CB-001

HBM-FPGA による Diffuse Photon の大規模輻射輸送計算と課題
HBM-FPGA implementation of a large-scale radiative transfer simulation of diffuse photon and its subjects

古川 和輝 * Kazuki Furukawa*
横野 智也 † Tomoya Yokono†
山口 佳樹 ‡ Yoshiki Yamaguchi‡
吉川 耕司 § Kohji Yoshikawa§
藤田 典久 § Norihisa Fujita§
小林 諒平 § Ryohei Kobayashi§
朴 泰祐 § Taisuke Boku§
梅村 雅之 § Masayuki Umemura§

1. はじめに

本稿では，宇宙輻射輸送シミュレーション ARGOT (Accelerated Radiative transfer on Grids using Oct-Tree) [1] の高速化に向け，Diffuse Photon の輻射輸送計算を HBM-FPGA によって実装するための提案手法について議論する．

ARGOT は，筑波大学計算科学研究センターが天体現象解明のためのプロジェクトの一環として開発を進めている演算手法である．ARGOT では，一般的に計算コストが高く考慮されてこなかった，Diffuse Photon を扱う ART (Authentic Radiation Transfer) [2] 法による演算スキームを組み込んでいる．ART は GPU による演算加速を行っても ARGOT 全体の計算時間の 90% 以上を占めていることから，FPGA によって高速化することが検討されている．これは，GPU が苦手とするランダムに近いメモリアクセスを ART が要求するために，FPGA による自由度の高いデータフロー制御を行うことで飛躍的な高速化が見込めるためである．

関連研究 [3] では，HLS 設計により ART スキームを FPGA に移植し，その実装や性能比較について詳細に報告されている．先行研究 [4] では，ART を FPGA に移植し，CPU による実装と比較して最大 17.5 倍の大幅な性能向上を達成したことが示されている．しかし，この実装は FPGA の Block RAM を積極的に用いたためにシミュレーション空間に制約があったり，演算回路に縮小の余地があったりして，最低でも 1 辺 1,024 メッシュの 3 次元空間のシミュレーションを実現するには課題が残っていた．

これらの課題を解決するため，以下の 4 点について提案し，検証した結果を示す．

- 外部メモリの性能を最大限に引き出すメモリ割り当てとメモリアクセスを行い，輻射輸送計算で用いるメッシュデータを高速に読み出す方法を提案する．
- ART が要求するメモリアクセス性能の不足を補うために，メッシュデータを “Prism Memory” 上にバッファリングする機構を考案し，外部メモリに対するメモリアクセス数を $N_{\mathrm{ray/bundle}}$ 分の 1 に縮小させられることを示す．
- 演算回路の縮小により，関連研究に比べてメモリ空間と演算回路の並列数をそれぞれ 16^3，4 倍に拡張させられることを示す．
- 複数 FPGA による演算を考慮し，境界面で受け渡しされるデータを外部メモリに保存できる提案手法を示す．

* 筑波大学大学院システム情報工学研究群, Graduate School of Science and Technology, University of Tsukuba
† 筑波大学大学院システム情報工学研究科 (現:日本電信電話株式会社 ソフトウェアイノベーションセンタ), Graduate School of Systems and Information Engineering, University of Tsukuba (Presently with NTT Software Innovation Center, Nippon Telegraph and Telepehone Corporation)
‡ 筑波大学システム情報系, Faculty of Engineering, Information and Systems, University of Tsukuba
§ 筑波大学計算科学研究センター, Center for Computational Sciences, University of Tsukuba

2.Diffuse Photon の輻射輸送計算の演算加速

2.1. 輻射輸送方程式

宇宙にある多くの天体はガスでできていることから，輻射と物質の相互作用を理解することは宇宙で起こる物理現象を理解する上で大変重要である．点光源を扱う輻射輸送シミュレーションは効率の良いスキームが提案されているが，より正確に天体現象を再現するために Diffuse Photon による輻射も扱うことができるシミュレーション手法が望まれていた．

そこで，筑波大学計算科学研究センターでは，格子法によって空間に広がった光源 (Diffuse Photon) から放射される輻射を ray tracing によってシミュレーションする ART method を提案している．一般的な粒子速度に比べて光速は遥かに速く，宇宙輻射輸送シミュレーションでは系を定常的に扱うことができる．そのため，ベクトル $\hat{\boldsymbol{n}}$ 上を進む光線が距離 ΔL を進んだ時の光学的厚みを $\Delta\tau$，無次元の源泉関数を S_ν とすると，ART method で解くべき輻射輸送方程式は以下のようになる [5].

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

2.2.ART method の要求

ART method は図 1 のように屈折や散乱をせず，直線的に ray tracing を行う．この ray は，3 次元空間上である角度成分を持ち，メッシュの 1 境界面の面積を通過する光線を束ねて 1 本として定義される．多数の ray がメッシュを ray し，互いに干渉せず独立していると近似して輻射輸送計算を行うため，並列に計算が可能である．

ART method は ray が通過するメッシュすべてに対して計算を行うため，シミュレーションコード ARGOT の中で ART method が実行時間の 90% 以上を占めるほどに計算量が多い．そのため，先述のように GPU を使っ

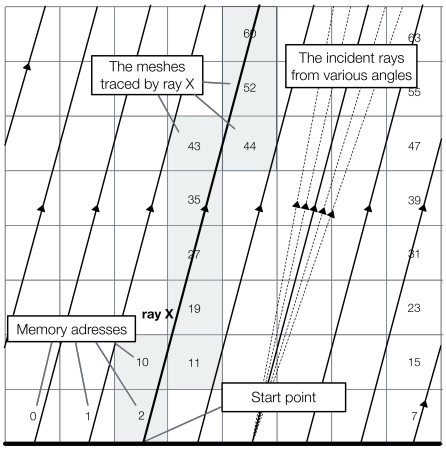

図 1　ART の ray tracing のイメージ：引用 [6] 図 1

た大規模並列演算による演算加速が検討されてきた．しかし，並列に輻射輸送計算した場合，計算時間を短縮することができる一方で，計算に必要なデータの回数も多い．メッシュデータとして読み込みデータ (source_func, absorption, Imesh, tau) と書き込みデータ (Imesh, tau) に加え，ray の初期値 (Iin) も考慮すると，合計 7 つの単精度浮動小数点数の Read-Write が必要となる．そのため，最内ループの 1 回の計算で 4 Byte の単精度浮動小数点数を最大 7 つ，つまり 28 Byte を外部メモリからの Read-Write を行うことが必要である．

シミュレーションとして有用性のある結果を得るためには，シミュレーション空間全体の 1 辺は 1024 メッシュ，ray の角度はそれと同じオーダーの数必要であり，これを満足させるメモリ容量は非常に大きい．そのため，単体のデバイスでシミュレーションを扱う際には空間全体を 8 つの空間に分割し，一辺 512 メッシュとするのが現実的である．

こういったことから，ART method は具体的に以下のような要求が存在する．

要求 1　1 デバイスで扱う部分空間の一辺が 512 メッシュであるとすると，メッシュデータの保存に必要なメモリ容量は約 6.4 [GB] である

要求 2　計算量が $O(N_{\mathrm{side}}^3 N_{\mathrm{angle}})$ 必要となる

2.3. 外部メモリに対する要求性能

図 2 は，ART を FPGA に実装すると仮定したときに，要求されるメモリ帯域幅と，メモリ各種類が応じることのできる最大の帯域幅を比較したものである．具体的などのような計算をして図 2 のような帯域幅を算出したかを示す．まず前提として，2.2 節で記述の通り，ART method において最内ループの計算 1 回につき 28 [Byte] のデータが必要である．これを FPGA 上浮動小数点演算器が動作可能と予想される 200 [MHz] で演算回路を動作すると仮定すると，単純に浮動小数点演算器で構成される Processing Element(PE)1 つにつき 5.6 [GB/s] 必要であるということになる．ただし，ART method で扱う輻射強度は，3 種類の原子を扱うため，1 本の ray を計算するためには演算器が 3 つ必要になる．

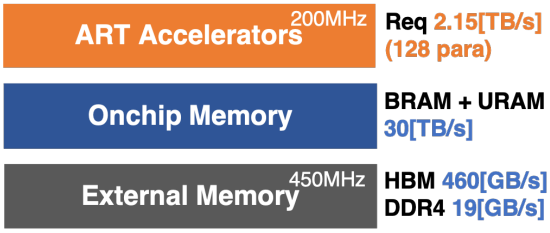

図 2　ART の要求メモリ性能とメモリ性能

また，図 2 でも見て取れるとおり，ART が要求するメモリ帯域幅は非常に大きく，現在市販される FPGA から利用できる DDR4 SDRAM や HBM といった大容量メモリの帯域幅はこの要求を満足させることができない．ここで，仮に先行研究 [7] の最大並列数と同じ 32 並列の PE を実装するものとすると，1 つの PE につき 1 種類の原子を演算できるため，$5.6[\mathrm{GB/s}] \times 3[\mathrm{atom\_types}] \times 32 = 537.6[\mathrm{GB/s}]$ のメモリ帯域幅を必要とすることになる．大容量メモリとして現在一般的に用いら入れているのは DDR4 SDRAM であるが，その帯域幅は一般的なもので 20 [GB/s] 程度となっており大幅に不足する．つまり，ART method でメッシュデータの格納に必要なメモリ容量を確保する必要があるものの，をたった 4 並列実装するだけで DDR4 SDRAM ではメモリ帯域幅が不足してしまうことになる．それどころか，Xilinx 社の高性能 FPGA から利用できる HBM の帯域幅の理論性能である 460 [GB/s] も大きく超えてしまう．

さらに，trace されたメッシュ同士は一見規則的なメモリアクセスではあるが，図 1 で示した番号のように，3 次元空間を扱うことに起因してメッシュに対応するメモリ空間のアドレスは必ずしも線型的に並んでおらず，ランダムライクアクセスが発生する．

このように，ART on FPGA には以下のような要求が追加される．

要求 3　メッシュに対応するメモリ空間のアドレスは必ずしもシーケンシャルに並んでおらず，メッシュデータのアクセスは不規則なものとなる

要求 4　PE 並列数を P として，$P \times 3[\mathrm{atom\_types}] \times 5.6[\mathrm{GB/s}]$ のメモリ帯域幅を要求する

2.4.FPGA 利用の利点

2.3 節で示したとおり，ART method のメッシュデータの保存には，それを保存する十分な容量と演算を遅延させない十分な帯域幅を持つ記憶機構が要求される．まず容量について見ると，一辺 512 メッシュの空間に対して 6.4 [GB] 以上が求められ，キャッシュやスクラッチパッドメモリと言った高速であっても小規模な機構での実現は困難である．一方，ART method で要求される転送速度は 2.15 [TB/s] に達するため，外部メモリを利用した単純な実装ではこの性能を満足できない．このような外部メモリの転送性能の制約は GPU も例外ではなく，ランダムメモリアクセスにも対応することが必要となる．

そこで，大量の小規模メモリを持ち高帯域幅を確保できる FPGA による演算加速が選択肢となる．FPGA の

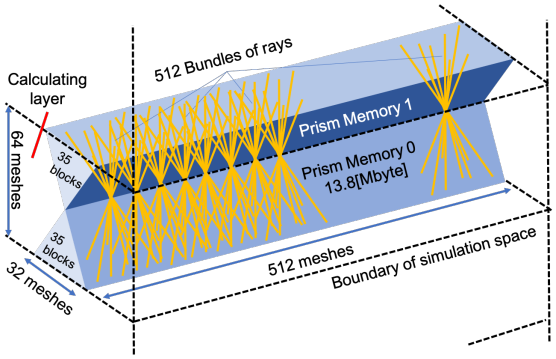

図 3　Prism Memory の概念図

オンチップメモリをバッファリング機構として利用し，メモリアクセス回数を削減し，局所性の高いメモリアクセスに変換することでメモリ性能によるボトルネックを解消することが可能となる．さらに，アプリケーション固有の PE を多数 FPGA に組み込むことによるハードウェア並列性を高くことで，宇宙輻射輸送シミュレーション ARGOT の ART method では他のアクセラレータよりも演算速度向上が期待される．

3.FPGA への実装

3.1. 外部メモリ利用の課題

2.3 節で示したとおり，ART を大規模 FPGA に実装すると PE の並列度を向上させることが可能となり，その結果，メモリに対する要求性能が非常に大きくなってしまう．また，ART の演算で発生するランダムライクアクセスは，外部メモリとして DRAM を利用するとその帯域幅にオーバーヘッドを作ってしまうことになる．このように，ART のメッシュデータを外部メモリに保持する回路を FPGA に実装する際には以下のような課題が存在する．

課題 1　ART on FPGA が要求するメモリ帯域幅は並列度 128 に対して 2.15 [TB/s] となり，HBM や DDR4 SDRAM では満たすことが困難である

課題 2　ART から外部メモリを直接利用すると，局所性が低いメモリアクセスが随所で発生し，読み出しのスループットが安定しない可能性がある

3.2.Prism Memory

本稿で，外部メモリからのデータの読み込みがボトルネックになることを防ぐための，ART 特有の Buffering 機構として triangular-Prism-shaped Memory を略して “Prism Memory” と名付けた．この Prism Memory は．シミュレーション空間を長い三角柱に切り出した形のメッシュデータを保存しておく Buffering 機構であり，特に ART をが等方性拡散する ray に基づいて計算を進めることを利用すると，大きく外部メモリへのアクセス回数を減らすことができる．

FPGA の場合も Onchip Memory をスクラッチパッドメモリとしてメッシュデータを Buffering する機構を作るとランダムアクセスによるメモリ帯域幅低下を軽減し，更にメモリアクセス回数を減少させることができる．

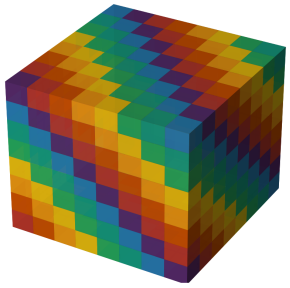

図 4　アドレッシング
同色の block が同じメモリバンク内に保存される

ここで，相互に近い角度の ray を，図 3 の黄色い線分のように $N_{\mathrm{ray/bundle}}$ 本束ねて扱うとする．外部メモリから一度データを読み込んだ Prism Memory からは，512 個の合同な束がメッシュデータを利用できる．つまり，1 束に含まれた ray の本数分だけデータをチップ内で再利用することができ，メッシュデータ外部メモリから再び読み込む必要がなくなる．そのため，外部メモリに対する要求帯域幅を実質的に $5.6 \times N_{\mathrm{para}}/N_{\mathrm{ray/bundle}}[\mathrm{GB/s}]$ まで抑えることができる．

本稿では，この Prism Memory 構築のために，Alveo U280 の大容量オンチップメモリ Ultra RAM(URAM) を用いる．そのため三角柱のサイズは，一辺 4 mesh の立方体を 1 つの block，要求されるシミュレーションのサイズを 8 分割した際の一辺を $N_{\mathrm{mesh/sim\_side}} = 512$ として考えて，$48[\mathrm{Byte/mesh}] \times 4^2 \times N_{\mathrm{block/cross\_section}} \times N_{\mathrm{mesh/sim\_side}} = 48 \times 16 \times 35 \times 512 = 13,763[\mathrm{KByte}]$ が必要となる．ただし，三角柱の断面に含まれる block の数を $N_{\mathrm{block/cross\_section}} = 35$ とした．一方で，URAM のサイズが $72[\mathrm{bitwide}] \times 4,096[\mathrm{deep}] \times 960 = 35,389[\mathrm{KByte}]$ であることから，少なくとも 2 つの Prism Memory を実装可能であるということがわかる．そのため，本稿では Prefetch 可能な Buffering 機構として 2 つ目の Prism Memory を利用することとした．

3.3. アドレッシングの工夫

課題 2 に対処するためには，外部メモリに対するアドレッシングを工夫する必要があった，外部メモリから読み出すメッシュデータは，Prism Memory に一時的に保存される．また，Prism Memory は 1 つの FPGA が扱うシミュレーション空間の一辺と同じ長さとなっている．さらに，すべての ray は等方性拡散することから，ray の初期地点はシミュレーション空間の各境界面となる．単純にメモリ空間を縦に分割することで外部メモリのポートの未使用率を下げることで帯域幅を確保しようとする方法では幾何学的に対称である演算に対して，必ずしもその対称性を維持することができない．そこで，図 4 のような分割方法を考案し，どの境界面からみても外部メモリに対するポートの数が対称となるようにした．

3.4. 境界面データの保存

ray が境界面に達した際には，シミュレーション空間の切れ目である場合は ray が持つデータを廃棄しても構わないが，シミュレーション空間を分割していて演算の途中である場合には，隣接する空間に ray が持つデータ

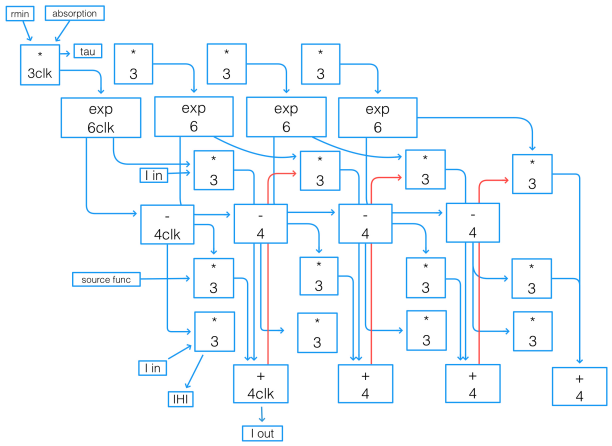

図 5　演算回路のデータフロー

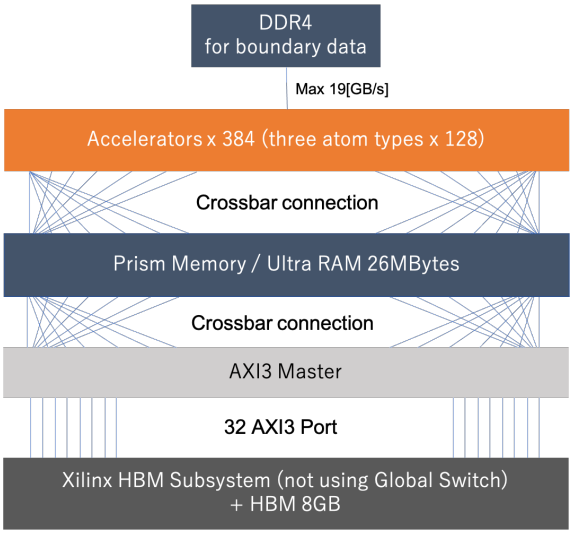

図 6　ブロック図の概要

を渡す必要がある．特に，一般的な高性能 FPGA ボードから利用できる外部メモリのメモリ容量は 1024^3 メッシュを保存するのに必要な 51[GB] を下回っており，空間を分割する必要がある．そのため，境界面のデータを保存する領域がメッシュデータとは別で必要となる．

しかし，境界面のデータは本稿のように 512^3 の空間を扱う場合には最大 9.6GB ほどの領域を必要とし，Onchip Memory や HBM では到底賄うことができない，そこで，本研究では境界面データを外部メモリである DDR4 SDRAM に保存することでこの問題を解決することとした．また，Prism Memory が高さ方向の空間を分割して扱うために，その中間データを保存するデバイスとして DDR4 SDRAM を用いることにした，境界面と高さ方向に分割したときの中間的な ray データの保存機構を共通化することによって，設計の際の煩雑さが軽減されるという利点もある．さらに，中間データをすべて一時的に保存する手段があるということは，複数 FPGA を利用して演算を行うことを考えたときに，各 FPGA 同士で ray データを受け渡しする際に，DDR4 SDRAM に対する DMA を行うだけで良いことになる．本稿が提案する境界面データの保存方式を利用することで，より大規模な輻射輸送計算を行うことを考えたときに，スムーズに FPGA 間通信部分を作ることができるようになる．

3.5. 演算回路の省略

すでに関連研究 [7] で実装された ART on FPGA では，ソフトウェアとして実装された ART を，浮動小数点演算器に置き換えることでストリーム演算を行った．そのため，実際には FPGA 上で並列に計算する必要のない演算器も存在している．特に，幾何学計算は演算負荷が大きい上に，すべての PE に対して同じ値を入力することから，改善が可能であることが分かった．そこで，本研究では最も演算負荷が大きい exp を含んだ輻射強度の計算のみを FPGA 上で計算させ，幾何学計算の結果を予めチップ内に格納しておく方法を取ることで，1 つの PE について FPGA 上のリソースを約 $\frac{1}{3}$ に減らすことができた．つまり，同じ回路規模の FPGA に対して，約 3 倍の PE を実装することができることになる．本研究で実装した Accelerator の演算器によるデータフローを図 5 で示す．

オンチップに格納するデータは，光路長データと次にトレースされるメッシュの方向の 2 種類であり，実際に必要となる総容量は，ray が立方体の対角線をトレースするときに通るメッシュの数を $\sqrt{3}N_{\mathrm{mesh/side}}$ として，$(32+2\times3)[\mathrm{bit}]\times\sqrt{3}N_{\mathrm{mesh/side}}\times(N_{\mathrm{angle}}/2)=6.47[\mathrm{MByte}]$ であり，オンチップメモリの中に実装することは可能である．

4. 実装結果

4.1. 設計回路

本研究で利用した Alveo U280 は 4GB の HBM スタックを 2 つ搭載しており，ART on FPGA 実装のためにこれを利用した．HBM の各スタックには 8 つの独立した Memory Channel(MC) が存在する．各 MC からは 2 つの 64bit Pseudo Channels がつながり，それぞれが 256MByte のメモリ領域のみにしかアクセスすることができない．そのため，Xilinx 社の HBM-FPGA には，32x32 のクロスバースイッチがハードウェア化され Xilinx HBM Subsystem として搭載されている．

しかし，Xilinx 社の HBM-FPGA に対するベンチマーク結果 [8] [9] や White Paper[10] からわかるように，この Subsystem は完全なものではなく，Mini Switch を跨いだメモリアクセスは帯域幅にオーバーヘッドが存在する．また，HBM は積層された DRAM を高帯域にチップに接続して実現したものであり，局所性の低いメモリアクセスが発生すると帯域幅が制限されることも示されている．

本稿において，図 6 のようなモジュールを作成し，実装を試みた．一番下には，HBM と Xilinx HBM Subsystem があり，前述のように Switch を利用しない実装とした．ただし，その代わりに HBM から Prism Memory に対するデータの移動のためにスイッチングが必要となる．さらに，128 個の Accelerator は ray が斜めに進行するために，Prism Memory に対して読み込むメモリ領域が変化していく．そのため，設計した回路としては，二重に

Crossbar 接続が存在している．

次に Accelerator を見てみると，原子を 3 種類扱うため，一本の ray に対して実質 3 個の演算器群を使用することになる．そのため，1 つの演算器が横方向で 4 メッシュを担当するとすると，一辺 512 メッシュに対して 128 個の Accelerator が必要であることになるが，3 種類の原子分を考えると細かく分けて 384 個必要であるということになる．

4.2. リソース使用率の考察

すでに記述したとおり，本稿で使用した FPGA は Alveo U280 であり，関連研究に比べて約 2 倍程度のロジックを有している．そのため，3.5 で記述したとおり，$\frac{1}{3}$ の回路規模で 1 つの Accelerator を作成することができるのであれば，関連研究 [7] の約 6 倍の並列度を実現することができるようになる．

表 1　FPGA ボードの概要 ([11][12] を参考に作成)

FPGA Board	本研究 Alveo U280	関連研究 [7] KCU1500
Chip	Virtex XCU280	Kintex XCKU115
Registers	2,607K	1,327K(FFs)
LUTs	1,204K	663K
BRAM+URAM (MB)	43.0	9.5(75.9Mb)
URAM Blocks	960	-
DSP Slices	9,607	5,520
On-Board RAM	16GB DDR4 x2	8GB DDR3 x2
HBM	4GB HBM stack x2	-

実際に実装を試みた結果，配線に失敗した．配線に失敗したデザインの，リソース使用率を表 2 に示す．まず最も使用率の高い DSP ユニットは，85% の量を使っていることがわかる，その一方で LUT の使用率は 55% と余裕があるため，今後浮動小数点演算器に使用する DSP の数を適切にパラメータを変更することによって減らし，今後，PE の数を増やすことで並列数を上げるということが考えられる．

表 2　リソース使用率 (failed route / 並列度 128)

Resource	Available	Used	Util%
CLB	162960	146483	89.89
LUT	1303680	714500	54.81
LUTRAM	600960	19623	3.27
Registers	2607360	11199	30.66
BRAM	2016	35	1.74
DSPs	9024	7683	85.14
URAM	960	688	71.67
BUFG	1008	17	1.69
MMCM	12	2	16.67

4.3. 理論性能

今回，次章で記述するような理由で，性能評価することができなかったため，実際に実装できた際に予想される理論性能を評価する．HBM の理論性能は 450MH z で動作させた際には，$450[\mathrm{MHz}] \times 32[\mathrm{port}] \times 256[\mathrm{bit}] = 460[\mathrm{GB/s}]$ の帯域幅を出すことができる．一方で [9] におけるパターン B の結果を考慮すると，Address 発行一回に付きデータが 16Beat 読み出されるときに，実測値において理論性能の約 85% の帯域幅を確保できる．これらを考慮すると，演算開始時における HBM

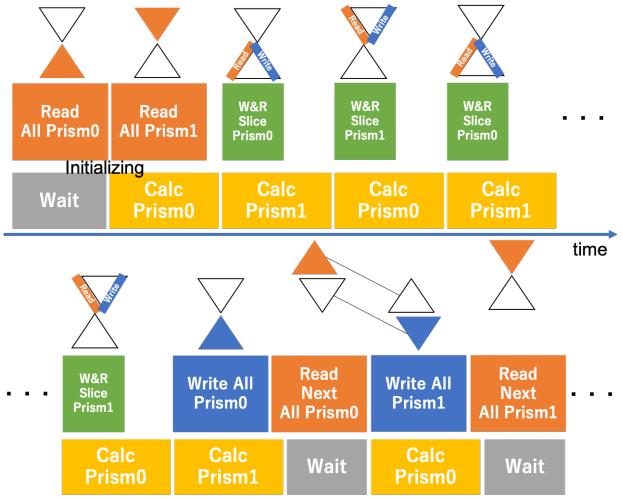

図 7　制御フロー

から Prism Memory に対する通信にかかる Clock cycle R は，$\mathrm{R}[\mathrm{clk}] = \mathrm{Size}_{\mathrm{prism}}[\mathrm{bit}]/256[\mathrm{bit/port}]/0.85 = 15,811[\mathrm{clk}]$ と計算できる．書き込み時は，メッシュデータが読み込み時の半分で構わないので，半分の Clock cycle，つまり $\mathrm{W}[\mathrm{clk}] = \frac{1}{2}\mathrm{R}$ で完了するとする．

また，Prism Memory は 2 つ用意することでメッシュデータの Prefetch が実現できることを考慮すると，図 7 のようにメッシュデータを保存していくことができる．1 つの Prism Memory のデータを使って計算できる ray の角度を 64 通りとすると，1 メッシュに付き 8 Clock cycle の計算を必要とすることを考慮して，$\mathrm{C}[\mathrm{clk}] = 8[\mathrm{clk\_cycle}] \times 32[\mathrm{mesh}] \times 64[\mathrm{angle}] \times 2 = 32,768[\mathrm{clk\_cycle}]$ が 1 つの Prism Memory を用いて計算にかかる Clock cycle となる．ただし，最後に 2 を掛けているのは，外部メモリから Prism Memory を一回読み込むと同じ角度の ray を 2×512 本計算することができるためである．一方で，1 つの Prism Memory に付き 2 倍の 128 本が計算できるようにすると，Clock cycle 数もそれに合わせて 2 倍になる．

これらより，理論的には $\mathrm{R+W} < \mathrm{C}[\mathrm{clk\_cycle}]$ となり，基本的には演算回路はメッシュデータの読み込みが完了するまで待たなければならない状況は発生しない．しかし，PE の並列数を 256 並列に上げたり，Read と Write の合計時間が Prism Memory の領域内の計算時間を上回ってしまったりすると，図 7 のように，Prism Memory が扱う空間の “層” が変化した際に Read と Write が同時に発生した際に，読み込み完了まで計算が止まってしまう．

図 7 は，計算と Prism Memory の読み書きのフローを示している．上段は，Prism Memory に対する初期データの入力時の制御フローであり，下段は Prism Memory の位置が上の段に変化する際に発生する Write と Read を含んだ制御フローを示している．実際に通信のみの時間で，計算が止まってしまう時間は，“Wait” の部分であり，$\mathrm{R+W} < \mathrm{C}[\mathrm{clk\_cycle}]$ となると，初期データの読み込み時以外は Wait がなくなるため，全体で $15,811[\mathrm{clk}]$ となる．ただし，この “Wait” が発生するのは，のうち

1/128 程度であるため，全体に占めるメモリアクセス完了の待ち状態も 1% 未満に収まる．

次に，128 個の PE による計算は，ストリーム演算を行うことから，ほとんど 1Clock につき 128 個の結果が出力されることになる．したがって，200 [MHz] で動作させることができるとすると，単純に掛け算して，$200[\mathrm{MHz}] \times 128 = 25.6[\mathrm{Gmesh/s}]$ 算出できる．

ここで，先行研究 [7] と比較する．先行研究では並列数 32 に対して本研究の手法では 128 並列と 4 倍の並列度向上を見込む．また, 先行研究では 2.7 [Gmesh/s] の計算性能のある回路の実装に成功しており，仮に本研究において通信完了を待たなければならない時間が，多く見積もって全体の 5% を占めるとしても，24.3 [Gmesh/s] であり，9 倍程度の性能向上を見込むことができる．最後にメモリ容量について見てみると，先行研究では Onchip Memory を 32^3 のシミュレーション空間で実装しているが，本研究においては，$(512/32)^3 = 16^3$ 倍の空間に拡張することができる．

5. 課題と展望

5.1. 課題

Alveo U280 FPGA のように, 大規模なチップに回路を実装する際には, Super Logic Region（SLR) と呼ばれ, 回路がいくつかの領域に分割されている．Alveo U280 は，HBM にアクセスできる SLR0 のほかに，SLR1，SLR2 が存在し，SLR2 が HBM から最も遠い．この SLR 同士をつなぐ配線は少なく，実際に実装を試みると本稿の実装では完了しない配線が数万程度発生した．そのため，SLR 同士を超えない配線で実装できる回路を設計する必要があることが分かった．

本稿で実装しようとした設計回路に存在するこの問題を解決するために，Accelerator-Prism Memory 間配線を減らして実装する必要があることが分かった．この部分の配線を減らすためには，Accelerator に Prism Memory を結合させ，ray データのみを Accelerator 同士でやり取りする方法が考えられる．

5.2. 結論と展望

本研究では，Diffuse Photon の輻射輸送計算の演算加速を FPGA で行い，HBM を使ってシミュレーション空間を拡張するために Prism Memory を用いた方法を提案した．実際に実装するためには，今後配線に配慮した設計を行う必要があるが，リソースは 128 並列で実装することが可能であることが分かった．本稿で提案した Prism Memory は，64 通りの角度を持った ray を一度に扱うことができることから，外部メモリへのアクセス回数を 64 分の 1 に減らすことができる. 仮に 128 並列で動作させることができた場合，Prism Memory がメッシュデータの Prefetch を行うため，演算回路側はほとんどメモリアクセスの完了を待つことなく計算を続行することができることから，200 [MHz] の理論値で 25.6 [Gmesh/s] を演算することができるようになる．

これを踏まえて，配線に配慮した設計を実装し，128 並列より大きな並列数の演算回路を組むことを目指す．そのために，まず Accelerator と Prism Memory を結合させた設計回路の開発を行う．

謝辞

本研究の一部は, JSPS 科研費 21H04869, 文科省 “次世代領域研究開発 (高性能汎用計算機高度利用事業)” における “次世代演算通信融合型スーパーコンピュー の助成を受けたものである．また Xilinx 社より “Xilinx University Program” を通じて開発ソフトウェアの支援を受けておりここに謝意を表する．

参考文献

[1] Takashi Okamoto, Kohji Yoshikawa, and Masayuki Umemura. ARGOT: accelerated radiative transfer on grids using oct-tree. *Monthly Notices of the Royal Astronomical Society*, Vol. 419, No. 4, pp. 2855–2866, February 2012.

[2] Satoshi Tanaka, Kohji Yoshikawa, Takashi Okamoto, and Kenji Hasegawa. 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, pp. 62(1–16), May 2015.

[3] 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, July 2019.

[4] 横野智也. FPGA を用いた宇宙輻射輸送シミュレーションの高速化. Master’s thesis, 筑波大学大学院 博士前期課程 システム情報工学研究科, March 2019.

[5] 梅村雅之, 福江純, 野村英子. 輻射輸送と輻射流体力学, 宇宙物理学の基礎, 第 3 巻. 日本評論社, December 2016.

[6] 古川和輝, 横野智也, 山口佳樹, 藤田典久, 小林諒平, 朴泰祐, 吉川耕司, 梅村雅之. 再結合光子の輻射輸送大規模計算に向けた HBM-FPGA 実装への考察. 情報科学技術フォーラム講演論文集 (FIT), 第 19 巻, pp. 21–26, September 2020.

[7] 横野智也, 山口佳樹, 藤田典久, 小林諒平, 朴泰祐, 吉川耕司, 安部牧人, 梅村雅之. 高位設計と低位設計の違いと FPGA 演算性能の関係について. 第 81 回全国大会講演論文集, 第 2019 巻, pp. 59–60, February 2019.

[8] Zeke Wang, Hongjing Huang, Jie Zhang, and Gustavo Alonso. Shuhai: Benchmarking High Bandwidth Memory On FPGAs. In *Proceedings of 2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 111–119, May 2020.

[9] 古川和輝, 横野智也, 山口佳樹, 藤田典久, 小林諒平, 朴泰祐, 吉川耕司, 梅村雅之. FPGA に組み込まれた HBM の効率的な利用とその考察. 電子情報通信学会 技術研究報告 (信学技報), 第 120 巻, pp. 30–35, September 2020.

[10] Mike Wissolik, Darren Zacher, Anthony Torza, and Brandon Day. White paper: Virtex UltraScale+ HBM FPGA: A Revolutionary Increase in Memory Performance. Technical Report WP-485 (v1.1), Xilinx, Inc., July 2019.

[11] Xilinx, Inc. *Alveo U280 Data Center Accelerator Card Data Sheet*, May 2020. DS963 (v1.3).

[12] Xilinx, Inc. *KCU1500 Board*, October 2018. UG1260 (v1.4).