Vol.2017-HPC-161 No.12
2017/9/20

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

藤田 典久[1]　小林 諒平[1,2]　山口 佳樹[2,1]　大畠 佑真[2]　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、アクセラレータ間を密結合し低レイテンシで通信を行う TCA (Tightly Coupled Accelerators) と呼ばれるアーキテクチャを提案し，FPGA (Field Programmable Gate Array) を用いた TCA 実装として PEACH2 (PCI Express Adaptive Communication Hub Ver.2) の開発を行ってきた．これらの研究を基に現在，TCA の概念をより進めたアーキテクチャとして AiS (Accelerators in Switch) というコンセプトの研究を進めている．AiS は通信機構の中にアプリケーションに特化した演算機構を組み込み，FPGA 内での演算機構と通信機構のより強い連携を実現する次世代の並列演算加速機構である．これまでにも PEACH2 に対して演算機構を組み込む研究は行われてきたが，PEACH2 は Verilog HDL (Hardware Description Language) によって全体が記述されており，AiS における演算部についても Verilog HDL を用いて記述しなければならず，開発コストが高く，FPGA の専門家でなければその開発ができないという問題があった．近年の FPGA 開発環境の進歩により，より一般的な環境で AiS を実現できるようになり，さらに通信性能についても 40Gbps, 100Gbps といった高速な通信機構を扱え，また，ソフトウェアで用いられている言語から回路を合成する高位合成と呼ばれる技術が普及してきた．Intel FPGA では OpenCL を用いた高位合成処理系があり，OpenCL 言語からの回路の生成だけでなく，OpenCL API を用いた FPGA の制御が可能となるが，CPU や GPU 向けに記述・最適化された OpenCL コードをそのまま用いても性能がでないことがわかっており，FPGA 向けの最適化をどう行うかが課題となる．本稿では Intel FPGA 向け高位合成開発環境である Intel FPGA SDK for OpenCL を用いて，宇宙輻射輸送シミュレーションコード ARGOT の中で用いられている ART 法を FPGA 向けに最適化を行う．ART 法を FPGA に実装するにあたって，どのように FPGA 内部で並列演算を行うか，どのような FPGA 向け最適化を行うかについて述べる．Intel Arria 10 FPGA を用いて性能評価を行い，CPU 実装と比べて 14.6 倍の高速化が得られ，その実装は 63%の回路リソースを利用し動作周波数は 236.11MHz であった．

1. はじめに

近年，GPU (Graphics Processing Unit) や Xeon Phi といったアクセラレータが HPC 分野で広く用いられている．これらのアクセラレータは，多数のスレッドや広い幅の SIMD (Single Instruction Multiple Data) 演算を用いて高いピーク演算性能を実現している．それに加えて，レイテンシ志向演算装置である CPU と，スループット志向演算装置である各種アクセラレータの間を埋めるデバイスとして，FPGA (Field Programmable Gate Array) が注目されている．

筑波大学 計算科学研究センターでは TCA (Tightly Coupled Accelerators) と呼ばれるアーキテクチャを提案してきた．TCA はアクセラレータ間を密結合し低レイテンシで通信を行うというものであり，NVIDIA GPU 向け TCA 実装として PEACH2 (PCI Express Adaptive Communication Hub Ver.2) [1] が開発され，基本通信機能の実装と各種ベンチマークの評価が行われてきた．しかし，PEACH2 上での演算加速機能の組み込みは，プログラミングが困難であることに加え，PEACH2 が当時の PCIe (PCI Express) 技術に基づく外部通信を採用していたことによる通信スループットと並列拡張性の限界が見えてきた．これらの背景の下，我々は TCA をより進めたシステムとして AiS (Accelerator in Switch) と呼ばれるコンセプトの研究を進めている．AiS はより密結合されたアーキテクチャであり，通信機構の中にアプリケーションに特化した演算機構をより積極的に組み込むものである．PEACH2 は FPGA で開発されており回路内部を変更できるため，これまで

[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

に PEACH2 に対して演算機構を組み込む研究 [2], [3] が行われている．しかしながら，PEACH2 は Verilog HDL (Hardware Description Language) によって全体が記述されており，AiS 演算部についても Verilog HDL を用いて記述しなければならず，開発コストが高く，FPGA の専門家でなければ AiS の開発ができない問題があった．

近年の FPGA 開発環境の進歩により，より一般的な環境で AiS を実現できるようになってきた．さらに，40Gbps〜100Gbps といった高速な通信機構を扱えるようになり，また，FPGA の開発環境においても，HDL をではなくより上位の記述言語から回路を合成する高位合成と呼ばれる技術が普及してきた．Intel FPGA では OpenCL を用いた高位合成処理系があり，OpenCL 言語からの回路の生成だけでなく，OpenCL API を用いた FPGA の制御が可能となる．我々のこれまでの研究 [4] で，OpenCL から通信機構を制御することは可能であるとわかっており，OpenCL を用いて AiS を実現することは可能であると考えている．

しかしながら，OpenCL を用いて記述できるとはいえ，CPU や GPU 向けに記述・最適化された OpenCL コードをそのまま用いても性能がでないことがわかっており，FPGA 向けの最適化をどう行うかが課題となる [5], [6], [7]．また，VHDL を用いて設計した回路と OpenCL を用いて設計した回路を比較している研究 [8] もあり，Hill らは OpenCL でも VHDL と同等の性能を得られるものの，リソース使用量が増大すると述べている．

AiS の研究においては，基本的な記述性や性能はもちろん，これを有効に利用可能なアプリケーションの開発が重要である．我々は，AiS の実用化のためのアプリケーション分野として，計算宇宙物理学，特に初期天体に関するシミュレーションを対象に選んだ．本研究の目的は Intel FPGA 向け高位合成開発環境である Intel FPGA SDK for OpenCL を用いて，宇宙輻射輸送シミュレーションコード ARGOT の中で用いられている ART 法を FPGA 向けに最適化することである．ART 法を FPGA に実装するにあたって，どのように FPGA 内部で並列演算を行うか，どのような FPGA 向け最適化を行うかについて述べる．将来的には ART 法に AiS を適用することを考えており，本研究ではまず計算部分を FPGA 向けに最適化を行う．

2. 宇宙輻射輸送シミュレーションコード ARGOT

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 [9]，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 [10] を用いて解く．ただし，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 法はレイトレーシングを用いたアルゴリズムであり，空間を 3 次元メッシュに分割して扱う．**図 1** にあるようにレイは境界から発射され直進する．ART 法で扱うレイは平行光のみであり，屈折や反射などは発生しない．レイの角度 (向き) は HEALPix [11] アルゴリズムを用いて生成される．HEALPix によって分割されたピクセルの中央を通るようなレイを計算に用いる．レイはレイトレーシング中に通過したそれぞれのメッシュでガスの反応を計算する．その結果は積分されるため，メッシュデータに足し込まれて更新される．

ART 法ではレイトレーシングを行うため，各レイについては進行方向に向かって順々に計算しなければならないため並列化が難しい．一方で，異なるレイの間に計算の依存関係はなく，どの順番で計算を行なっても問題ないため，並列化が容易である．**図 3** にあるように，ART 法はスレッドを用いたノード内並列化，MPI を用いたノード間並列化のどちらにも対応しており，スレッド並列では複数のスレッドがそれぞれ異なるレイについて計算を行うことで並列計算を行う．また，ノード間並列化には MWF (Multiple Wave Front) [12] と呼ばれるアルゴリズムを用いる．

MWF の概要を**図 2** に示す．図にある 3x3 の正方形はそれぞれプロセスを表し，色の違う矢印はそれぞれ角度の異なるレイを表す．プロセス全体で大きなパイプラインを形成し，順々に異なる角度のレイについて計算を行う．レイの進行方向に応じてプロセス間の順番を決定することで，計算順序を正しく保持する．

図が煩雑になるため図 1，図 2 は簡略化しており，実際の計算で用いられるメッシュは 3 次元空間に配置され，MWF 法におけるプロセス分割も同様に 3 次元に行われる．メッシュサイズは小規模な問題で 100^3，大規模な問題で 1000^3 程度となる．レイは境界のメッシュの中央より発射されるため，レイの本数はメッシュ数に比例し，レイの角度の数については，700 から数千となる．

ART 法は計算するレイの角度によってメッシュデータへのアクセスパターンが変化し，また，レイは HEALpix を用いて生成されていることから全球に均一に分布しているため，CPU 等の SIMD 型の計算機では高速に計算することが難しい．FPGA であれば，内蔵メモリに低レイテンシ高バンド幅でアクセスでき，アクセスパターンも自由にプログラムできる．したがって，ART 法は FPGA に適したアルゴリズムであると考えている．

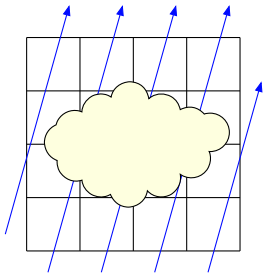

図 1: ART 法によるレイトレーシング．青の矢印はレイ，黄色の雲は計算対象のガスを表す．

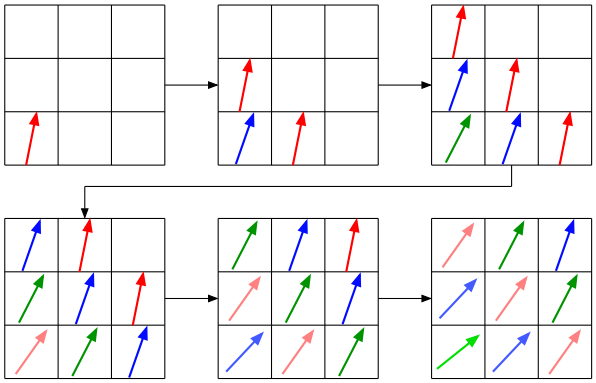

図 2: Multiple Wave Front 法のイメージ．正方形は MPI プロセスを示す．

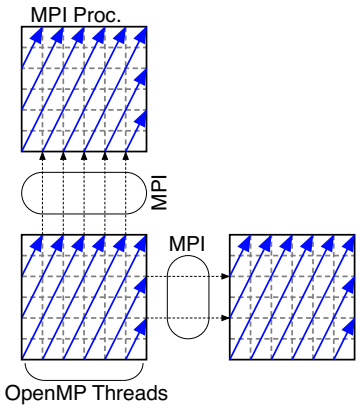

図 3: MPI, OpenMP 並列化の概要．

3. Intel FPGA SDK for OpenCL 開発環境

本章では，Intel FPGA における開発手法について述べる．FPGA の回路を開発する際の手法には，大きく分けて 2 つの手法があり，1 つは，ハードウェア記述言語 HDL (Hardware Description Language) と呼ばれる言語を用いて記述する方法，もう 1 つは，高位合成言語を用いて記述する方法である．高位合成言語は，一般的なソフトウェア開発に用いられる言語を用いて回路開発を行うものであり，C 言語などで記述されたプログラムを HDL に変換し，FPGA 回路の開発を行う．

3.1 Intel FPGA SDK for OpenCL を用いる開発の流れ

Intel 社は OpenCL を用いて FPGA 回路を設計できる Intel FPGA SDK for OpenCL というパッケージを提供しており，その SDK を用いることで，OpenCL で記述したプログラムを FPGA 回路にコンパイルできる．Intel FPGA SDK for OpenCL は all-in-one 型の開発環境であり，OpenCL を用いて生成された回路およびホストプログラムは，それだけで FPGA を動作させることができ，HDL の知識がほとんど無くても FPGA を利用できる．

ホスト側で動作するコードは通常の OpenCL ソフトウェア開発と同じく，ホスト用のコンパイラを利用してコンパイルを行い，Intel FPGA SDK for OpenCL 用のライブラリをリンクする．OpenCL では，ソースコード (.cl ファイル) を実行時に読み込みコンパイルする方式 (オンラインコンパイル) と，事前にソースコードをオブジェクトファイルに変換しておき，そのファイルを実行時に読み込む方式 (オフラインコンパイル) の 2 つの利用方法があるが，OpenCL コードを FPGA の回路データにコンパイルする作業は時間がかかるため，Intel FPGA SDK for OpenCL ではオフラインコンパイル方式のみ利用できる．

Intel FPGA SDK for OpenCL では，OpenCL の標準で定義されている API を用いて FPGA の制御を行える．例えば，回路データを FPGA に書き込む際，HDL を用いる従来手法では専用のツールを用いて行うが，OpenCL では標準 API である `clCreateProgramWithBinary` 等を使うことで行える．

3.2 Channel によるカーネル間通信

Intel FPGA SDK for OpenCL による OpenCL 拡張の一つに Channel 機能があり，カーネル関数の間でデータを直接通信する機能である．Channel を用いない従来の手法でカーネル間通信を行う場合，**図 4** の様にグローバルメモリを経由してデータをやりとりする．一方で，Channel を用いてカーネル間通信を行う場合は，**図 5** の様に，グローバルメモリ (DDR4 DRAM) を経由することなく通信を行う．この場合，通信路は FPGA チップ内部で完結しており，低コストにカーネル間通信を行える．

OpenCL 言語で Channel を宣言するコード例を**図 6** に示す．図 6 の 2 行目にある pragma をソースコード中に記述すると，Channel 機能が有効化され，Channel 変数の宣言などが可能となる．Channel 変数は図 6 の 4 行目の様に宣言を行う．Channel 変数のデータ型は int, float のような基本的な型だけでなく，struct などのユーザ定義型を用いることもできる．`read_channel_altera`, `write_channel_altera` 関数はそれぞれ Channel に読み書きする API である．

Channel には FIFO バッファを含めることもでき，図 6

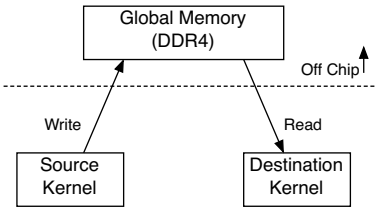

図 4: グローバルメモリを経由するカーネル間通信.

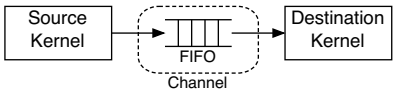

図 5: Channel を経由するカーネル間通信.

```
1  // Channel機能を有効化するためのpragma
2  #pragma OPENCL EXTENSION cl_altera_Channels:enable
3  // float型のデータを通信するChannelの宣言
4  Channel float c;
5  // FIFOサイズがfloat型8要素のChannelを宣言
6  Channel float c2 __attribute__((depth(8)))
7  // Channel cにvalueを書き込む
8  write_channel_altera(c, value);
9  // Channel cから値を読み込む
10 value = read_channel_altera(c);
```

図 6: Channel 機能を用いるコード例.

の 6 行目の様に depth 属性でそのサイズを指定する．FIFO バッファのサイズは 0 (FIFO なし) を指定することもできる．depth 属性を指定しない場合は，FIFO バッファのサイズをコンパイラが自動的に決定する．depth 属性で指定する値は要求値であり，実際の FIFO サイズは指定した値を上回る場合がある．FPGA の内蔵メモリは一定サイズのブロック単位で実装されており，コンパイラが内蔵メモリの利用に無駄が出ないように FIFO サイズを自動で調整するためである．

3.3 autorun によるカーネル関数の自動起動

Intel FPGA SDK for OpenCL による FPGA 向けの OpenCL 拡張の一つとして，カーネル関数に対する autorun 属性がある．本来，OpenCL においてカーネルはホストから起動されるものでありデバイスは自律動作できないが，autorun 属性が付与されたカーネルは，`clBuildProgram` 関数を用いて FPGA に回路データが書き込まれた後，自動で起動された状態となる．

一般的に，前述した Channel 機能を用いて複数のカーネルから FPGA 回路を構成する場合，1 つの処理を行う際に複数のカーネルを起動したり終了を待機したりしなければならず，カーネルの起動や同期に関するオーバーヘッドが大きく，処理全体の性能が悪化する．autorun を用いることで，これらのオーバーヘッドによる影響を小さくできる．

```
Channel float input, output;

__kernel __attribute__((
  autorun,
  max_global_work_dim(0)))
void autorun_sample() {
    float v = read_channel_altera(input);
    v = v + 1.0f;
    write_channel_altera(output, v);
}
```

図 7: autorun 属性を用いるコード例.

autorun を用いるコード例を**図 7** に示す．autorun を適用するカーネルにはいくつかの制限があり，これらの制限に抵触するとコンパイルエラーとなる．制限の内容は次の通りである．

(1) カーネル関数に引数を持たない．
(2) I/O Channel を利用しない．
(3) work item, work group の数がコンパイル時に決定できる．

autorun カーネルはホストとの介在なく自動的に起動されるため，制限 1 および制限 3 がある．引数では入出力を与えられないため，autorun カーネルのデータ入出力は Channel 経由で他のカーネルより行う．すなわち，データを別のカーネルから受け取り，別のカーネルに出力することとなる．また，autorun カーネルは自身の実行が完了したとしても，即座に再度実行される．

4. ART on FPGA 実装

4.1 実装概要

本節では，本研究で実装した FPGA 実装の概要について述べる．基本的な設計方針は演算を行うカーネルを複数用意し並列化を行い，それらのカーネル間に Channel を用いて相互接続することである．それぞれの演算カーネルは異なるメッシュ領域を担当し，レイトレーシングを行う．レイがどのカーネルで演算されるかはレイの位置によって決まり，レイトレーシングの計算に伴ってレイが自担当領域から外れる際は，隣接カーネルに Channel を通じて通信を行いレイの計算を引き継ぎ，実際のレイの動きに伴い計算を行うカーネルが移動していく．

FPGA に実装する OpenCL カーネル構造の概要を**図 8** に示す．“Memory Reader” はグローバルメモリである DDR4 メモリよりメッシュデータを読み込むカーネル，“Memory Writer” は計算結果をグローバルメモリに反映するカーネルである．“PE Array” は ART 法の計算を行う計算部分である．“PE Array” 部分は複数の要素から構成されており，詳細は次節で述べる．“PE Array” より得られる計算結果は積分の部分和であるため，“Memory Writer” はグローバルメモリよりデータを読み込み，部分和の結果を反映

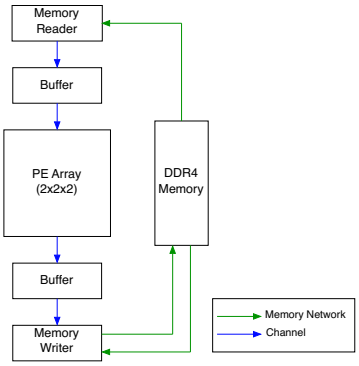

図 8: 回路の全体構成.

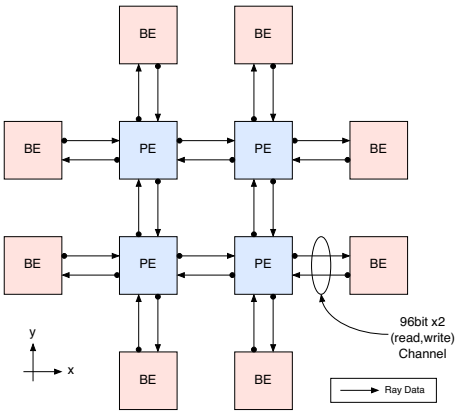

図 9: x-y 平面における，PE (Processing Element), BE (Boundary Element) の接続関係.

した後にグローバルメモリに書き戻す動作を行う．また，“Buffer” は “Memory Reader” の後，“Memory Writer” の前に接続されており，メモリデータを一時的に保存する．

図 8 の矢印の色は，コンポーネント間を接続しているバスの種類を表す．OpenCL カーネルとメモリ間の接続は OpenCL コンパイラが生成するメモリネットワーク (緑) によって接続され，OpenCL カーネル間は Intel FPGA SDK for OpenCL の拡張である Channel 機能 (青) を用いて接続する．

現時点で ART on FPGA は実装途中であるため，CPU 版と比べるといくつかの機能が不足している．図 8 ではグローバルメモリにアクセスする様に書かれているが，計算の始めに初期値をグローバルメモリから読み込み，計算の最後に結果を書き込むのみである．計算の進行にあわせてデータを入れ替える機構がないため，計算に必要なデータは全て FPGA の内部メモリに置かれ，したがって，扱える問題サイズが内蔵メモリサイズに制限される．また，ネットワーク通信を用いた並列化は実装されておらず，1 FPGA のみで計算を行う．

4.2 Channel を用いた並列化

本節では，図 8 で “PE Array” として表している部分のカーネルの構造について述べる．“PE Array” は**図 9** にあるように，PE (Processing Element) と BE (Boundary Element) から成り，相互に Channel を用いて接続される．PE は ART 法の計算カーネルを含んでおりアルゴリズムのコアとなる部分であり，BE はレイトレーシングにおける袖領域の処理を行う部分である．

本研究で開発した FPGA 実装では，FPGA チップ内の並列化に Channel を用いる．オリジナルの実装において用いられているノード間並列化手法の MWF 法を FPGA 向けに実装したものであるが，Channel による通信のコストは低いため，演算パイプラインの中に Channel アクセスを組み込み，より細粒度に通信を行うように改良したものである．FPGA 内部では，**図 10** にあるように 1 つの FPGA が担当する問題領域を複数の小空間に分割し，PE の間で処理を分割する．すなわち，CPU の MWF 実装における MPI プロセスが PE に相当する．

Channel を通じて PE がレイデータを相互に通信することでレイトレーシングアルゴリズムを実現する．BE は袖領域におけるレイの処理を行うが，現在の実装ではレイの初期値設定および，計算領域外に向うレイの破棄を行うのみである．今後，ネットワークを用いた複数 FPGA を用いる並列処理を行う際に，処理の実装を容易にするために PE と BE を分割した実装を行っている．

PE は，x, y, z それぞれの次元において隣接する PE と相互に接続される．図 9 は，図が煩雑になるのを避けるため，図 9 は x-y 平面における状態のみを示している．全体として，x 次元，y 次元，z 次元にそれぞれ 2 つの PE が配置されており，全体として $2 \times 2 \times 2 = 8$ 個の PE から構成される．また，BE は袖領域の処理を担当するためのものであるため，隣接する PE と接続される．BE の接続は PE の接続と異なり，隣接する 1 つの PE のみと接続され，隣接する BE 間は接続されない．PE, BE どちらの接続も 96bit の Channel によって接続されており，双方向に通信を行うために 2 組の Channel を用いる．Channel のビット幅は，ART 法が扱うレイを示す構造体のサイズによって決定される．

Channel へのアクセスは演算パイプラインに組み込まれ，送信側・受信側のどちらもがパイプラインストールを起すことなく動作する場合，1 クロックサイクルにつき 1 要素 (= sizeof(Channel の型) byte/clock) のデータを通信できる．また，FPGA 内に複数の Channel が存在する場合，それぞれが独立に動作し通信できる．したがって，FPGA 内の Channel を用いた通信コストはノード間通信よりも低コストであるといえる．

4.3 autorun を用いた最適化

図 8 の図にあるカーネルのうち，Buffer (x2), PE Array を構成するカーネルに autorun を付与し，残る Memory

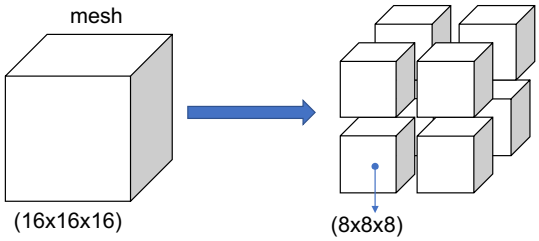

図 10: FPGA 内における問題の領域分割の方法.

Reader, Memory Writer については従来手法で制御を行う．これらの autorun を付与するカーネルは，図 8 からわかるように，他のカーネルから Channel 経由で入力し，他のカーネルに Channel 経由で出力を行うものである．

全体の動作の制御は Memory Reader の起動によって行う．全体で Channel を用いたパイプラインを構成しているため，パイプライン中間にある autorun カーネルが自動起動するとしても，パイプラインの上流にある Memory Reader を起動しない限り計算は進まないためである．

Memory Writer も，autorun を付与するカーネルと同様に，パイプライン上流からデータが来なければ処理が進まない．しかしながら，アクセス先のメモリアドレスをホストから与える必要があることと，ホストが計算結果の書き込みが完了したことを検知しなければならないため，通常のカーネルとして定義している．

5. 性能評価

5.1 評価環境

性能評価には PPX (Pre-PACS Version 10) 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の次世代機 PACS-X (PACS Version 10) のプロトタイプシステムである．PPX は AiS コンセプトの実証実験に用いられている．

PPX ノードの構成を**表 1** と**図 11** に示す．1 ノードあたり，CPU として Intel Xeon E5-2660 v4 を 2 ソケット，FPGA ボードとして BittWare A10PL4 ボードを 1 枚搭載している．CPU と FPGA ボード間の接続は PCIe バスであり，Gen.3 x8 レーンを用いて接続されており，OpenCL を用いる際のホストからの FPGA の制御は PCIe を経由して行われる．ただし，PPX システムは CPU, GPU, FPGA の 3 種類の演算装置を持つヘテロジニアスな環境であるが，本研究では GPU は利用しておらず，CPU と FPGA のみを用いる．

5.2 評価手法

性能評価には，ARGOT プログラムから ART 法の計算を行うコア部分のみを抜き出して作成したベンチマークプログラムを用いる．ARGOT に組み込んでおらず，また，

表 1: 評価環境

PPX 計算ノード	
CPU	Intel Xeon E5-2660 v4 × 2
CPU Memory	DDR4 2400MHz 64GB (8GB × 8)
GPU	NVIDIA Tesla P100 (PCIe card version)
Infiniband	Mellanox ConnectX-4 EDR
Host OS	CentOS 7.3
Host Compiler	gcc 4.8.5
FPGA Compiler	Intel Quartus Prime Pro, Intel FPGA SDK for OpenCL Version 16.1.2.203
Ethernet Switch	Mellanox MSN2100-CB2R
FPGA ボード (BittWare A10PL4)	
FPGA	Intel Arria10 GX 10AX115N3F40E2SG
LE (Logic Element)	1,150 K
ALMs (Adaptive Logic Module)	427,200
Registers	1,708,800
DSP (Digital Signal Processor)	1,518
M20K memory blocks	2,713
M20K memory size	53 Mbits
Exteral Memory	DDR4 2133MHz 8GB (4GB × 2)
通信ポート	QSFP+ x2 (40 Gbps x2)

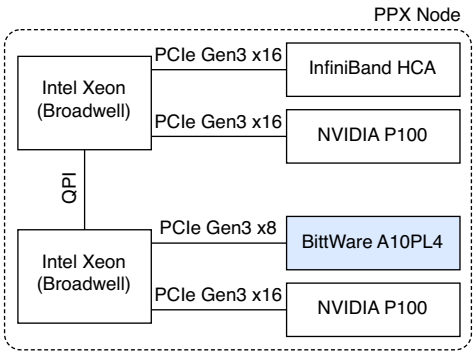

図 11: PPX ノードの構成図.

ART 法の計算量は入力データに依存しないため，入力メッシュデータは乱数より生成している．ARGOT は扱うガスの組成を，水素のみ，水素+ヘリウムの 2 通り扱える．ガス組成は`#ifdef` によって切り替えられ，解く問題によってどちらを利用するかを切り替えるが，今回の性能評価では水素+ヘリウムのガスを扱う．

問題サイズは FPGA 実装のサイズ制限より $(N_x, N_y, N_z) = (16, 16, 16)$ である．HEALPix を用いてレイを生成する際に用いるパラメータ $N_{side} = 8$ とし，生成されるレイの種類 (角度) は 768 通りとなる．

評価に用いる環境は前節で述べた PPX システムを用い，ノード数は 1 ノードである．PPX 1 ノードあたり CPU は

2 ソケットあるが，QPI を跨ぐメモリアクセスによる性能低下を避け，また，1 CPU と 1 FPGA と比較するために CPU は 1 ソケット・14 コアを用いる．スレッド並列化には OpenMP を用い，1 ノードによる性能測定のため MPI は利用しない．また，FPGA 実装においては，計算時間はホスト CPU 上で測定しており，OpenCL のカーネル起動時間および同期待機時間は含まれるが，メモリ転送時間は計算時間に含めていない．

性能の指標には，1 秒あたりのメッシュ計算量 (M mesh/sec) を用いる．この値は，1 本のレイが 1 個のメッシュ通過することを 1 として数え，1 秒間に何個のメッシュ通過を処理できるかを表す．予め，1 回の計算あたりの総メッシュ通過数を求めておき，それを計算時間で割ることによって求める．総メッシュ通過数は，メッシュサイズ N_x, N_y, N_z および レイ生成パラメータ N_{side} に依存し，今回の性能評価で用いるパラメータの場合，総メッシュ通過数は 5,848,064 となる．

5.3 FPGA リソース使用量

FPGA の各種リソース使用量を**表 2** に示す．ALM (Adaptive Logic Module) は Arria10 FPGA における論理回路を実現する最小ブロック単位であり，表中の “ALMs” はそのブロックの使用量を表す．“Registers” は ALM 含まれているレジスタの使用量を表す．“M20K” は FPGA 内部にある SRAM ブロックを指し，1 つの M20K ブロックは 20K bit の容量を持つ．“MLAB” は MLAB モードとして動作している ALM の数を表す．ALM の動作モードの 1 つに MLAB と呼ばれるものがあり，ALM を M20K よりも小さい小型 SRAM ブロックとして利用するモードである．“DSP (Digital Signal Processor)” は加算器や乗算器を含む演算のための回路ブロックである．従来の Intel FPGA の DSP は整数演算のみを行えたが，Arria 10 の DSP は IEEE 準拠の単精度浮動小数点数積和器を含んでおり，OpenCL 上で float 型の演算を記述すると DSP を用いて実装される．

autorun のあり・なしどちらの実装も最も利用量の多いリソースは M20K SRAM ブロックであり，autorun ありで 68%，autorun なしで 63%を利用している．M20K はさまざまな用途に用いられているが，パイプライン FIFO，DRAM アクセスのバッファ，OpenCL における変数データを保持する等の用途がある．

autorun のありとなしを比較すると，大きく使用量が変化しているのは MLAB 使用量である．autorun をカーネルに付与すると，それらのカーネルに対してホストから制御を行う必要がなくなり，その用途に用いられている回路量を削減できる．また，M20K の使用量も減少していることから，autorun なしでは M20K に割り当てられていた領域が，autorun を用いることで利用が減少した ALM を用いて MLAB 領域に割り当てられたと考えられる．

演算を司る DSP の使用量は，autorun のありなしに関わらず 35%を使用している．DSP は演算ロジックによって利用されているため，autorun によって使用量が変動しない．FPGA における演算性能を高めるためには，全ての DSP を利用することが望ましく，DSP の使用率を高めていくことが今後の課題である．本実装で DSP の使用量を増やすということは，すなわち，PE の数を増やすということであるが，PE を実装すると DSP だけでなくその他のリソースも消費されるため，全体のバランスを考慮して実装しなければならない．

各カーネルのリソース使用量の詳細を**表 3** に示す．“BSP” は Board Support Package に含まれている回路使用量を示し，“Channel FIFO” は実装全体の Channel で利用している利用量を示す．括弧の中の数字は同型カーネルの複製量を示し，例えば PE は同じカーネルが 8 個複製されているため “(x8)” となる．BSP は FPGA ボード間の差異を吸収するために用いられるハードウェアコンポーネントであり，一般的にボードベンダより提供され，PCI Express コントローラやメモリコントローラなどが実装されている．

5.4 性能評価

CPU 実装と FPGA 実装で性能を比較したデータを**表 4** に示す．比較に用いる CPU 実装は FPGA 実装の基となったものであり，C 言語を用いて記述されている．本研究で実装した FPGA 向け OpenCL コードを，CPU 向け OpenCL コンパイラでコンパイルしたものではない．CPU 実装は OpenMP を用いて並列化されており，レイ単位でスレッド並列化されている．FPGA 実装 (with autorun) は 1715 M mesh/sec という性能が得られ，CPU と比べると 14.6 倍高速である．また，FPGA 実装における autorun なしと autorun ありを比較すると，autorun ありの方が約 2.9 倍高速であり，Intel FPGA SDK for OpenCL においては autorun による最適化が重要だということがわかる．

PE1 個あたりのスループットは 1 mesh/cycle であるため，FPGA 全体としては 8 mesh/cycle のスループットを持つ．autorun あり実装の動作周波数は表 2 より 236.11 MHz であるため，理論ピークスループットは 1888.88 M mesh/s となる．したがって，autorun ありの実装は $1714.97/1888.88 = 0.908$ より，90.8% の効率であり高効率であるということがわかる．

FPGA 実装は CPU 実装と比べると高速であるが，現在の FPGA 実装は内蔵メモリに格納できないような大きな問題は扱えないという制限があり，より大きい問題を扱えるようにすることが今後の課題である．その際は，内蔵メモリと外部メモリ間でデータを入れ替えつつ計算を行う必要があり，効率よいメモリアクセスを実現できるかが問題となると考えている．

表 2: autorun 属性の利用の有無による違いによる FPGA リソース使用量の差．3 行目は 1 行目と 2 行目の差を示す．

	ALMs	(%)	Registers	(%)	M20K	(%)	MLAB	MLAB [bit]	DSP	(%)	freq.[MHz]
without aurotun	228610	54%	473747	55%	1839	68%	4330	47968	536	35%	228.57
with autorun	228835	54%	467225	55%	1716	63%	7350	138288	536	35%	236.11
差分	+225		-6522		-123		+3020	+90320	0		+7.54

表 3: autorun ありの場合のリソース消費量の詳細．

	ALMs	(%)	Registers	(%)	M20K	(%)	DSP	(%)
PE (x8)	122158.8	28.6%	153310	17.9%	600	22.1%	512	33.7%
BE (x24)	6765.9	1.6%	10103	1.2%	0	0.0%	0	0.0%
Buffer (x2)	18643.7	4.4%	27687	3.2%	552	20.3%	0	0.0%
Memory Writer	12112.7	2.8%	16712	2.0%	91	3.4%	24	1.6%
Memory Reader	3018.5	0.7%	4440	0.5%	61	2.2%	0	0.0%
Channel FIFO	25713.2	6.0%	56740	6.6%	144	5.3%	0	0.0%
BSP	34535.4	8.1%	25536	3.0%	226	8.3%	0	0.0%
etc.	5886.8	1.4%	172697	20.2%	42	1.5%	0	0.0%

表 4: CPU および FPGA 実装の性能比較．

実装	性能 [M mesh/sec]	CPU 比
CPU	117.49	-
FPGA (without autorun)	593.11	5.05
FPGA (with autorun)	1714.97	14.60

表 5: FPGA 実装の計算時間内訳．

	without autorun	with autorun
カーネル起動時間 [ms]	6.05	0.02
カーネル完了待機時間 [ms]	3.81	3.40
合計 [ms]	9.86	3.41

OpenCL 実装の計算時間をより詳細に分別したものが**表 5** である．“カーネル起動時間” は動作に必要なカーネルを順次起動するためにかかる時間であり，OpenCL の API としては `clEnqueueNDRangeKernel` に相当する．“カーネル完了待機時間” はカーネルの起動が終了してから，全ての Command Queue で `clFinish` の呼び出しが完了するまでの時間を示す．

表 4 より autorun のありなしの違いで，カーネルの起動時間に大きな差があることがわかる．autorun ありの場合では 58 個のカーネルの起動操作を行うが，autorun なしの場合は 2 個のカーネルのみで起動できるため，大きな差が生れていると考えられる．両実装で演算部の構造に変化はないため，両実装の性能差は動作周波数の差より生れると考えられるが，カーネル完了待機時間 (1.12 倍) の差は動作周波数の差 (1.03 倍) よりも大きい．この差も取り扱うカーネルの数の差によるものだと考えられる．カーネルの終了にも数に応じたコストがかかるためである．

以上の結果より，Intel FPGA SDK for OpenCL 環境では多数のカーネルを操作する際のオーバーヘッドが無視できないことがわかる．したがって，Channel を用いて複数のカーネルから成るプログラム構造を取る場合，autorun 属性によるカーネルの自動起動を用いた最適化を行わなければ，FPGA の高性能を活かせないといえる．

6. おわりに

本研究では，宇宙輻射輸送シミュレーションコード ARGOT で用いられているアルゴリズムである ART 法を，Intel FPGA 向けの高位合成開発環境である Intel FPGA SDK for OpenCL を用いて FPGA 上に実装した．Intel FPGA SDK for OpenCL の Channel 拡張を用いて複数の PE を接続することで演算の並列化を行い，FPGA 向けの最適化を行うことで，CPU と比べて 14.58 倍の高速化が達成された．

Intel FPGA SDK for OpenCL 環境ではカーネル操作に関するオーバーヘッドが大きく，また，前述した Channel による複数カーネルの協調動作を伴う場合は，一度に取り扱うカーネルの量が多く，このオーバーヘッドが無視できない．autorun なしと autorun ありを比較すると，autorun ありの方が約 2.9 倍高速であり，このオーバーヘッドを軽減するために，autorun 化できるカーネルは出来るだけ autorun にする方が望ましい．オーバーヘッド軽減だけでなく，リソース使用量も軽減できるためである．

現在の実装では扱える問題サイズに制限があるため，これを解決するために内蔵 SRAM だけでなくグローバルメモリも併用して演算できる機能の追加が今後の課題である．また，ART on FPGA に Accelerator in Switch コンセプトを適用することで，複数の FPGA に分散した並列処理の実現に関する研究を進めていく予定である．

謝辞 本研究の一部は，JST-CREST 研究領域「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」研究課題「ポストペタスケール時代に向けた演算加速機構・通信機構統合環境の研究開発」，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] Hanawa, T., Kodama, Y., Boku, T. and Sato, M.: Interconnection Network for Tightly Coupled Accelerators Architecture, *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*, pp. 79–82 (online), DOI: 10.1109/HOTI.2013.15 (2013).

[2] Kuhara, T., Tsuruta, C., Hanawa, T. and Amano, H.: Reduction calculator in an FPGA based switching Hub for high performance clusters, *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–4 (online), DOI: 10.1109/FPL.2015.7293985 (2015).

[3] Tsuruta, C., Miki, Y., Kuhara, T., Amano, H. and Umemura, M.: Off-Loading LET Generation to PEACH2: A Switching Hub for High Performance GPU Clusters, *SIGARCH Comput. Archit. News*, Vol. 43, No. 4, pp. 3–8 (online), DOI: 10.1145/2927964.2927966 (2016).

[4] 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).

[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] 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).

[8] 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).

[9] 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).

[10] 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).

[11] 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).

[12] Nakamoto, T., Umemura, M. and Susa, H.: The effects of radiative transfer on the reionization of an inhomogeneous universe, *Monthly Notices of the Royal Astronomical Society*, Vol. 321, No. 4, pp. 593–604 (online), DOI: 10.1046/j.1365-8711.2001.04008.x (2001).