情報処理学会論文誌　コンピューティングシステム　Vol.12　No.3　64–75 (July 2019)

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

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

受付日 2018年12月4日, 採録日 2019年2月26日

概要：近年，High Performance Computing（HPC）におけるチャレンジの中の一つに，高い性能と低い消費電力を持つ Field Programmable Gate Array（FPGA）技術をどのようにして次世代のスーパーコンピュータに用いるかという問題がある．Graphics Processing Unit（GPU）が HPC におけるアクセラレータとして最も広く用いられているが，均一な大量の並列計算が必要であり，これが性能上のボトルネックとなる場合がある．一方で，FPGA は再構成回路による柔軟さと効率さを持っており，様々な問題に適応できる可能性を持つ．しかしながら，ハードウェアの動作を記述することは複雑であり，アプリケーションの開発者が FPGA 回路を実装することは容易ではない．近年の FPGA における開発環境の進歩により，OpenCL 言語を用いた高位合成（HLS: High Level Synthesis）開発環境が一般的になってきている．我々のこれまでの OpenCL を用いたカーネル記述の経験より，FPGA 向けにアプリケーション記述する際は “co-design” に基づくアグレッシブなプログラミング戦略が高い性能を達成するうえで必要であることが分かっている．本研究では，宇宙輻射輸送を解くプログラム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 向けに最適化を行う．OpenCL で記述されたアプリケーションに対して co-design に基づく FPGA 向け最適化を適用し，CPU，GPU，FPGA 間での性能比較を行った．マルチコア CPU 実装と比べて最大 4.9 倍の高速化が達成され，GPU 実装との比較では 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，FPGA の方が通信オーバヘッドは 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ことから，今後，並列 FPGA 計算の実装を行う予定である．

キーワード：FPGA，OpenCL，演算加速装置

Optimization on Astrophysical Radiative Transfer Code for FPGAs with OpenCL

Norihisa Fujita[1]　Ryohei Kobayashi[1,2]　Yoshiki Yamaguchi[1,2]　Taisuke Boku[1,2]
Kohji Yoshikawa[1,3]　Makito Abe[1]　Masayuki Umemura[1,3]

Received: December 4, 2018, Accepted: February 26, 2019

Abstract: One of the recent challenges faced by HPC is how to apply FPGA technology to accelerate a next-generation supercomputer as an efficient method of achieving high performance and low power consumption. GPU is the most commonly used accelerator for HPC supported by regularly executed high degree of parallel operations which causes performance bottleneck in some cases. On the other hand, there are great opportunities to flexibly and efficiently utilize FPGAs in reconfigurable circuits to fit various computing situations. However, it is not easy for application developers to implement FPGA logic circuits for their applications and algorithms, which generally require complicated hardware logic descriptions. Because of the progress made in the FPGA development environment in recent years, the HLS development environment using the OpenCL language has become popular. Based on our experience describing kernels using OpenCL, we found that a more aggressive programming strategy is necessary to realize true high performance based on a “co-design” concept to implement the necessary features and operations to fit the target application in an FPGA design. In this paper, we optimize the ART method used in space radiative transfer problems on an FPGA using OpenCL. Using a co-designed method for the optimized programming of a specific application with OpenCL for an FPGA, we achieved a performance that is 4.9 times faster than that of a multicore CPU implementation, and almost the same performance as a GPU implementation. Considering the current advanced FPGAs with interconnection features, we believe that their parallelized implementation with multiple FPGAs will achieve a higher performance than GPU.

Keywords: FPGA, OpenCL, accelerator

1. はじめに

近年，High Performance Computing（HPC）の分野では Graphics Processing Unit（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 Central Processing Unit（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，Field Programmable Gate Array（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．それに加えて，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．

FPGA 開発では，Verilog HDL や VHDL といったハードウェア記述言語（Hardware Description Language: HDL）を用いて回路を記述することが一般的である．HDL は 1 クロック・1 ビット単位で回路の動作を記述するものであり，利用に際してハードウェアの知識が必要となり，計算科学者が FPGA を用いる際の障壁となっていた．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ってきており，前述した開発コストの問題が徐々に解決されている．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 回路を開発できる．たとえば，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 [1], [2]，OpenCL による FPGA 回路の開発が行える．

現在の FPGA の演算性能やメモリ帯域といった絶対性能は，アクセラレータとして広く用いられている GPU に敵わず，すでに GPU で高速に計算できる問題を FPGA に適用しても GPU の性能を超えられるとはいいにくい．したがって，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られる．

本研究の目的は，高性能計算で用いられているアプリケーションを高位合成開発環境で FPGA 向けに最適化し，高性能計算において FPGA がアクセラレータとして有用かどうかを明らかにすることである．本研究では，宇宙物理学の計算で用いられる Authentic Radiative Transfer（ART）法を FPGA のハードウェアの特性を考慮しつつ OpenCL を用いて記述し FPGA 向けに最適化を行う．その際に，FPGA の内部メモリに収まらない大きな実問題を解けるように DDR メモリを用いて計算を行う．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．

本論文の貢献は以下のとおりである．

- ART 法で用いられている指数関数を含む複雑な計算が OpenCL を用いて記述可能であることを示した．
- ART 法に対して FPGA のハードウェア構成を考慮した FPGA 向け最適化を適用し，NVIDIA P100 GPU と同程度の性能を達成した．
- FPGA は空間並列だけでなくパイプラインによって時間方向にも並列計算を行う．したがって，問題サイズが小さい場合であっても性能を発揮でき，その場合，GPU を凌駕する性能を持つことを示した．

本論文の構成は以下のとおりである．2 章で関連研究について述べた後，本研究で対象にするアルゴリズムである ART 法について 3 章で述べ，4 章で高位言語開発環境である OpenCL 開発環境および本研究で利用する拡張機能について述べる．そして，5 章で ART on FPGA の実装方法および並列化手法について述べ，6 章で ART on FPGA の性能評価を行う．最後に，7 章で今後の課題である複数の FPGA を用いて並列計算する手法について，8 章で本研究の結論についてそれぞれ述べる．

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研究はいくつか行われている．文献 [3] 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．文献 [4] 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．文献 [5] 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．Kenter らは文献 [6] で Maxwell の方程式を OpenCL

[1] 筑波大学計算科学研究センター
Center for Computational Sciences, University of Tsukuba, Tsukuba, Ibaraki 305–8577, Japan
[2] 筑波大学システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba, Tsukuba, Ibaraki 305–8577, Japan
[3] 筑波大学数理物質科学研究科
Graduate School of Pure and Applied Sciences, University of Tsukuba, Tsukuba, Ibaraki 305–8577, Japan

で記述し FPGA 向けに最適化を適用し性能評価を行った．このアプリケーションでは非構造格子を採用しており，一般的な格子を用いる計算とは異なりメモリアクセスが複雑になるが，注意深く最適化を行うことで高い性能が達成できることを示している．マルチスレッド実装になっている CPU と性能を比較し，FPGA が約 2 倍高速である結果が得られている．

ここまででいくつかの FPGA を用いた関連研究を示しているが，これらの論文より，FPGA に適した処理をオフロードしまた注意深く FPGA 向けの最適化を行わなければならないことが示されている．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，それに加えて CPU や GPU とはまったく異なる実行モデルを持つ．したがって，アプリケーションのどの処理を FPGA にオフロードするのかが重要となるといえる．

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．そ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，SIMD やマルチコアによる並列性ではうまく扱えない．したがって，我々は本研究で扱うアルゴリズムが FPGA に適するアルゴリズムであると考えている．

3. 宇宙輻射輸送コード：ARGOT

Accelerated Radiative transfer on Grids using Oct-Tree（ARGOT）は筑波大学計算科学研究センター（Center for Computational Sciences: 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り，高速に解くことが求められている．ARGOT は 2 つのアルゴリズム ARGOT 法 [7] [*1] と 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 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 実装および最適化を行う．

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たびに計算する．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 法の計算はすべて単精度浮動小数

[*1] 本論文では，対象とするプログラム名を ARGOT 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．

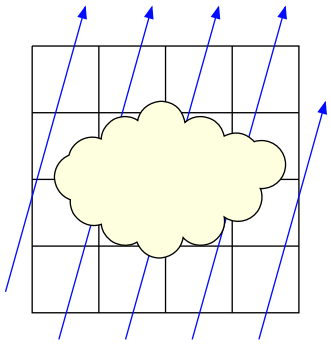

図 1　ART 法で用いられているレイトレーシングの概念図．青い矢印はレイを表し，黄色の雲は反応を計算するガスを表す

Fig. 1　Ray tracing method used in the ART method. Blue arrows and yellow cloud show rays and gas to compute reactions respectively.

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) の組合せの数）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 (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 ごとに 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 メッシュ通過ごと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どお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 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うる．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-like 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めるため，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. Intel FPGA SDK for OpenCL

4.1 概要

本研究では Intel 製の FPGA を対象とする．Intel は自社 FPGA 向けの開発環境として Intel FPGA SDK for OpenCL を提供しており，この SDK を使うことで OpenCL 言語を用いて記述されたプログラムから FPGA 回路を構築できる．この SDK は all-in-one の開発環境であり，OpenCL 言語からハードウェア記述言語へ変換するコンパイラだけでなく，ホスト上で動く OpenCL ランタイムライブラリやカーネルドライバも含む．FPGA は PCIe バスを通じてホストから制御され，それには OpenCL の標準の Application Programming Interface（API）が用いられる．これらの制御回路も SDK によって提供されるため，ユーザはハードウェア記述言語を記述せずとも FPGA を利用することができ，プログラミングコストが低下する．

HLS 環境は開発コストを下げ，FPGA を広く用いるために重要な要素となる．したがって，様々なアプリケーションで高位合成の利点・欠点を明らかにすることは重要であると我々は考える．

Intel FPGA SDK for OpenCL 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 つ拡張を用いており，それぞれの拡張の詳細については次節以降で述べる．

4.2 Channel を用いたカーネル間通信

“Channel” は Intel FPGA SDK for OpenCL による拡張の 1 つであり，カーネル間の通信を行えるようにするものである．Channel の実態はバッファ付き（オプション．なしも可）First-In-First-Out（FIFO）であり，カーネル間に FIFO を通じて通信を行う回路が FPGA 内に生成される．

Channel を使うことの利点は，外部メモリにアクセスすることなく 2 つのカーネル間でデータ交換を行える仕組みであることである．一般的な OpenCL の環境においてカーネル間でデータ交換をする場合は，グローバルメモリを用いるしかなかったが，FPGA 環境におけるグローバルメモリは DDR3 や DDR4 の採用が一般的であり，レイテンシやバンド幅の面から性能が期待できない．一方で，2 つのカーネルが Channel で接続されると，グローバルメモリにアクセスすることなく FPGA 内部のデータパスで通信が完了し，低レイテンシ・高バンド幅の通信が行えるようになる．

なお，UNIX におけるソケット通信はそのつど通信オーバヘッドが発生するが，Channel を用いた FPGA のカーネル間接続は，実際にはクロック単位で同時並列動作するハードウェア（パイプライン）として実装されるため，システムリソース（回路）を消費するが，実行時のオーバヘッドはソケットよりも遥かに少ない．

4.3 autorun 属性によるカーネルの自動起動

“autorun” はカーネル関数に対する属性の拡張であり，カーネルの自動起動を可能とするものである．標準的な OpenCL 環境ではカーネルはホストから明示的に制御されるものであり，自動起動のような挙動はできない．カーネルに autorun 属性を付与すると，FPGA にプログラムが書き込まれた後にそのカーネルが自動で実行されるようになる．

Autorun カーネルは前述した Channel と組み合わせて利用するのが一般的である．なぜならば，autorun カーネルは自動起動されるためホストからパラメータを渡すことができず，入出力は Channel に限定されるからである．このプログラミングモデルは UNIX における daemon に近いものである．daemon は自動的にバックグラウンドで起動し，ソケットを通じて通信を行うことで仕事を行うが，autorun 属性を持つカーネルは FPGA プログラムの起動時に自動的に開始され，Channel を UNIX daemon におけるソケットのような入出力チャネルとして用いることで，daemon 的な使い方が可能となる．

Channel を用いて多数のカーネルを接続する設計では，1 つの計算を行うために多数のカーネル起動と終了待ちが発生する．しかしながら，それぞれの API 呼び出しはホスト上での実行オーバヘッドや，PCIe バスを通じて FPGA を操作するためのオーバヘッドがあり，性能を考慮するとできる限り回数を減らすことが望ましい．回数を減らすために autorun を活用する．アプリケーション中のカーネルをできる限り autorun にすることで，カーネル操作に関するコストを削減でき，また，制御のオーバヘッドが減るだけでなく，autorun カーネルはホストからの制御に必要な回路が必要なくなるため，autorun カーネルは回路リソースの節約にもなる．

5. ART on FPGA の実装

5.1 実装の概要

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．

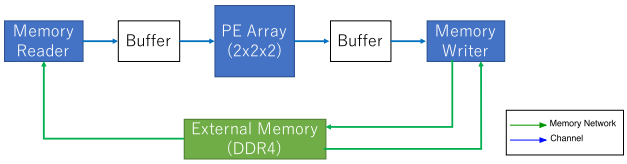

図 2　ART on FPGA 実装の概要
Fig. 2　Outline of ART implementation on FPGA.

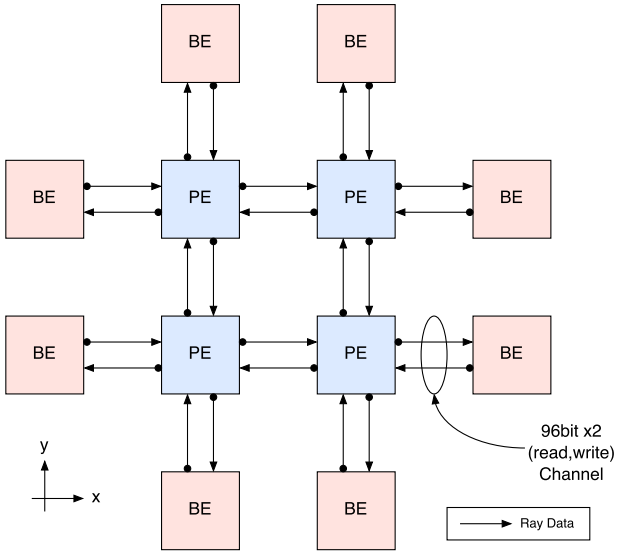

図 3　X，Y 次元における PE と BE の接続ネットワーク図
Fig. 3　Channel connections among PEs and BEs on the X and Y dimension.

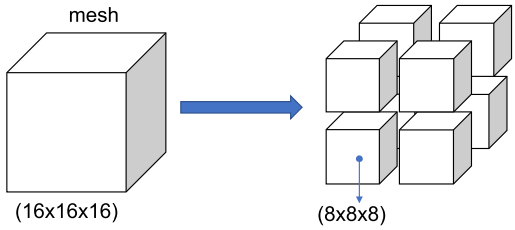

図 4　空間分割の概念図．大きな問題空間を小さく分割し，それぞれを PE に割り当てる
Fig. 4　Divide a large mesh into small meshes, and assign PEs to each of them.

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ランダムアクセスする必要があるため，Block Random Access Memory（BRAM）を用いて格納しており，それぞれの PE が演算用の BRAM を持つ．BRAM は FPGA 内部に実装されているメモリ（一般的に SRAM である）のことを指し，チップ内に一定のサイズのブロック単位で分散配置されている．BRAM は低レイテンシ・高バンド幅にランダムアクセスでき，非常に高性能であるが，外部メモリと比べて容量が少ない．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．なお，レイバッファを用いる処理は DDR を用いた計算に重要な処理を行う部分であるため，5.4 節で詳細に述べる．

5.2　Channel を用いた並列化

本節では，図 2 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 3 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．その手法は，オリジナルの CPU/GPU 版実装において用いられているノード間並列化手法の Multiple Wave Front（MWF）法 [10] を，FPGA 向けに改良し実装したものである．CPU や GPU におけるノード間通信よりも 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 4 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 3 は，図が煩雑になるのを避けるため，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 96 bit 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．

5.3　autorun を用いた最適化

図 2 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 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 2 から分かるように，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．

5.4　DDR によるレイバッファ

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 で解ける問題サイズが FPGA の BRAM の容量によって制限されてしまう．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 128 MB であり，現在の最先端の FPGA の BRAM 容量が高々 20〜30 MB 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．計算に使うデータは DDR メモリに格納し，BRAM をスクラッチパッドキャッシュのように扱う実装を行う．

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 5** に DDR 実装の疑似コードを示す．ただし，図 5 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．ここで，`ray_directions[]` はレイが向く方面（X 軸，Y 軸，Z 軸における正負の向き）を示す配列，`small_blocks[]` は前述した計算を行う小ブロック配列，`ipix_list(dir)` は Healpix ライブラリを用いて計算された角度のうち dir 方面を向いているものを返す関数を意味する．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し，Channel を用いて接続され，全体を構成している．

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

図 5　DDR 実装の疑似コード
Fig. 5　Pseudo code of the DDR implementation.

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．ここで，メッシュデータとは式 (1) にある source function パラメータ（光子の吸収量・拡散量）および積分の中間結果を格納する領域を指し $4 \times sizeof(float) \times \nu$ バイト必要であり，また，レイデータには放射強度（Intensity）の値が含まれ $sizeof(float) \times \nu$ バイトを占める．なお，ART 法は三次元空間をメッシュに分割し輻射輸送問題を解くため，メッシュデータのメモリ使用量は問題サイズの 3 乗に比例する．また，レイは問題の境界面のメッシュサイズに比例した本数生成されるため，最大で $12 \times N_{side}^2 \times N^2$（$N_{side}$ は Healpix における解像度パラメータ．$12 \times N_{side}^2$ 種のレイ角度が生成される）のメモリ量を必要とするが，入射するレイは放射強度が 0 であることと，計算がおわったデータは保持する必要がないことから，一度にメモリに展開する必要のあるレイデータは最大数より少なくなる．ただし，メッシュデータについてはデータサイズが 2 のべき乗の値にならない場合があるが，FPGA におけるメモリアクセスの効率を考慮し，その際はパディングを追加することで 2 のべき乗のサイズになるように調整を行う．

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 のよう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 5 の A），ART 法の計算を BRAM 上で行い（図 5 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 5 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あらかじめ分かっているため，計算中に次に必要なデータを FIFO バッファにあらかじ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 6** に示す．図 6 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．また，図では X 軸+かつ Y 軸+方向に向かうレイを計算する場合のみを例として示しているが，ART 法で扱うレイは様々な角度で入力されるため，X 軸，Y 軸，Z 軸それぞれに正負の向きがあり全 8 通り存在するしたがって，実際の計算におけるデータの流れも同様に 8 通りの組合せが存在する．図 6 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 5 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，たとえばブロック 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．

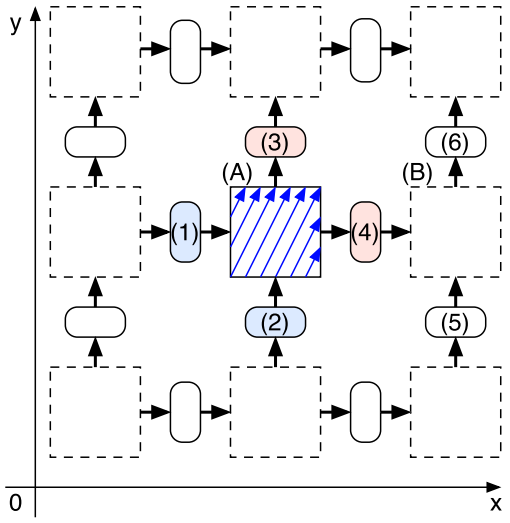

図 6　レイバッファの概念図．赤い箱は出力レイバッファ，青い箱は入力レイバッファ，青い矢印は計算するレイを表す．

Fig. 6　Overview of the ray buffer. Red boxes, blue boxes and blue arrows represent output ray buffers, input ray buffers, and rays to compute respectively.

6. ART on FPGA の性能・リソース使用量評価

6.1 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，本論文では 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．

表 1　評価環境

Table 1　Evaluation environment.

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)

表 2　リソース使用量とカーネルの動作周波数
Table 2　Resource utilizations and frequency of the kernels.

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

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 サイズの場合はすべ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} はすべての問題サイズで 8 に設定しており，異なる 768 方向のレイを生成する．ここでいう方向とは，球面座標系における偏角（θ, ϕ）の組合せが 768 種類という意味であって，レイの本数が計算全体で 768 本であるという意味ではない．レイ（平行光）が 768 種の角度で問題サイズに依存した本数分（N^2）生成されるため，ART 法の計算量は非常に多いものとなる．

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，値が大きいほど，演算性能が優れていることを意味する．

6.2　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 法の演算はすべて単精度で構成されており，指数関数を含めすべての演算は 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分かる．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すべてのデータを BRAM に格納することができるため，レイバッファが使われない．したがって，レイバッファに関するカーネルが削除され FPGA に実装されないためである．それぞれの PE に 8^3 の計算用の BRAM があり，本論文の実装では PE が 2^3 個あるため，問題サイズが 16^3 の場合はすべてのデータを計算用の BRAM に含めることができる．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 20 MHz 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 3　DDR メモリ使用量
Table 3　DDR memory utilization.

size	メッシュサイズ	レイバッファサイズ
(16, 16, 16)	0.25 MB	0 MB
(32, 32, 32)	2 MB	264 MB
(64, 64, 64)	16 MB	1,128 MB
(128, 128, 128)	128 MB	4,584 MB

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，すべての 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ですべての 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．

表 3 に DDR メモリの使用量を示す．“メッシュサイズ” はメッシュデータを格納するのに必要なメモリ量，“レイバッファサイズ” はレイバッファとして使用するメモリ量を表す．ただし，レイバッファのサイズについては最適化が不十分であり，改善の余地がある．最大値を元にメモリ量を決定しているが，計算のパラメータごとに使用量を詳細に見積もれればさらに削減できる可能性があると考えている．またレイの角度の数は最大で 1,024 まで対応できるようにメモリ量を計算している．レイバッファのサイズが

表 4　FPGA，CPU，GPU 間の性能比較の結果．数値の単位は “M mesh/sec”

Table 4　Performance comparison results among FPGA, CPU and GPU. The unit is “M mesh/sec”.

Size	CPU(14C)	CPU(28C)	P100	FPGA
(16, 16, 16)	112.4	77.2	105.3	1,282.8
(32, 32, 32)	158.9	183.4	490.4	1,165.2
(64, 64, 64)	175.0	227.2	1,041.4	1,111.0
(128, 128, 128)	95.4	165.0	1,116.1	1,133.5

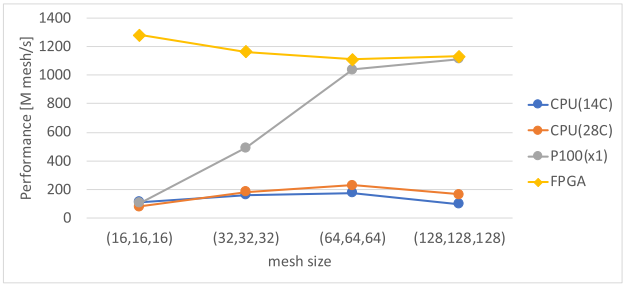

図 7　FPGA，CPU，GPU 間の性能比較の結果グラフ

Fig. 7　Graph of performance comparison results among FPGA, CPU and GPU.

十分最適化できていないものの，本実験で使用した FPGA の搭載メモリは 8 GB であり，これに対してメッシュデータのサイズは最大でも 128 MB と全体に対して小さい割合しか要求しないため，レイバッファ用の領域は十分に残されている．また，問題サイズ 16^3 のケースではレイバッファを用いないためサイズは 0 バイトである．

6.3　性能評価

表 4 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 1,283 M mesh/s，1,165 M mesh/s，1,111 M mesh/s，1,134 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 4.6 倍，FPGA 実装と比較して 4.9 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．また，2 つの CPU を用いても 1 CPU と比べて性能が倍になっていないのは，NUMA 環境であることを考慮せずにすべてのスレッドで同じメモリ領域を共有して計算しており，キャッシュコヒーレンシのオーバヘッドや，異なる CPU にあるメモリへのアクセスに起因するオーバヘッドがあるためと考えられる．

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 3,584 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．ART の CUDA 実装では，1 本のレイに対して 1 スレッドを割り当て，異なるレイ角度間はシーケンシャルに計算を行う．したがって，一度に動作するスレッド数は N^2 であり，問題サイズが小さい場合に多数の CUDA コアにスレッドを割り当てることが難しい．

GPU 実装とは異なり，FPGA 実装はすべ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 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分かる．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いえる．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，すでに述べたように 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．

7.　Next Step：FPGA 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 を組み合わせて利用する場合，GPUDirect for RDMA（GDR）[11] 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，通信のオーバヘッドは無視できない．

一方で，最新の FPGA（Intel Stratix10 など）は複数の高速通信機構（最大で 100 Gbps × 4）を持つ．FPGA で CPU を経由せずに直接通信する研究は文献 [12] や文献 [13] で行われており，FPGA の通信性能が非常に高いことが示されている．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

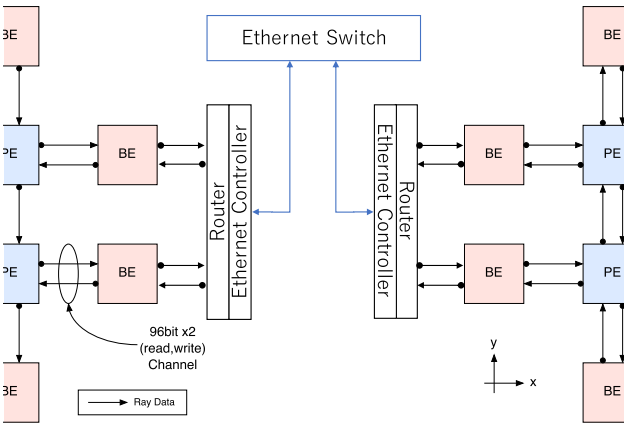

図 8　ART on FPGA を複数の FPGA で並列化する際の通信の概念図

Fig. 8　Communication overview of parallelized ART on FPGA.

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．我々はこれまでの研究 [14] で，OpenCL から FPGA が持つ通信機構を操作できることを示しており，今後この手法を拡張して ART 法を複数 FPGA で並列計算を行う予定である．

8.　おわりに

本研究では初期宇宙における天体形成シミュレーションで重要な役割を持つ輻射輸送を解く 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った．FPGA 実装の性能は CPU と比べて最大 4.9 倍，GPU と比べてほぼ同じ性能を達成した．FPGA の内蔵 BRAM だけではメモリ容量が足りず ARGOT で実際に計算したい問題サイズに対応できなないため，大容量の DDR メモリを併用することでこの問題を解決した．

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ると考えている．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ものの，SIMD のように各 PE の内部で 2 つ以上のメッシュを同時に計算すれば，演算性能に対するループの制御に要するリソースの割合を減らせると考えている．

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，さらに最大で DDR4-2400（19.2 GB/s peak）×4 の外部メモリを持つ．また，最大で 100 Gbps × 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．筑波大学計算科学研究センターでは次世代のスーパーコンピュータとして Cygnus の導入を予定している．Cygnus は 1 ノードあたり 2×Intel Xeon Gold CPU，4×NVIDIA V100 GPU，2×Intel Stratix10 FPGA が搭載され，各 FPGA は 100 Gbps×4 の外部リンクを持ち，2D トーラスによる FPGA 専用ネットワークが構築される予定である．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を実現する通信システムの開発を行っており，今後，その通信システムを ART on FPGA に適用する予定である．

謝辞　本研究は，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）次世代演算通信融合型スーパーコンピュータの開発およ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」の一環として実施したものです．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]　Intel: Intel® FPGA SDK for OpenCL™ (online), available from ⟨https://www.intel.co.jp/content/www/jp/ja/software/programmable/sdk-for-opencl/overview.html⟩.
[2]　XILINX: SDAccel 開発環境（オンライン），入手先 ⟨https://japan.xilinx.com/products/design-tools/software-zone/sdaccel.html⟩.
[3]　Zohouri, H.R., Maruyama, N., Smith, A., Matsuda, M. and Matsuoka, S.: Evaluating and Optimizing OpenCL

Kernels for High Performance Computing with FPGAs, *Proc. International Conference for High Performance Computing, Networking, Storage and Analysis, SC '16*, pp.35:1–35:12, IEEE Press (2016) (online), available from ⟨http://dl.acm.org/citation.cfm?id=3014904.3014951⟩ (2016).

[4] 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).

[5] 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).

[6] Kenter, T., Mahale, G., Alhaddad, S., Grynko, Y., Schmitt, C., Afzal, A., Hannig, F., Forstner, J. and Plessl, C.: OpenCL-Based FPGA Design to Accelerate the Nodal Discontinuous Galerkin Method for Unstructured Meshes, *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines* (*FCCM*), pp.189–196 (online), DOI: 10.1109/FCCM.2018.00037 (2018).

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

[9] 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 (2005) (online), available from ⟨http://stacks.iop.org/0004-637X/622/i=2/a=759⟩.

[10] 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).

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

[12] Weerasinghe, J., Polig, R., Abel, F. and Hagleitner, C.: Network-attached FPGAs for data center applications, *2016 International Conference on Field-Programmable Technology* (*FPT*), pp.36–43 (online), DOI: 10.1109/FPT.2016.7929186 (2016).

[13] 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, *Proc. 41st Annual International Symposium on Computer Architecuture* (*ISCA*), pp.13–24, IEEE Press (2014) (online), available from ⟨https://www.microsoft.com/en-us/research/publication/a-reconfigurable-fabric-for-accelerating-large-scale-datacenter-services/⟩.

[14] Ryohei, K., Yuma, O., Norihisa, F., Yoshiki, Y. and Taisuke, B.: OpenCL-ready High Speed FPGA Network for Reconfigurable High Performance Computing, *Proc. International Conference on High Performance Computing in Asia-Pacific Region, HPC Asia 2018*, pp.192–201, ACM (online), DOI: 10.1145/3149457.3149479 (2018).

藤田 典久　（正会員）

2016 年筑波大学システム情報工学研究科博士後期課程修了．博士（工学）．同年同大学計算科学研究センター研究員．高性能計算における演算加速装置およびそれらを接続する通信機構に関する研究に従事．情報処理学会会員．

小林 諒平　（正会員）

2011 年上智大学理工学部電気電子工学科卒業．2016 年東京工業大学大学院情報理工学研究科博士課程修了．博士（工学）．同年筑波大学計算科学研究センター助教．高性能計算のための FPGA の利活用技術についての研究に従事．電子情報通信学会，ACM 各会員．

山口 佳樹　（正会員）

筑波大学工学研究科（一貫制博士課程）修了．博士（工学）．現在，筑波大学システム情報系情報工学域准教授．研究テーマは，リコンフィギャラブルコンピューティングアーキテクチャおよびその応用に関する情報工学．

朴 泰祐　（正会員）

1960 年生．1984 年慶應義塾大学工学部電気工学科卒業．1990 年同大学大学院理工学研究科電気工学専攻後期博士課程修了．工学博士．1988 年慶應義塾大学理工学部物理学科助手．1992 年筑波大学電子･情報工学系講師，1995 年同助教授，2004 年同大学大学院システム情報工学研究科助教授，2005 年同教授，現在に至る．超並列計算機アーキテクチャ，ハイパフォーマンスコンピューティング，クラスタコンピューティング，GPU コンピューティングに関する研究に従事．筑波大学計算科学研究センターにおいて，超並列計算機 CP-PACS，PACS-CS，HA-PACS 等の研究開発を行う．2002 年および 2003 年度情報処理学会論文賞，2011 年 ACM ゴードンベル賞，2012 年度情報処理学会山下記念研究賞各受賞．2015 年度情報処理学会フェロー認証．IEEECS，ACM 各会員．本会フェロー．

吉川 耕司

京都大学大学院理学研究科博士課程修了．博士（理学）．筑波大学計算科学研究センター・講師．専門分野は宇宙論・宇宙大規模構造の数値シミュレーション．

安部 牧人

筑波大学大学院数理物質科学研究科博士後期課程修了．博士（理学）．筑波大学計算科学研究センター研究員．主な研究テーマは，銀河形成，星団形成，輻射輸送計算．

梅村 雅之

北海道大学大学院理学研究科博士課程修了理学博士．筑波大学計算科学研究センター教授．研究テーマは，宇宙構造，銀河，ブラックホール，アストロバイオロジーに関する計算宇宙物理学．