる．演算用のデータは高頻度にアクセスする必要があるため，BRAM を用いており，それぞれの PE が演算用の BRAM を持つ．BE は PE に対するレイの入出力処理を行うものであり，袖領域に対するレイデータの入出力すなわちレイの初期生成および不要なレイの廃棄と，過去の計算で生成されたレイをレイバッファから読み出す処理，将来の計算で再び用いるためレイバッファに書き出す処理を行う．な

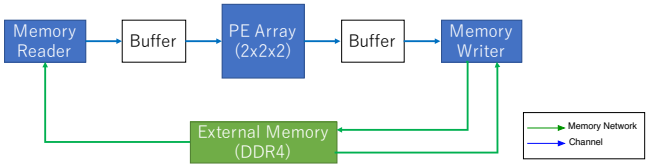

図 2: ART on FPGA 実装の概要．

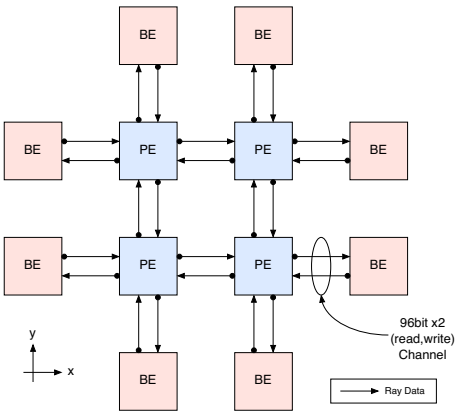

図 3: X, Y 次元における PE と BE の接続ネットワーク図．

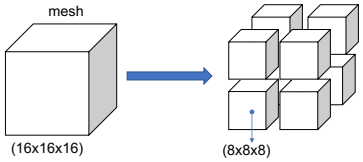

図 4: 空間分割の概念図．大きな問題空間を小さく分割し，それぞれを PE に割り当てる．

お，レイバッファを用いる処理は本原稿で新たに追加したカーネルであり，DDR を用いた計算に重要な処理を行う部分であるため，次節で詳細に述べる．

4.2 DDR 実装

前節で，それぞれの PE で行われる演算は BRAM を用いていると述べたが，そのような実装では FPGA の BRAM の大きさに解ける問題サイズが制限されてしまう．我々は ARGOT プログラムで実用的な問題を計算するために，1FPGA あたり少なくとも 128^3 の問題を割り当てられる必要があると考えている．128^3 のメッシュに必要なメモリ量は 128MB であり，現在の最先端の FPGA の BRAM 容量が高々 20 ～ 30MB であることを考えると，より大きな問題を解くために DDR メモリを併用することは不可欠である．

DDR 実装では, ART 法で解く大きな問題を小さな FPGA の BRAM に格納できるサイズに分割し，ブロック単位で順々に計算を行う．**図 6** に DDR 実装の疑似コードを示す．ただし，図 6 はアルゴリズムの流れを示すものであり，実際の実装を反映しているものではない．実際の実装では，アルゴリズムを前節で述べたように複数のカーネルに分割

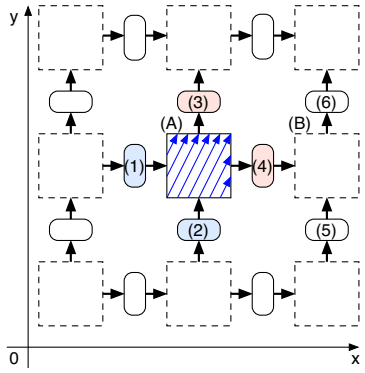

図 5: レイバッファの概念図．赤い箱は出力レイバッファ，青い箱は入力レイバッファ，青い矢印は計算するレイを表す．

```
for dir in ray_directions[] {
  for b in small_blocks[] {
    A: mesh_load(b)
    for i in ipix(dir) {
      for iray in rays(ipix) {
        r = ray_load(iray)
        for m in compute_path(r) {
          C: compute reaction between r and m
        }
        ray_store(iray, r)
      }
    }
    B: mesh_store(b)
  }
}
```

図 6: DDR 実装の疑似コード．

し，Channel を用いて接続され，全体を構成している．

DDR 実装では，DDR を用いるために 2 つの種類のカーネルを追加する．1 つはメッシュデータの計算の進行に伴い入れ替えを行うものであり，もう 1 つはレイデータに関するものである．

メッシュデータに関する DDR アクセスは，PE が持つメッシュデータ用の BRAM を cache の様に使うことで実現される．メッシュデータを DDR メモリから BRAM にコピーし (図 6 の A)，ART 法の計算を BRAM 上で行い (図 6 の B，この段階では DDR アクセスはない)，結果を BRAM から DDR に書き戻す (図 6 の C)．メッシュデータの DDR メモリ上のアクセス順序は固定で予めわかっているため，計算中に次に必要なデータを FIFO バッファに予め読み出しておくことで，演算パイプラインのストールを最小する．

レイデータに関する DDR メモリアクセスは，メッシュデータに対するそれよりも複雑である．DDR メモリ上にレイバッファを確保し，演算に必要なレイデータを格納する．レイバッファに必要なメモリ量は BRAM のサイズに対して大きく，DDR メモリ上に確保しなければならない．レイバッファへのアクセスを**図 5** に示す．図 5 は 2D 空間に簡略化しているが，実際の計算では 3D 空間を扱うため，同様の構造が Z 次元にも存在する．図 5 には 9 つの箱があり，それぞれがメッシュデータを表す．中央にある黒い実線で描かれている箱 (A) は現在計算中のブロックを表す (図 6 の変数 b)．ブロック A の計算に用いられるレイデータは青い 2 つの入力レイバッファ (1 番, 2 番) から読まれ，そして赤い 2 つの出力レイバッファ (3 番, 4 番) に結果が書き込まれる．入力と出力のレイバッファは計算するメッシュの位置によって変化し，例えばブロック B を計算する際には 4 番と 5 番が入力バッファになり，6 番が出力バッファになる．また，ブロック B は計算領域の端に位置しているためブロック B の右側に出力バッファはなく，計算領域の外にレイが出た場合は破棄される．

5. ART on FPGA の性能・リソース使用量評価

5.1 評価環境

性能評価には Pre-PACS version X (PPX) クラスタシステムを用いる．PPX は筑波大学 計算科学研究センターで運用中のシステムであり，同センターが開発を計画している PACS シリーズ・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る．PPX は FPGA プラットフォーム比較用に Intel FPGA ノードグループ, Xilinx FPGA ノードグループの 2 グループからなり，それらのノードを一体運用しているが，本原稿では Intel FPGA のみを用いるため，本節では Intel FPGA を持つノードについてのみ述べる．**表 1** に PPX システムの Intel FPGA ノードの仕様を示す．各ノードに Broadwell Xeon CPU ×2, NVIDIA P100 GPU×2, InfiniBand EDR HCA×1, BittWare FPGA ボード ×1 が搭載されている．CPU と GPU は PCIe Gen.3 x16 レーンで接続されているが，CPU と FPGA は FPGA ボードの仕様により PCIe Gen.3 x8 レーンで接続されている．Quick Path Interconnect (QPI) を経由する PCIe アクセスによる性能低下を回避するために，FPGA と GPU 実装の性能評価時は各デバイスが直接接続されている CPU を用いる．

性能評価では，ARGOT プログラムから作成したベンチマークプログラムを用いる．ただし，このプログラムは ART 法の演算部分のみ含んでおり，ノード間通信は行わず 1 ノードのみで計算を行う．レイデータについては HEALpix ライブラリを用いて ARGOT プログラムと同様の方法で作成するが，ART 法はメッシュデータの値によって計算負荷が変化しないためメッシュデータの入力には疑似乱数を用いる．

評価に用いるデータサイズは 16^3 から 128^3 までの間で変化させる．現在の FPGA 実装は 8 PE ($= 2^3$) で構成さ

表 1: 評価環境

CPU	Intel Xeon E5-2660 v4 × 2
CPU Memory	DDR4 2400 MHz 64 GB (8 GB × 8)
GPU	NVIDIA Tesla P100 (PCIe)
Infiniband	Mellanox ConnectX-4 EDR
Host OS	CentOS 7.3
Host Compiler	gcc 4.8.5
OpenCL SDK	Intel FPGA SDK for OpenCL 16.1.2.203
CUDA	CUDA 8.0.61
FPGA	BittWare A10PL4 (10AX115N3F40E2SG)
FPGA Memory	DDR4 2133 MHz 8 GB (4 GB × 2)
Communication Port	QSFP+ × 2 (40 Gbps × 2)

れ，そして各 PE は 8^3 メッシュを格納できる BRAM を持つ．すなわち，16^3 サイズの場合は全てのメッシュデータを FPGA の BRAM に格納できる．HEALpix アルゴリズムの解像度パラメータ N_{side} は全ての問題サイズで 8 に設定しており，異なる 768 種類のレイを生成する (768 はレイの方向の数であって，レイの本数ではない)．性能評価では，演算時間は CPU 上で計測し，デバイス上で計算を行うためのコスト (カーネルの起動・同期) を含むが，FPGA と GPU の性能評価においてデータ転送にかかる時間は演算時間に含めない．また，本評価では性能の指標として “mesh/s” という単位を用いる．これは 1 秒間に何個のメッシュをレイトレースできたかを示すものであり，高い値は高い性能を意味する．

5.2 リソース使用量

表 2 に FPGA 実装のリソース使用量を示す．Adaptive Logic Modules (ALMs) と Registers は FPGA の基本要素であり論理回路を構成するために用いられる．M20K と MLAB は分散 BRAM を表し，DSP は浮動小数点数演算に用いられるブロックである．ART 法の演算は全て単精度で構成されており，指数関数を含め全ての演算は DSP ブロックに実装される．“Freq.” は OpenCL カーネルから生成された回路のクロックドメインにおける動作周波数を表す．

問題サイズ 16^3 と 32^3 のリソース使用量を比較すると，大きな差があることがわかる．これらの差は前章で述べたレイバッファの制御カーネルによって発生する．16^3 は問題サイズが小さく，レイバッファを使わずに問題を解けるため，それらのカーネルは FPGA 内部に実装されないためである．また，レイバッファ制御カーネルはクリティカルパスにもなっており，問題サイズ 16^3 と 32^3 の比較で約 20MHz 動作周波数が低下している原因であり，今後これらのカーネルの最適化を行わなければならない．

表 2 にあるように，ALMs と registers が最も多くのリソース (40%) を消費しており，これらのリソース消費が FPGA のパフォーマンスを向上させる際のボトルネックとなる．FPGA で高いパフォーマンスを得るためには，PE の数を増やし，全ての DSP を利用して ART 法の計算を行う状況が理想である．現在のリソース使用量の割合のまま全ての DSP を利用すると仮定すると，およそ 190%の ALM が必要という計算になり実現不可能である．したがって，リソース面における OpenCL コードの最適化が性能を高めるために必要であるといえる．

問題サイズ 32^3, 64^3, 128^3 における周波数の差は，OpenCL コンパイラによって発生している．これらのサイズで用いている OpenCL コードは，問題サイズやループカウントなど一部の定数を除いて同じである．コンパイラによって生成される Verilog HDL コードは可読性が低いため，OpenCL コードがどのように回路として表現されているかを知ることが難しく，周波数の差がどのように発生しているかを解析することは困難である．

5.3 性能評価

表 3 と**図 7** に CPU, GPU, FPGA 間における性能評価の結果を示す．CPU 実装は C 言語で記述され OpenMP によるスレッド並列計算が実装されている．“CPU(14C)” は CPU 実装を 14 スレッド (= 1 ソケット) で実行する場合，“CPU(28C)” は 28 スレッド (= 2 ソケット) で実行する場合を表す．GPU 実装は CPU 実装を元に実装されており，CUDA によって GPU カーネルが記述されている．

FPGA 実装は問題サイズ 16^3, 32^3, 64^3, 128^3 の場合において，それぞれ 1283 M mesh/s, 1165 M mesh/s, 1111 M mesh/s, 1134 M mesh/s の性能を達成した．3 つの実装の中で CPU 実装が最も遅く，CPU 実装が最も性能の良い問題サイズ 64^3 の場合であっても，GPU 実装と比較して 6.8 倍，FPGA 実装と比較して 6.9 倍遅い．CPU 実装が問題サイズ 64^3 と 128^3 の間で性能が低下しているのは，メッシュデータが大きくなり，キャッシュヒット率が低下したからだと考えられる．

GPU 実装において，問題サイズ 16^3 の性能が著しく悪く，問題サイズ 64^3, 128^3 と比べて 10 倍以上性能が悪い．この性能低下は問題サイズが小さすぎ，P100 GPU が持つ 3584 CUDA Core に対して十分な演算の並列度が得られないためと考えられる．

GPU 実装とは異なり，FPGA 実装は全ての問題サイズで高い性能を発揮する．FPGA の性能がこのような性質を示す理由は，FPGA 実装が空間並列だけでなくパイプライン並列によっても並列化されているためと考えられる．1

表 2: リソース使用量と動作周波数.

size	# of PEs	ALMs	(%)	Registers	(%)	M20K	(%)	MLAB	DSP	(%)	Freq. [MHz]
(16, 16, 16)	(2, 2, 2)	132,283	31%	267,828	31%	739	27%	14,310	312	21%	193.2
(32, 32, 32)	(2, 2, 2)	169,882	40%	344,447	40%	796	29%	21,100	312	21%	173.8
(64, 64, 64)	(2, 2, 2)	169,549	40%	344,512	40%	796	29%	21,250	312	21%	167.0
(128, 128, 128)	(2, 2, 2)	169,662	40%	344,505	40%	796	29%	21,250	312	21%	170.4

表 3: FPGA, CPU, GPU 間の性能比較の結果．数値の単位は “M mesh/sec”.

Size	CPU(14C)	CPU(28C)	P100	FPGA
(16,16,16)	112.4	77.2	105.3	1282.8
(32,32,32)	158.9	183.4	490.4	1165.2
(64,64,64)	175.0	227.2	1041.4	1111.0
(128,128,128)	95.4	165.0	1116.1	1133.5

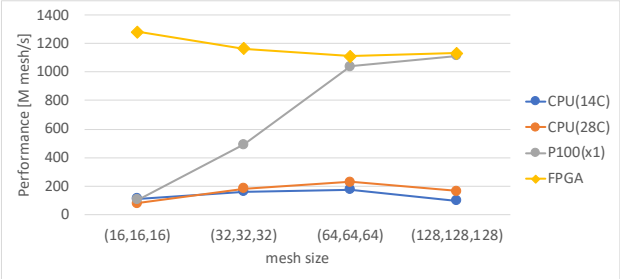

図 7: FPGA, CPU, GPU 間の性能比較の結果グラフ.

クロックあたりの性能がどの問題サイズでも同程度 (6.64 ∼ 6.70 mesh per cycle) であり，FPGA 実装の性能は動作周波数によって律速されていることがわかる．したがって，FPGA 実装の性能を向上させるためには，周波数を高く保つことが重要であると言える．FPGA 実装は大きな問題サイズにおいても GPU 実装とほぼ同程度の性能が得られており，既に述べたように FPGA 実装はリソース面，周波数面どちらにおいてもさらなる最適化を行う必要があるものの，FPGA の基本的な演算性能は GPU に対して十分対抗できるものであると考えられる．

6. Next Step: FPGA 並列化

6.1 FPGA 間並列化

前章で FPGA のパフォーマンスが GPU と同程度であると確認した．これをふまえて，我々は，ART on FPGA の実装にネットワーク機構を追加し，問題サイズのさらなる拡大と性能の向上を行おうと考えている．GPU のプログラミングモデルでは，GPU はスレーブデバイスであり，ノード間通信も含めてホスト CPU によって制御されなければならない．ノード間通信は CPU によって制御されるため，CPU と GPU は通信を開始する前に同期する必要がある．NVIDIA GPU と Mellanox HCA を組合せて利用する場合，GPUDirect for RDMA (GDR) [14] を通信性能向上のために利用できる．GDR を用いることで通信のバンド幅とレイテンシを改善できるが，依然として通信は CPU によって開始されるため，通信のオーバーヘッドは無視できない．

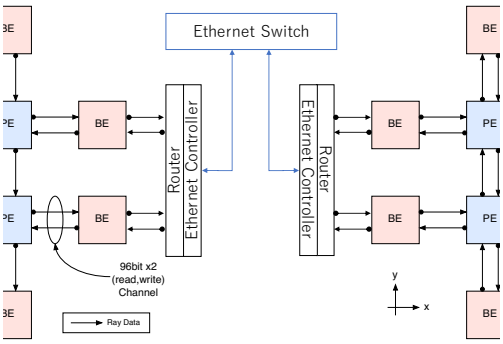

図 8: ART on FPGA を複数の FPGA で並列化する際の通信の概念図.

一方で，最新の FPGA (Intel Stratix10 など) は複数の高速通信機構 (最大で 100Gbps × 4) を持つ．現状の ART on FPGA の性能は GPU と同程度であるが， FPGA における通信オーバーヘッドは GPU と比べると小さく， 複数の FPGA で計算を行う際の全体性能は容易に GPU の性能を超えられると考えており，本研究の next step として，ネットワークを用いた複数の FPGA を用いる並列計算を行う予定である．

現時点で検討中の ART on FPGA の並列化手法の概要を**図 8** に示す．ART 法を並列化する場合，ネットワークを通じて通信する必要があるのはレイのデータだけであり，メッシュのデータは移動しない．現実装の時点で，複数の PE と BE 間を Channel で接続し，並列計算を行っており，複数 FPGA に計算を拡張する場合は，Channel による FPGA 内のネットワークを FPGA 間に拡張すれば良い．したがって，図 8 にあるように，BE に到達したレイを必要に応じて隣接する FPGA にネットワーク経由で転送すれば計算できる．

6.2 CoE: Channel over Ethernet

我々はこれまでの研究 [2], [3] で，OpenCL から通信機構を操作できることを示したものの，その実装は基本的なもので，Ethernet のパケット生成といった低レイヤーな処理も OpenCL レベルで記述し動作していた．これを発展さ

せ，FPGA チップ内の通信に用いる Channel を外部ネットワークを通して FPGA 間に拡張するというコンセプトを実現するために，Channel over Ethernet (CoE) システムの開発を行なっている．CoE は Channel に書き込んだデータを Ethernet プロトコルで通信し，異なる FPGA で Channel から読み出せるようにするものである．

図 9 に送信コード例を，**図 10** に受信コードの例を示す．Ethernet はパケット通信型のネットワークであり，Channel のストリーミング型通信とは性質が異なるため，プロトコル形式の変換が必要になる．その変換は OpenCL ではなく Verilog HDL によって実装されており，OpenCL Board Support Package (BSP) に組み込まれる．送信側はバッファリングおよびタイムアウトによってパケットを生成し，受信側はバッファからデータを Channel に供給する．データ送信時にどの FPGA のどの Channel に送信するかは，ホストから PCIe 経由でアドレスを書き込むことで指定し，受信側では各 Channel には ID が割り振られており，宛先 MAC アドレスだけでなく，Channel ID も組み合わせて宛先を決定する．

今後の研究として，CoE システムの機能発展および CoE を ART on FPGA に適用しようと考えている．しかしながら，現在の CoE では ART に適用し実運用に用いるには機能が不足しており，特にフロー制御とパケット消失時の再送制御がない点が問題である．Ethernet のプロトコルにはフロー制御の概念がなく，フロー制御が必要な場合は Transmission Control Protocol (TCP) などを上位プロトコルに用いてフロー制御を行うことが一般的である．TCP プロトコルはインターネットなど広域網で利用するために設計されており，クラスタシステムにおける FPGA 通信で用いるには複雑である．複雑なプロトコルを用いると，レイテンシやスループットの性能面だけでなく，回路リソースの使用量からもオーバーヘッドが大きい．そこで，IEEE 802.3x で定義されている PAUSE フレームの利用を検討している．PAUSE フレームは Ethernet に対する拡張の 1 つであり，バックプレッシャー機能を実現するものである．受信側でバッファ溢れの可能性が発生した場合に特別なパケットを用いて送信側に状態を通知し，そして，PAUSE フレームを受け取った送信側は送信を一時的に停止することで通信路の輻輳を回避する．

7. おわりに

本研究では初期宇宙における輻射輸送を解く ARGOT プログラムで用いられている ART 法を Intel FPGA SDK for OpenCL を用いて FPGA 向けに最適化および CPU, GPU, FPGA 間での性能比較を行なった．これまでの実装では FPGA 内の BRAM に収まる大きさの問題しか解けず，ARGOT で実際に計算したい問題サイズに対応できなかったが，大容量の DDR メモリを併用することで実用的な問題サイズを FPGA で解けるようになった．

```
channel float tx0 __attribute__((depth(0)))
    __attribute__((io("tx0")));
__kernel void f(
    __global float const* restrict x, uint n) {
    for (uint i = 0; i < n; i++) {
        float value = x[i];
        write_channel_intel(tx0, value);
    }
}
```

図 9: CoE の送信 Channel を用いるコード例.

```
channel float rx0 __attribute__((depth(0)))
    __attribute__((io("rx0")));
__kernel void g(
    __global float* restrict x, uint n) {
    for (uint i = 0; i < n; i++) {
        float value = read_channel_intel(rx0);
        x[i] = 10.0f * value;
    }
}
```

図 10: CoE の受信 Channel を用いるコード例.

FPGA 実装の性能は NVIDIA P100 GPU と同程度であるものの，FPGA 実装はまだ最適化の余地があり，最も重要な最適化項目は，FPGA の回路リソース使用量の最適化である．1 PE あたりのリソースが減少すれば，空いたリソースにさらなる PE を FPGA 内部に実装でき，結果として性能が向上する．現在の実装でボトルネックなリソースは ALM と Register であるが，それらの要素はループ制御や状態保持などユーザが OpenCL レベルのコードに直接的な記述をしていない用途でも利用されるため，最適化が困難である．

次世代 Intel FPGA の Stratix10 の利用も今後の課題の 1 つとなる．Arria10 と比較して，動作周波数が向上するだけでなく，2.2 倍の ALM，3.8 倍の DSP を持ち，少なくとも倍の規模の回路を FPGA 内に実装できることが期待される．また，最大で 100Gbps × 4 ポートの通信機構を持つため，通信性能も大幅な向上が期待される．

FPGA 実装の性能は GPU と同程度であるが，GPU は CPU のスレーブデバイスであり能動的に通信を行えない．通信性能においては，自ら通信機構を操作できる FPGA における通信オーバーヘッドは GPU と比べると小さく，並列計算を行う際の性能は GPU の性能を超えられると考えられる．FPGA 内の並列化に用いている Channel 通信を外部ネットワークに拡張するというコンセプトで CoE 通信システムの開発を行なっており，今後，CoE を ART on FPGA に適用する予定である．

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」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] 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).

[2] 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).

[3] Ryohei, K., Yuma, O., Norihisa, F., Yoshiki, Y. and Taisuke, B.: OpenCL-ready High Speed FPGA Network for Reconfigurable High Performance Computing, *HPC Asia 2018*, pp. 1–10 (2018).

[4] 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，情報処理学会研究報告, 2017-HPC-161 (2017).

[5] 横野智也，藤田典久，山口佳樹，大畠佑真，小林諒平，朴　泰祐，吉川耕司，安部牧人，梅村雅之：宇宙輻射輸送計算における HDL 設計と OpenCL 設計の比較，情報処理学会研究報告, 2018-HPC-163 (2018).

[6] Zohouri, H. R., Maruyama, N., Smith, A., Matsuda, M. and Matsuoka, S.: Evaluating and Optimizing OpenCL Kernels for High Performance Computing with FPGAs, *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '16, Piscataway, NJ, USA, IEEE Press, pp. 35:1–35:12 (online), available from ⟨http://dl.acm.org/citation.cfm?id=3014904.3014951⟩ (2016).

[7] Hill, K., Craciun, S., George, A. and Lam, H.: Comparative analysis of OpenCL vs. HDL with image-processing kernels on Stratix-V FPGA, *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 189–193 (online), DOI: 10.1109/ASAP.2015.7245733 (2015).

[8] Luo, Y., Wen, X., Yoshii, K., Ogrenci-Memik, S., Memik, G., Finkel, H. and Cappello, F.: Evaluating irregular memory access on OpenCL FPGA platforms: A case study with XSBench, *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–4 (online), DOI: 10.23919/FPL.2017.8056827 (2017).

[9] 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，情報処理学会研究報告, 2016-HPC-153 (2016).

[10] 塙　敏博，伊田明弘，大島聡史，河合直聡：FPGA を用いた階層型行列ベクトル積，情報処理学会研究報告, 2016-HPC-155 (2016).

[11] 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).

[12] 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, p. 62 (online), DOI: 10.1093/pasj/psv027 (2015).

[13] Górski, 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, p. 759 (online), available from ⟨http://stacks.iop.org/0004-637X/622/i=2/a=759⟩ (2005).

[14] NVIDIA Corporation: GPUDirect for RDMA, `https://docs.nvidia.com/cuda/gpudirect-rdma/index.html`, (online), available from ⟨`https://docs.nvidia.com/cuda/gpudirect-rdma/index.html`⟩.