Vol.2020-HPC-175 No.8
2020/7/30

Stratix 10 FPGAを用いた ray-tracing法による輻射輸送計算の高速化

藤田 典久[1,2] 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[1] 梅村 雅之[1,3]

概要：我々はこれまでの研究で，宇宙輻射輸送問題で用いられる Authentic Radiative Transfer (ART) 法を Arria 10 FPGA 上に実装し性能評価を行ってきた．本稿では，ART 法を最新の Intel Field Programmable Gate Array (FPGA) である Stratix 10 FPGA 向けに最適化し，性能評価を行う．また，我々が提唱している FPGA 間通信フレームワークである Communication Integrated Reconfigurable CompUting System (CIRCUS) を用いて並列計算を実現し，複数 FPGA を用いる際の性能評価も行う．

1. はじめに

近年，高性能計算の分野で Field Programmable Gate Array (FPGA) が注目されている．従来 FPGA 開発を行う際は，ハードウェア記述言語 (Hardware Description Language, HDL) を用いてプログラムを記述しなければならず，開発コストが高いという問題があった．しかしながら，開発コストの問題は高位合成 (High Level Synthesis, HLS) 開発環境の普及によって解決されつつある．

FPGA は高速シリアル通信を行えるトランシーバーを有しているものがあり，最近の FPGA ボードでは 100Gbps×4 の通信性能を有する．この通信機構は，FPGA の内部演算回路と直接接続されており，FPGA 内部に実装した演算回路から通信を直接行えるため，低オーバーヘッドな通信が可能となる．我々は，FPGA が持つ高い通信性能と演算性能を組み合わせることで，HPC アプリケーションを加速することが出来ると考えている．HPC で広く用いられているアクセラレータに Graphics Processing Unit (GPU) がある．NVIDIA GPU は GPUDirect for RDMA[1] (GDR) と呼ばれる技術をサポートしており，Network Interface Card (NIC) が GPU メモリに対して PCI Express (PCIe) バスを通じて直接アクセスができる．しかしながら，GDR を用いたとしても GPU メモリへのアクセスが直接できるのみであり，通信に関わる制御系は CPU 上で行われる．したがって，CPU-GPU 間の同期や PCIe バスに由来するオーバーヘッドは依然として残っている．

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 筑波大学 システム情報系

本研究の目的は，FPGA を用いた並列システムを構築し，ray-tracing 法による輻射輸送計算の高速化することである．アプリケーションの記述には OpenCL で記述できる高位合成処理系を用いる．また，並列計算に必要な通信も OpenCL を用いて記述し，高位合成で科学技術計算アプリケーションを FPGA 上で実行でき，それに加えて，OpenCL 上で通信を記述し複数の FPGA を用いる並列計算が実現できることを示す．

本稿の貢献は以下の通りである．

- 輻射輸送計算を Stratix 10 FPGA 向けに最適化を行う．
- OpenCL と Verilog HDL の混合記述を行うことで，OpenCL が持つプログラマビリティと Verilog HDL が持つ細粒度な最適化を組み合わせられることを示す．
- 複数の FPGA を用いた並列輻射輸送計算を OpenCL で記述できることを示す．
- FPGA 実装と GPU 実装を比較し，FPGA 実装が高い性能を発揮できることを示す．

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．[2] 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．[3] 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．[4] では，XSBench を用いてイレギュラーなメモリア

© 2020 Information Processing Society of Japan

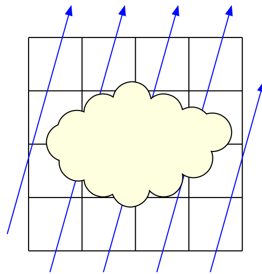

図 1: ART 法で用いられているレイトレーシングの概念図．青い矢印はレイを表し，黄色の雲は反応を計算するガスを表す．

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，[5] や [6] 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．

複数の FPGA をネットワークで接続して利用する研究はいくつか知られている．[7] では，データセンター内に 6×8 の 2D トーラスネットワークを持つ FPGA クラスタを構築し，Bing Web 検索エンジンのアクセラレータとして用いている．Paderborn University の Paderborn Center for Parallel Computing は Noctua supercomputer [8] を運用している．そのうちの 16 ノードは FPGA を有しており，ノードあたり Intel Stratix 10 FPGA を 2 枚持つ．Tiziano らは，OpenCL から FPGA 間通信を可能とする通信フレームワーク Streaming Message Interface (SMI) [9] を Noctua 上で開発し評価を行った．しかしながら，性能評価はマイクロベンチマークにとどまっており，実際の科学技術計算のアプリケーションを用いたものではない．

本章で述べたように，FPGA 上の高位合成に関する研究や，FPGA ネットワークに関する研究は盛んに行われているが，それらを組み合わせた研究は十分なされているとは言えない．本研究の独自性は，これら 2 つの要素を組み合わせて，科学技術アプリケーションを FPGA 上で並列計算を行う点にある．

3. ART 法

筑波大学 計算科学研究センターでは，宇宙輻射輸送シミュレーションコード ARGOT (Accelerated Radiative transfer on grids using Oct-Tree) の開発を行っている．ARGOT は輻射輸送問題を解く際に 2 つにアルゴリズムを組み合せており，点光源からの輻射輸送を ARGOT アルゴリズム [10]，空間的に広がった光源からの輻射輸送を ART (Authentic Radiation Transfer) アルゴリズム [11] を用いて解く．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 FPGA 向けに最適化を行うため，本章では ART 法についてのみ述べる．

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．

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

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 [12] 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．振動数 ν 毎に 1 回の指数関数の計算が必要であり，振動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

© 2020 Information Processing Society of Japan

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．

4. ART on FPGA

4.1 これまでの研究との違い

我々はこれまでの研究で，ART 法の FPGA 向け最適化を行い，FPGA の性能は小さい問題サイズでは GPU を凌駕し，大きい問題サイズであっても GPU と同等であることを示した [13], [14], [15]．また，ART on FPGA の実装を ARGOT プログラムに組み込み，ART 法を FPGA・ARGOT 法を GPU で計算し，複数のアクセラレータを用いた輻射輸送計算が実現できることを示した [16], [17], [18], [19]．

これらの研究は，一つ前の世代の FPGA である Intel Arria 10 FPGA を用いていたため，より新しい FPGA での評価が求められていた．そこで，本稿では，最新の Intel FPGA である Stratix 10 FPGA を用いて，ART 法に FPGA 向け最適化を行い性能評価を行う．また，複数の FPGA を用いた並列計算の実装を行い性能評価を行う．

4.2 Intel FPGA SDK for OpenCL

本研究では，FPGA 回路の実装に Intel FPGA SDK for OpenCL を用いる．Intel FPGA SDK for OpenCL は高位合成 (High Level Systhesis, HLS) の処理系であり，OpenCL コードから FPGA で動作するハードウェアを生成できる．本 SDK は，All-in-One 型の総合開発環境であり，FPGA 向け OpenCL コンパイラだけでなく，ホストで動作する OpenCL ランタイムライブラリおよび FPGA PCIe ドライバを含み，この SDK だけでハードウェア開発に加えてホストから PCIe 経由で FPGA を制御する動作環境を構築できる．

OpenCL をサポートする FPGA ボードは数多くあるが，それらのハードウェア仕様（FPGA チップの型番，メモリチップの仕様など）はそれぞれ異なる．したがって，この SDK では，FPGA ボードに固有な情報を Board Support Package (BSP) としてコンパイラに与えることで，ボード毎の差異を吸収し，同じ OpenCL コードを異なるボード上で実行できる仕組みを持つ．

OpenCL コンパイラは，FPGA に特化した言語拡張を有しており，その中の一つに “Channel” と呼ばれる仕組みがある．Channel はカーネル間でデータを直接やり取りできる仕組みである．Channel の実態は，FPGA 内部メモリを用いた First-In-First-Out (FIFO) バッファであり，OpenCL の組み込み関数 (read_channel_intel, write_channel_intel) を用いて値を読み書きできる．Channel を使うことで，ユーザはモジュール化された設計を実現できる．FPGA 内に複数のカーネルを実装し，それらを Channel を通じて接続することで，空間並列性を明示的に記述できる．

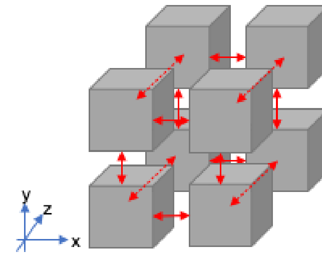

図 2: 2 × 2 × 2 構成の Processing Element (PE) および PE 間接続図．ただし，立方体は PE を，赤矢印は Channel による接続を表す．

4.3 Channel を用いた空間並列

ART 法の FPGA 実装は，これまでの研究 [13], [14], [15] で Arria 10 向けに実装した構造をベースに，演算カーネル内部を Stratix 10 向けに最適化したものである．**図 2** にあるように Processing Element(PE) を $2 \times 2 \times 2 = 8$ 個実装し，それらを Channel で接続する構造を持つ．Arria 10 で用いていた実装では，PE と Boundary Element (BE) から演算カーネルは構築されていたが，Stratix 10 実装では，PE/BE の区別を廃止し，BE が有していた機能を PE に含めることとした．BE の機能を PE に含めても性能面でのペナルティはなく，そうであればカーネル数が少ない方が保守性が高いと考えたからである．

ART 法を FPGA に実装する際の考え方は，前述した実装とほぼ同じであるため，本稿では主に Stratix 10 向けに変更した箇所について述べる．それ以外の実装に関する詳細は前述した研究会報告および論文を参照していただきたい．

PE は OpenCL カーネルとして実装されており，それぞれの PE は並列動作して，演算を行う．そして，各 PE に，Block RAM (BRAM) を用いて実装された演算用のワーキングメモリがあるという構造は，Arria 10 実装と同じである．しかしながら，Arria 10 実装では，8^3 の空間を保持するメモリを各 PE に実装していたところを，Stratix 10 実装では 16^3 の空間を保持できようにメモリサイズを増量している．これは，Stratix 10 は Arria 10 よりも多くの BRAM を持つことに対応するためである．

また，本稿で実装した ART 法は，ベースとする CPU 実装のコードを最新のバージョンに変更した．そのため，振動数 ν の値が変更されている．Arria 10 向け実装においては，$\nu = 3$ という想定で実装していたが，本稿の実装では $\nu = 6$ になっている．そのため，ART 法で光学的厚みを計算する箇所の演算量が 2 倍に増えていることに留意していただきたい．

4.4 HDL を用いた reduction の最適化

ART 法では，メッシュ上をレイが通過するたびに，メッシュとレイの相互作用を計算し，光学的厚み（Intensity）

© 2020 Information Processing Society of Japan

```
ray = ray_init();
while (ray is alive) {
  pos = ray.ix * Ny * Nz + ray.iy * Nz + ray.iz;
  mesh[pos].I_nu1 +=
    ray->I_in_nu1 * -expm1f(-tau);
  ray = next_mesh(ray);
}
```

図 3: メッシュにおける reduction 演算の例.

を求め，その結果を各メッシュ上で積分を行う．プログラムの実装としては，**図 3** にあるように，メモリに対する加算として表現される．このループをパイプラインで実装すると，配列 `mesh` に対する加算が loop carreied dependency となり，Initiation Interval (II) が 1 より大きくなってしまう．II はパイプラインを何クロック毎に動作させるかを示す値である．II=1 が理想の状態であり，II が 1 より大きくなるとスループットが低下する．例えば II=2 の場合，2 クロックに 1 回しか計算が実行されないため，スループットが 50%に低下する．II=1 を達成するためには，ループの iteration 間で依存関係がある計算を 1 クロックで求めなければならない．しかしながら，依存関係がある計算が複雑な場合，クリティカルパスとなり，回路の動作周波数が低下してしまう．そのため，OpenCL コンパイラは，II を 1 より大きくすることで動作周波数を維持する動作を行う．ART on FPGA 実装では，配列 `mesh` は BRAM 上に実装されているが，BRAM へのアクセスと浮動小数点数加算を 1 クロックで行うことは難しい．

Arria 10 を用いた ART on FPGA の実装 [19] では，メモリアクセスを Shift Register で実装し，演算を Verilog HDL で記述することで，演算を行うループが II=1 になるように最適化を行った．本来，OpenCL コンパイラでは，ループの II を指定して強制できない[*1]のであるが，演算の一部を Verilog HDL で記述し，そのパイプラインの必要クロックサイクル数を調整することで，コンパイラによって II=1 のループが作成されるように調整を行った．しかしながら，この手法を用いる場合，演算部を何クロックサイクル以内で実装すれば II が 1 になるのかが定量的に判断できず，Try-and-Error によって II=1 を達成していた．利用する FPGA チップだけでなく，コンパイラのバージョンによっても変わる可能性があるためである．また，ART 法に固有な演算を Verilog HDL で記述していたため，他のアプリケーションに最適化をそのまま適用することはできなかった．

本稿では，「加算器付きメモリ」を Verilog HDL で記述し，それを OpenCL コードから呼び出すことで，演算ループの II=1 を達成した．加算器付きメモリの実装には，Stratix 10 の Digital Signal Processor (DSP) ブロックが持

*1 ただし，II を緩和する場合は，`#pragma II` を用いることで可能．

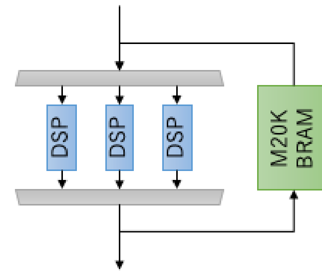

図 4: 加算器付きメモリの概要.

```
float accram(
  ushort index, uchar set, float x, float y);
```

図 5: accram 関数の prototype 宣言.

```
float accram(
  ushort index, uchar set, float x, float y) {
  float newValue =
    (set) ? (x * y) : (bram[index] + x * y);
  bram[index] = newValue;
  return newValue;
}
```

図 6: accram 関数の動作を示す擬似コード.

つ Accumulator モードを用いる．Accumulator モードは，計算結果を DSP ブロック内のレジスタで保持することで，1 クロック毎に値の足し合わせができるモードである．

図 4 にあるように，DSP を 3 つとデータを保持する M20K BRAM を組み合わせて実装を行う．基本的な動作は，インデックスで指定されたデータを BRAM から読み出し，DSP で加算を行い，結果を BRAM に書き戻すものである．DSP のレイテンシが 2 サイクルと，それに加えて動作周波数を高めるために DSP の出力にレジスタを 1 段挿入しており，合計で 3 クロックの演算レイテンシがある．そのため，DSP を 3 つ実装し，3 クロック以内に同じインデックスへのアクセスがある場合は，メモリから読みだしたデータを利用せず，DSP 内部のレジスタに保持されているデータを用いる．これによって，連続した同じインデックスに対する加算が発生しても，正しく演算を行える．計算方法だけを考えれば，この手法は OpenCL でも記述可能であるように見えるが，OpenCL のみでこれを実装することは難しい．正しい結果を得るためには，BRAM へのアクセス，DSP を用いた演算，スケジュールを行う回路の所要クロックサイクル数が設計時にわかっていなければならないが，それを明示的に制御する手段がないためである．

Verilog HDL で実装した加算器付きメモリは，OpenCL からは関数として呼び出すことで利用できる．関数の宣言を**図 5** に示す．また，この関数の動作を表す擬似コードを**図 6** に示す．単純に値を加算するのではなく，積を加算し

© 2020 Information Processing Society of Japan

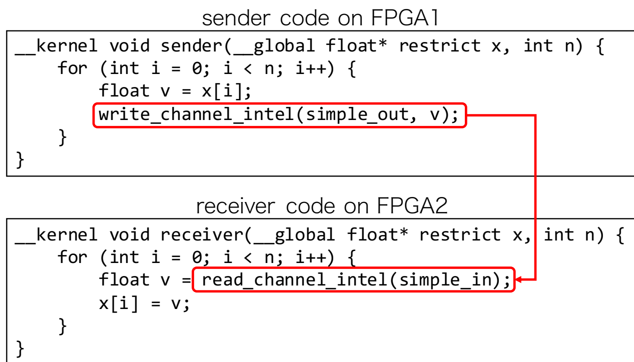

図 7: CIRCUS の通信コード例.

ているのは，前述した Accumulator モードの DSP ブロックは必ず 2 入力を取り積＋加算の形でしか利用できない制限があるためである．BRAM 上で加算を行うという汎用性のある演算を Verilog HDL でライブラリ化を行ったため，この実装には他のアプリケーションにも適用できる汎用性があると考えている．

4.5 Parallelized ART on FPGA

FPGA 間通信が可能なポートを有する FPGA ボードは数多く存在するが，OpenCL 環境からそれらの外部ポートを制御できることは一般的ではない．我々は，高速な外部通信機構を OpenCL から扱えることを既に示しており [20], [21], [22], [23]，これらを元に OpenCL を用いた FPGA 間通信フレームワークとして Communication Integrated Reconfigurable CompUting System (CIRCUS) システムを提唱している [24].

CIRCUS は Channel の接続を FPGA 間に拡張するというコンセプトで通信を実現する．CIRCUS を用いることで，**図 7** にあるように，異なるノードにある FPGA 間で Channel 通信が可能となる．CIRCUS を用いる際の開発フローを**図 8** に示す．CIRCUS の通信に用いるカーネルは，Generator プログラムを用いて作成する．

通信に用いる Channel (以下，CIRCUS Channel と呼ぶ) の要求（名前・データ型・数量など）はアプリケーション毎に異なる．したがって，アプリケーションに必要な通信構造を XML ファイルに定義し，それを Generator が読み取り，OpenCL コードを生成する．最終的には，アプリケーションの OpenCL コードから，自動生成されたファイルを `#include` し，1 つの FPGA 回路を作成する．

本稿では，ART on FPGA の実装に CIRCUS を用いた FPGA 間通信を実装し，並列計算が可能であることを示す．ART 法の実装は，前章で述べたように，複数の PE を Channel で接続した構成をしている．したがって，**図 9** 似あるように，PE 間の Channel 接続を CIRCUS を用いて異なる FPGA 間に拡張することで並列計算を実現する．言い換えると，複数の FPGA を組み合わせることで，巨大な PE Array を構築する．

ただし，現時点の実装は X 次元方向のみ CIRCUS 通信に対応する．今後，Y,Z 次元にも並列化可能なように開発を行う予定である．また，これまでの研究で動作実績のあった 512bit 幅の CIRCUS Channel で通信を記述している．312bit (=39byte) の大きさを持つ構造体に 200bit のゼロパディングを挿入することで 512bit の大きさの構造体を定義し，それを通信している．本稿では並列計算が可能なことを示すことが目的の一つであるため，通信の動作安定性を重視した設計・実装をとったが，今後通信の最適化を実施する．

図 10 に PE のメインループの擬似コードを示す．PE の計算は，レイ入力，Intensity 計算，レイ出力の 3 つのパートから構成される．ここで，レイ入出力判定はレイの座標から，どの次元に対してアクセスを行うかを求める．OpenCL で，PE Array のような，ほぼ同一な複数のカーネルを実装する簡便な方法はなく，コードを重複して記述する必要がある．そのため，実際のコードは C 言語の Preprocessor を用いて PE を複製しており，PE がアクセスする Channel は Preprocessor で定義してコンパイル時に置換される．図 10 では，$(x, y, z) = (0, 0, 0)$ の PE のコード例を示しているため，x– (x_neg) の方向へのアクセスは CIRCUS 経由で行い，それ以外の次元は内部 Channel へのアクセスとなる．

4.6 制限事項

Stratix 10 向け ART on FPGA は，開発中であるため，いくつかの制限事項がある．

- 各 PE が持つ計算用メモリ（BRAM）に収まる問題しか扱えない
- レイとメッシュが交わる距離が短い場合，計算が停止する可能性がある
- X 次元の方向しか CIRCUS を用いた領域分割ができない

Arria 10 を用いた ART on FPGA の実装 [19] では，計算の進捗にあわせて BRAM と DDR メモリの間でデータを入れ替えることで，BRAM サイズに収まらない大きな問題を扱えた．Stratix 10 向け実装では，CIRCUS を用いた並列計算の実現を優先したため，BRAM と DDR メモリを組み合わせる機能が実装できていない．計算データは開始時に DDR メモリから BRAM にロードし，完了後に DDR メモリに書き戻すのみである．ART 法は直行格子上で平行光を扱うため，隣り合う並行レイ同士は，レイとメッシュが交わる距離は同じであり（図 1 を参照），進行する次元（X, Y or Z）も同じである．ところが，レイとメッシュが交わる距離が短い場合，演算誤差によって，あるレイは X 次元方向に進むが，あるレイは Y 次元に進むという可能性がある．この条件が満たされると，FPGA 内の計算が破綻

© 2020 Information Processing Society of Japan

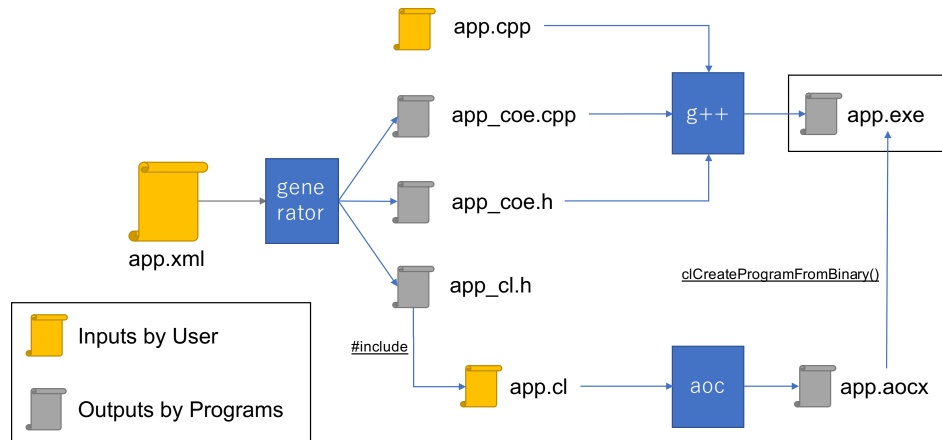

図 8: CIRCUS を用いる際の開発フロー.

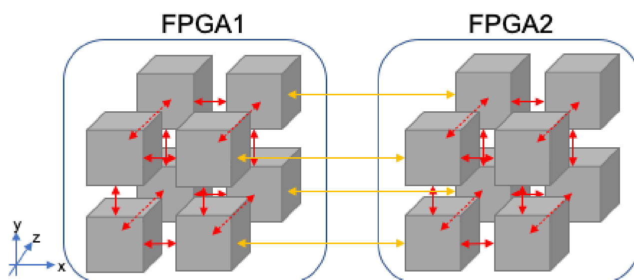

図 9: CIRCUS を用いた PE の拡張．赤矢印は Channel を，黄矢印は CIRCUS Channel を表す．

し，カーネルの実行が完了しなくなってしまう．

また，以上の制限に加えて，以下に示すような CIRCUS の通信機能に由来する制限も存在する．

- フロー制御が未実装
- エラー処理（再送・訂正）が未実装

フロー制御がないことは並列計算を行う上で大きな障壁となる．本稿で行う実験の範囲では，用いる FPGA の数が少ないため，通信路内部にあるバッフがが溢れず問題なく通信できている．しかしながら，多数の FPGA を用いる場合，FPGA をごとの計算進捗の差が大きくなりバッファが溢れ通信が破綻する可能性がある．また，Virtual Channel が実装されていないため，通信路に循環があるとデッドロックが発生しうるが，ART 法においては，通信パターンに循環が現れないため問題にならない．

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中の Cygnus スーパーコンピュータ（**表 1**）と，同センターで運用している実験クラスタ Pre-PACS-X (PPX)（**表 2**）を性能評価に用いる．Cygnus はマルチ・ヘテロジニアスなシステムであり 80 ノードから構成される．そのうち，32 ノードは Albireo ノードと呼ばれ，CPU + GPU + FPGA ノードである．PPX は実験用システムなため，様々な仕様のノードが混在しているが，PPX システムの Intel Stratix 10 FPGA ボードを搭載する 4 ノードから，最大 2 ノードを用いて性能評価を行う．4 枚の FPGA ボードは 100Gbps 通信が行える光ケーブルで接続されており，2x2 の 2D トーラスネットワークを構成している．

表 1: 評価環境 (Cygnus)

CPU	Intel Xeon Gold 6126 × 2
CPU Memory	DDR4 192GB (96GB / CPU)
Infiniband	Mellanox ConnectX-6 HDR100 ×4
Host OS	CentOS 7.6
Host Compiler	gcc 4.8.5
OpenCL SDK	Intel FPGA SDK for OpenCL 19.4.0.64
FPGA	Bittware 520N (1SG280HN2F43E2VG)
FPGA Memory	DDR4 2400MHz 32GB (8GB × 4)
Comm. Port	QSFP28 × 4
GPU	NVIDIA V100 × 4
CUDA	10.1.2

表 2: 評価環境 (PPX)

CPU	Intel Xeon E5-2690 v4 × 2
CPU Memory	DDR4 2400 MHz 64 GB (8 GB × 8)
Infiniband	Mellanox ConnectX-4 EDR
Host OS	CentOS 7.3
Host Compiler	gcc 4.8.5
OpenCL SDK	Intel FPGA SDK for OpenCL 19.4.0.64
FPGA	Bittware 520N (1SG280HN2F43E2VG)
FPGA Memory	DDR4 2400MHz 32GB (8GB × 4)
Comm. Port	QSFP28 × 4

© 2020 Information Processing Society of Japan

```
channel rt_ch[6][NPE_X][NPE_Y][NPE_Z];
channel extout_x_neg_x0_y0_z0;
channel extin_x_neg_x0_y0_z0;
#define PE_INPUT_X_POS  rt_ch[1][1][0][0]
#define PE_INPUT_X_NEG  extin_x_neg_x0_y0_z0
#define PE_INPUT_Y_POS  rt_ch[3][0][1][0]
#define PE_INPUT_Y_NEG  rt_ch[2][0][1][0]
#define PE_INPUT_Z_POS  rt_ch[5][0][0][1]
#define PE_INPUT_Z_NEG  rt_ch[4][0][0][1]
#define PE_OUTPUT_X_POS rt_ch[0][0][0][0]
#define PE_OUTPUT_X_NEG extout_x_neg_x0_y0_z0
#define PE_OUTPUT_Y_POS rt_ch[2][0][0][0]
#define PE_OUTPUT_Y_NEG rt_ch[3][0][0][0]
#define PE_OUTPUT_Z_POS rt_ch[4][0][0][0]
#define PE_OUTPUT_Z_NEG rt_ch[5][0][0][0]

__kernel void PE_x0_y0_z0(...) {
 while (!exit) {
  bool x_neg, x_pos, y_neg, y_pos,
       z_neg, z_pos = ...;
  if (x_neg)
   rt = read_channel_intel(PE_INPUT_X_NEG);
  else if (x_pos)
   rt = read_channel_intel(PE_INPUT_X_POS);
  else if (y_neg)
   rt = read_channel_intel(PE_INPUT_Y_NEG);
  else if (y_pos)
   rt = read_channel_intel(PE_INPUT_Y_POS);
  else if (z_neg)
   rt = read_channel_intel(PE_INPUT_Z_NEG);
  else if (z_pos)
   rt = read_channel_intel(PE_INPUT_Z_POS);
  ...
  calc_intensity(...);
  ...
  bool x_neg_out, x_pos_out, y_neg_out, y_pos_out,
       z_neg_out, z_pos_out = ...;
  if (x_neg_out)
   write_channel_intel(PE_OUTPUT_X_NEG, rt);
  else if (x_pos_out)
   write_channel_intel(PE_OUTPUT_X_POS, rt);
  else if (y_neg_out)
   write_channel_intel(PE_OUTPUT_Y_NEG, rt);
  else if (y_pos_out)
   write_channel_intel(PE_OUTPUT_Y_POS, rt);
  else if (z_neg_out)
   write_channel_intel(PE_OUTPUT_Z_NEG, rt);
  else if (z_pos_out)
   write_channel_intel(PE_OUTPUT_Z_POS, rt);
 }
}
```

図 10: PE の擬似コード．

2 システムの使い分けについては，GPU の性能評価には Cygnus が持つ NVIDIA V100 を，FPGA の性能評価には PPX が持つ Bittware 520N を用いる．Cygnus と PPX は同じ FPGA ボードを搭載しているにもかかわらず FPGA の性能評価に用いない理由は，CIRCUS の通信を利用するためである．CIRCUS は通信機能を含んだ改造 BSP を用いるため，FPGA の書き換えおよび計算ノードの再起動が必要となる．しかしながら，Cygnus は本番運用に供されているシステムであるため，ノードの再起動に制限を伴う．したがって，FPGA の性能評価には PPX を用いる．同一 FPGA ボードを用いていることと，計算を行うのはアクセラレータのみであり CPU は FPGA の制御に専念するため，Cygnus 上で実行する場合と性能に大きな違いはないと考えられる．

表 3: FPGA 実装のリソース消費量および動作周波数．

ALM	Reg.	M20K	MLAB	Fmax[MHz]
617,423 (66.2%)	1,285,365 (34.4%)	5,053 (43.1%)	1,916 (33.3%)	261.1

表 4: Seed を変化させた際の動作周波数の違い．

Seed	1	2	3	4	5
Fmax[MHz]	225.0	261.0	240.0	261.1	245.0

表 5: 測定結果（詳細）．それぞれの数字の単位は ms.

	GPU					FPGA
	Comp.	H2D	D2H	Others	Total	Total
1 Node	135.7	28.0	41.6	21.1	226.4	24.1
2 Node	135.9	26.0	44.0	60.6	266.5	25.2

5.2 FPGA リソース消費量, 動作周波数

FPGA 実装のリソース消費量と OpenCL カーネルの動作周波数を表 3 に示す．ここで，ALM (Adaptive Logic Module) は Lookup Table (LUT) と Register から構成される論理回路ブロック，M20K は FPGA 内蔵 BRAM ブロック，DSP (Digital Signal Processor) は浮動小数点数演算や整数乗算を行うブロックである．FPGA 全体の 66%のリソースを消費し，動作周波数は 261MHz である．

FPGA 版の性能評価には，Fitter が扱う乱数の Seed[*2] を 5 通り試したもので最も動作周波数が高いものを採用する．Seed を変化させた際に，どのように動作周波数が変化するかを表 4 に示す．ただし，1 回のコンパイルあたり 8〜9 時間の時間がかかるため，本稿では 5 通りのみしか試験できていない．表 4 からわかるように，最速の結果と最遅の結果を比べると 16%の差があるため，OpenCL を用いて FPGA プログラムを記述する際は複数の Seed で合成を試してみる事が重要であるといえる．

5.3 性能評価

性能評価には ARGOT プログラムから ART 法に関する部分を抜き出して制作したベンチマークプログラムを用いる．ART 法の計算のみを行うものであるため，問題の初期

[*2] `aoc -seed=X` で指定できる．

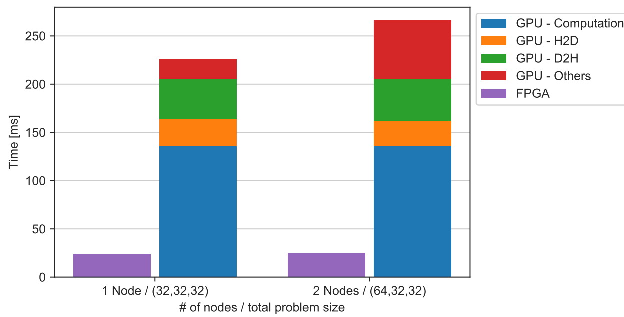

図 11: 測定結果.

値は一様乱数で生成する．Cygnus の GPU を 1 ノードあたり 1GPU 最大 2 ノード，PPX の FPGA を 1 ノードあたり 1FPGA 最大 2 ノードの条件で測定を行う．ただし，4.6 節で述べたように，FPGA 実装は扱える問題サイズが固定であるという制限がある．そのため，FPGA・GPU 1 台あたり $(32, 32, 32)$ の問題サイズで固定して，weak scaling の条件で ART 法の 1 ステップの計算時間を測定する．

測定結果を**図 11** と**表 5** に示す．GPU 実装は cudaEvent と `gettimeofday` 関数を用いて，Device to Host (D2H) 転送，Host to Device (H2D) 転送，計算，その他の Breakdown を計測する．GPU 実装は，MPI 通信をホスト側から制御しているため，計算ループ毎に D2H, H2D 転送が現れる．MPI 通信が不要な時（1 ノード）であっても，これらの転送は実施される実装になっていることに注意していただきたい．ただし，D2H 転送はレイデータの初期値をホスト側から与えているため，1 ノード時であっても必須である．なお，ノード間の MPI 通信は，Others に含まれる．

一方で，FPGA 実装は通信を含めすべて FPGA で完結しているため，また，パイプライン化によって区別した時間測定が難しいため，Breakdown は示せない．ホスト OS 上で，OpenCL カーネルの起動 (`clEnqueueNDRangeKernel`) から完了 (`clFinish`) までの時間を `gettimeofday` 関数を用いて測定を行う．なお，どちらのデバイスの測定も，初期値の転送および計算結果の転送は時間に含めていない．

5.4 考察

1 ノードの計算時間を比較すると，FPGA 版が 9.4 倍高速という結果が得られた．GPU 版のコードが 1 ノード実行時に最適化されいないことを考慮して，GPU の計算時間のみを FPGA 版の実行時間と比較したとしても，FPGA 版が 5.6 倍高速である．2 ノードの実行時間で比較すると，1 ノード時よりも性能に差が開き，FPGA 版が 10.6 倍という結果が得られた．今回の実験は Weak Scaling の条件で実行しているため，1 ノード時と 2 ノード時では実行時間はほぼ同じとなることが期待されることに注目すると，1 ノード実行時と 2 ノード実行時の総計算時間の増加分は通信のオーバーヘッド由来であると考えられる．また，GPU 版の Breakdown の結果より，通信以外の部分がほぼ同じであることからも，計算量は想定通りに Weak Scaling していると考えられる．FPGA 版は通信も演算もパイプライン構造で一体化しているため，通信時間のみを抜き出して測定することが困難であるが，計算時間の増加量という観点からは評価可能である．FPGA 版の計算時間の増加は 1.1ms ($\times$1.046)，GPU 版の増加は 40.1ms ($\times$1.178) であり，FPGA 版の方がオーバーヘッドが小さいことがわかる．これは，FPGA では演算回路と通信機構が直接接続されてるため高効率に通信できることにくわえて，CIRCUS が提供するパイプライン通信の効果によって細粒度に通信と演算がオーバーラップしているためと考えられる．

ただし，本稿の評価では，ノードあたり 32^3 という問題サイズで評価していることに注意が必要である．我々のこれまでの研究 [14], [19] から，FPGA はパイプライン並列と空間並列の併用しているため 32^3 の問題サイズでも十分な性能が得られるが，GPU にとっては 1 枚あたり 32^3 という問題サイズは小さく，GPU の性能を十分に引き出せていないと考えられる．また，ART 法ではノードあたりの計算量は $O(N^3)$ であるが通信量は $O(N^2)$ となるため，ノードあたりの問題サイズを増加させると相対的に通信の割合が減少し，GPU-FPGA 間の差が減少すると予想できる．

6. まとめと今後の課題

本稿では，宇宙輻射輸送シミュレーションコード ARGOT の中で重要なアルゴリズムである ART 法を Intel Stratix 10 FPGA 向けに最適化を適用し性能評価を行った．ART 法で用いられる Reduction 演算に対して Verilog HDL で記述する最適化を適用し，計算を行うメインループが II=1 になる実装ができ，これによって演算のスループットが最大化された．1 ノードの計算時間を比較すると，FPGA 版が 9.4 倍高速という結果が得られ，2 ノードの実行時間で比較すると，FPGA 版が 10.6 倍という結果が得られた．また，我々が研究開発している FPGA 間通信フレームワーク CIRCUS を ART 法に適用し，複数の FPGA を用いた並列計算が可能であることを示した．1 ノードを用いた実行時間と 2 ノードを用いた場合の実行時間を比較すると，FPGA 版の方がオーバーヘッドが小さいことが明らかとなった．

今後の課題としては，より大規模な問題を用いた性能評価を行うことが挙げられる．我々がターゲットとしている問題サイズは 1024^3 であり，これは 1 ノードのメモリで扱える問題サイズでは到底ない．Cygnus 環境では，最大で 64 枚の FPGA が扱えるため，それらをすべて用いた並列計算を行いたいと考えている．ただし，現在の FPGA 版の実装では，その条件で計算を行うために必要な機能が不足しているため，不足している機能の実装を進めなければならない．

© 2020 Information Processing Society of Japan

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．本研究成果は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）における 2020 年度課題「FPGA-GPU 混載プラットフォームにおける HPC アプリケーションとシステム・ソフトウェアの開発」を利用して得られたものです．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] NVIDIA Corporation: GPUDirect for RDMA, `https://docs.nvidia.com/cuda/gpudirect-rdma/index.html`.

[2] 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).

[3] 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).

[4] 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).

[5] 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).

[6] 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).

[7] Putnam, A., Caulfield, A., Chung, E., Chiou, D., Constantinides, K., Demme, J., Esmaeilzadeh, H., Fowers, J., Gray, J., Haselman, M., Hauck, S., Heil, S., Hormati, A., Kim, J.-Y., Lanka, S., Peterson, E., Smith, A., Thong, J., Xiao, P. Y., Burger, D., Larus, J., Gopal, G. P. and Pope, S.: A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services, *Proceeding of the 41st Annual International Symposium on Computer Architecture (ISCA)*, IEEE Press, pp. 13–24 (online), available from ⟨https://www.microsoft.com/en-us/research/publication/a-reconfigurable-fabric-for-accelerating-large-scale-datacenter-services/⟩ (2014).

[8] Center for Parallel Computing: PC2 - Noctua (Universität Paderborn), `https://pc2.uni-paderborn.de/hpc-services/available-systems/noctua/`.

[9] De Matteis, T., de Fine Licht, J., Beránek, J. and Hoefler, T.: Streaming Message Interface: High-performance Distributed Memory Programming on Reconfigurable Hardware, *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC ’19, New York, NY, USA, ACM, pp. 82:1–82:33 (online), available from ⟨https://doi.org/10.1145/3295500.3356201⟩ (2019).

[10] 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).

[11] 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).

[12] 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).

[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，情報処理学会研究報告, 2017-HPC-161 (2017).

[14] 藤田典久，小林諒平，山口佳樹，朴 泰祐，吉川耕司，安部牧人，梅村雅之：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，情報処理学会研究報告, 2018-HPC-165 (2018).

[15] 藤田典久，小林諒平，山口佳樹，朴 泰祐，吉川耕司，安部牧人，梅村雅之：宇宙輻射輸送コードにおける OpenCL による FPGA 演算加速最適化，情報処理学会論文誌コンピューティングシステム（ACS），Vol. 12, No. 3, pp. 64–75 (2019).

[16] 小林諒平，藤田典久，山口佳樹，中道安祐未，朴 泰祐：OpenCL 対応 FPGA 間通信機能による GPU・FPGA 複合型演算加速，情報処理学会研究報告, 2019-HPC-170 (2019).

[17] 中道安祐未，藤田典久，小林諒平，朴 泰祐，吉川耕司，梅村雅之：GPU・FPGA 複合演算加速による輻射流体シミュレーションコード ARGOT の実装，情報処理学会研究報告, 2019-HPC-170 (2019).

[18] 小林諒平，藤田典久，中道安祐未，山口佳樹，朴 泰祐，吉川耕司，安部牧人，梅村雅之：OpenCL 対応 GPU・FPGA デバイス間連携機構による宇宙輻射輸送コードの演算加速，情報処理学会研究報告, 2019-HPC-172 (2019).

[19] 小林諒平，藤田典久，中道安祐未，山口佳樹，朴 泰祐，吉川耕司，安部牧人，梅村雅之：GPU・FPGA 複合演算加速による宇宙輻射輸送コード ARGOT の性能評価，情報処理学会研究報告, 2020-HPC-173 (2020).

[20] 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).

[21] Kobayashi, R., Oobata, Y., Fujita, N., Yamaguchi, Y. and Boku, T.: OpenCL-ready High Speed FPGA Network for Reconfigurable High Performance Computing, *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, HPC Asia 2018, New York, NY, USA, ACM, pp. 192–201 (online), available

© 2020 Information Processing Society of Japan

from ⟨http://doi.acm.org/10.1145/3149457.3149479⟩ (2018).
[22] 藤田典久，小林諒平，山口佳樹，朴 泰祐：OpenCL による FPGA 上の演算と通信を融合した並列処理システムの実装及び性能評価，情報処理学会研究報告, 2018-HPC-167 (2018).
[23] 藤田典久，小林諒平，山口佳樹，朴 泰祐：再構成可能なハードウェアを用いた演算と通信を融合する手法の提案と性能評価，情報処理学会研究報告, 2019-HPC-171 (2019).
[24] Norihisa, F., Ryohei, K., Yoshiki, Y., Tomohiro, U., Kentaro, S. and Taisuke, B.: Performance Evaluation of Pipelined Communication Combined with Computation in OpenCL Programming on FPGA, *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (2020 (to be published)).