ホールスラスタ・シミュレーションにおける割付処理の Altera SDK for OpenCL を用いた高速化

野田 裕之† 酒井 諒太郎† 宮島 敬明‡ 藤田 直行‡ 天野 英晴†

†慶應義塾大学 理工学部 情報工学科 〒223-0061 神奈川県横浜市港北区日吉 3-14-1
‡宇宙航空研究開発機構 航空技術部門 数値解析技術研究ユニット 〒118-8522 調布市深大寺東町 7-44-1
E-mail: †‡cfd@am.ics.keio.ac.jp

あらまし Full Particle-In-Cell(Full-PIC)法は、電気推進エンジンの一種であるホールスラスタの研究開発で用いられる数値シミュレーション手法である。Full-PIC 法は、イオン・中性子・電子の全てを粒子として扱うため、粒子を流体モデルに近似させる他の手法と比べ高精度であるが、計算コストが高いことが知られている。宇宙航空研究開発機構(JAXA)が研究開発を進めるホールスラスタ用シミュレーション・コード(NSRU-Full-PIC)は、Full-PIC 法を用いており、処理に膨大な時間を要することが問題である。NSRU-Full-PIC において、特に計算負荷の高い処理は割付処理である。割付処理は、粒子の情報をセル四隅へ割り付ける処理であり、Read After Write(RAW)ハザードを引き起こすために並列化を阻害する要因となる。本研究では、Altera 社のミッドレンジ SoC FPGA である Arria 10 SoC を複数用いたホールスラスタ・シミュレーション用マルチ FPGA クラスタ構築の第一段階として、Altera 社が提供する FPGA 向け OpenCL ベース高位合成環境である Altera SDK for OpenCL を用いて、NSRU-Full-PIC において特に高負荷である割付処理を Arria 10 SoC にオフロードし高速化を検討する。本実装では、RAW ハザードを回避しつつ効率のよい処理を行うため､粒子がもつ情報をセル単位でリダクションの形でまとめてセル四隅へ割り付ける。オフロード結果を CPU での実行結果と比較したところ、ARM Cortex-A9 1.5GHz と比較して最大で約 11.4 倍の高速化を達成し、Xeon E5-2667 0 2.9GHz と比較して最大で約 2.1 倍の高速化を達成した。

キーワード ホールスラスタ、Particle-In-Cell、FPGA、Altera SDK for OpenCL、OpenCL、高位合成

1. はじめに

電気エネルギーを利用して推力を得る電気推進エンジンとして、ホールスラスタやイオンスラスタが知られている。電気推進エンジンは、ロケットの打ち上げ等に利用される化学推進エンジンと比べ、得られる推力は低いが、燃費の指標である比推力が高い。ホールスラスタは、小惑星探査機「はやぶさ」等で搭載されたイオンスラスタと比べ、低比推力領域において推力効率が高いこと、機構が簡単であるために軽量化が容易であること、比較的推力レベルが大きいことが知られる。以上の理由から、ホールスラスタは人工衛星の軌道間輸送や姿勢制御といった地球近傍ミッションに適しているとされている[1]。

ホールスラスタの研究開発において、数値シミュレーションは必須である。これは、数値シミュレーションを用いた試験が実機試験と比べて低コスト・高精度だからである。Full Particle-In-Cell(Full-PIC)法は、ホールスラスタの研究開発で用いられる数値シミュレーション手法である。Full-PIC 法は、イオン・中性子・電子の全てを粒子として扱うため、粒子を流体モデルに近似させる他の手法と比べ高精度であるが、計算コストが高いことが問題とされている[2][3][4[5]。宇宙航空研究開発機構(JAXA)が開発を進めるシミュレーション・コード(NSRU-Full-PIC)は、Full-PIC 法を用いており、処理に膨大な時間を要する。NSRU-Full-PIC において、計算負荷の高い処理として割付処理が知られる。割付処理は、粒子が持つ情報をセルの四隅に割り付ける処理である。ひとつのセルには複数の粒子が存在するため、粒子単位で並列処理を行うと Read After Write(RAW)ハザードが発生する。これが割付処理の並列化を阻害する要因となる。

先行研究として､NSRU-Full-PIC の割付処理を含む処理部を Graphics Processing Unit(GPU)にオフロードすることで高速化を検討したものが存在する[6]。この先行研究では、割付処理に排他処理の一種であるアトミック処理を適用した。また、Field-Programmable Gate Array(FPGA)と CPU が密に結合した Xilinx 社のローエンド System on Chip(SoC)である Zynq に NSRU-Full-PIC の高負荷部をオフロードし、高速化を行った先行研究が存在する[7]。この先行研究では、電場・イオンの相互作用を計算する処理を Zynq にオフロードし、NSRU-Full-PIC の高速化を行った。

先行研究[6]では、NSRU-Full-PIC における割付処理部にアトミック処理を適用した。しかし、アトミック処理はあるスレッドがある値に対して処理を行う間、他のスレッドはその値に対して処理を行うことが出来ないために非効率である。よって、この先行研究では、割付処理部の十分な高速化には至らなかった。また､先行研究[7]では、

NSRU-Full-PIC の実装に Zynq を用いたが、Zynq はローエンド SoC FPGA であるために FPGA リソースに限りがあり、オフロード範囲が限られた。また、この先行研究では、割付処理をオフロード対象として扱わなかった。

Altera 社の FPGA 向け OpenCL ベース設計環境である Altera SDK for OpenCL は、ヘテロジニアスな環境で並列プログラミングが可能な高位合成フレームワークであり、短期間で高性能な FPGA 回路の記述が可能である。また、Altera 社のミッドレンジ SoC FPGA である Arria 10 SoC は、ARM プロセッサを内蔵することで簡単に Linux が動作し、浮動小数演算用の DSP ブロックを持つことから演算性能にも優れる。本研究では、Arria 10 SoC を複数用いたホールスラスタ・シミュレーション用マルチ FPGA クラスタ構築の第一段階として、Altera SDK for OpenCL を用いて NSRU-Full-PIC において特に高負荷である割付処理を Arria 10 SoC にオフロードし、割付処理の高速化を検討する。本実装では、RAW ハザードを回避しつつ効率のよい処理を行うため、粒子がもつ情報をセル単位でリダクションの形でまとめてセル四隅へ割り付ける。

本論文の構成を以下に示す。第 2 章では、ホールスラスタの動作原理、Full-PIC 法および NSRU-Full-PIC における割付処理の詳細を述べる。第 3 章では、Altera SDK for OpenCL の詳細を述べる。第 4 章では、Altera SDK for OpenCL を用いた Arria 10 SoC へのオフロードの詳細を述べる。第 5 章では、オフロード結果と CPU 実行での結果を比較し、評価を行う。最後に、第 6 章を本論文のまとめとする。

2. Full-PIC 法を用いたホールスラスタ・シミュレーション

2.1. ホールスラスタ

図 1 に、ホールスラスタの動作原理の概略を示す。円環状のプラズマ加速部(チャネル)において半径方向の外部磁場と軸方向の電場を印加することで、陰極からチャネルに流入した電子が周方向に円周運動を行う。電子の円周運動は、円周方向にホール電流を発生させる。陽極からチャネルに流入した推進剤(キセノンなどの希ガス)は、円周運動を行う電子と衝突することでプラズマ化する。そのプラズマのうち、電子はホール電流と半径方向の磁場によって発生するローレンツ力によって軸方向の電場を打ち消す動きを妨げられ、結果として軸方向の電場は維持される。この軸方向の電場によって、プラズマ内のイオンのみが加速され、スラスタ外に排出される。スラスタ自身は、プラズマ内の電子が軸方向の電場を打ち消す動きを阻害するように発生するローレンツ

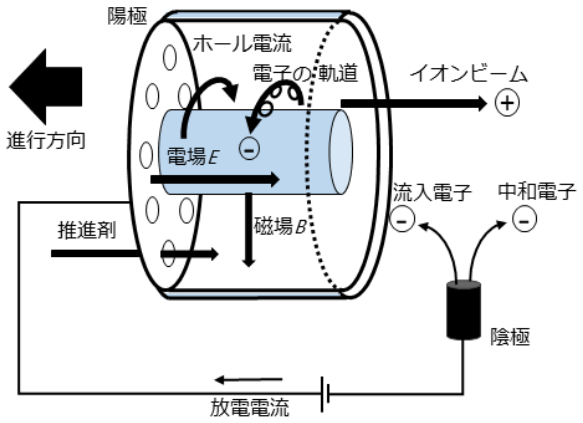

図 1. ホールスラスタの動作原理

力の反力を得て、推力を得る。陰極から流入した電子は、チャネル内を陽極に向かって拡散するため、スラスタ内部は準中性に保たれる。また、陰極から供給される電子の一部がスラスタ外に排出されたイオンを中和することで、スラスタ内部が負に帯電することを防ぐ。

2.2. Full-PIC 法

ホールスラスタで用いられるシミュレーション手法は、大別して以下の 2 種類が存在する。そのうち、PIC 法はさらに 2 種類に分類される。

- 完全流体モデル
- Partial-In-Cell(PIC)法

完全流体モデルは、プラズマ内の粒子・場を流体として扱い、計算領域を分割して計算する手法である。一方、PIC 法は、プラズマを粒子と場に分割し、粒子については運動方程式や衝突等の計算を解き、場については空間格子(セル)を用いて計算を行う。PIC 法は連続体の運動を有限数の粒子の運動として離散化する粒子法の一種であるが、同じく粒子法に分類される Moving Particle Semi-implicit(MPS) 法 や Smoothing Particle Hydrodynamics(SPH)法といった、粒子のみをあつかう他の手法とは処理が大きく異なる[6]。PIC 法のうち、Hybrid-PIC 法はプラズマ内のイオン・中性子を粒子として扱い、電子については流体として近似する。Full-PIC 法は、イオン・中性子・電子の全てを粒子として扱う。Full-PIC 法は流体モデルを用いないため高精度にシミュレーションを行うことが可能だが、計算コストの高さが問題とされている[2][3][4][5]。

2.3. NSRU-Full-PIC における割付処理

NSRU-Full-PIC は、JAXA が開発を進めるホールスラスタ用シミュレーション・コードであり、約 7000 行の Fortran90 で記述された CPU 用コードである。本研究では、このコードを研究対象とする。NSRU-Full-PIC は大規模なループ構造を持

ち、粒子と場の物理量を交互に更新し、タイムステップを進展させることで、スラスタ内部のプラズマのふるまいと静電場の時間発展を解く。NSRU-Full-PIC のフローを図 2 に示す。

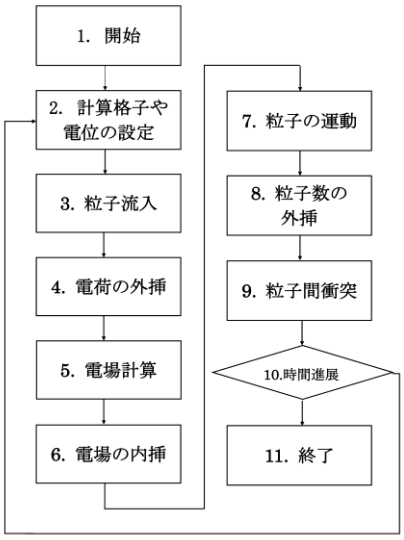

図 2. NSRU-Full-PIC のフロー

割付処理は、粒子が持つ情報を、粒子が属するセルの四隅に加算する処理である。割付処理は、図 2 においてステップ 4 の「電荷の外挿」処理と、ステップ 9 の「粒子数の更新」処理に含まれる。NSRU-Full-PIC においては、ひとつのセル内に複数の粒子が存在するため、複数の粒子に対して割付処理を行う。図 3 に、ひとつのセル内で複数の粒子に対して割付処理を行う様子を示す。ここでは、二つの粒子 p1 および p2 が持つ情報をそれぞれ *p*1、*p*2 とし、これらの情報を GP[0]-GP[3]に加算し、その値を更新している。ここで、GP[0]に注目し、粒子 p1 の GP[0]への情報の読み書きが完了する前に粒子 p2 がその値に対して読み込みを行うと仮定すると、粒子 p2 は GP[0]の古い値に対して読み書きを行う。その結果、Read After Write (RAW)ハザードが発生して格子点の値は正しく更新されない。図 4 に、GP[0]において RAW ハザードが発生する様子を示す。したがって、割付処理の並列化には RAW ハザードの回避が必要である。

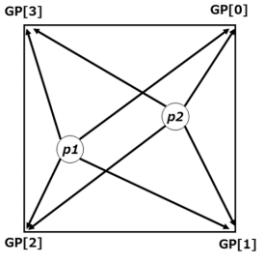

図 3. 割付処理

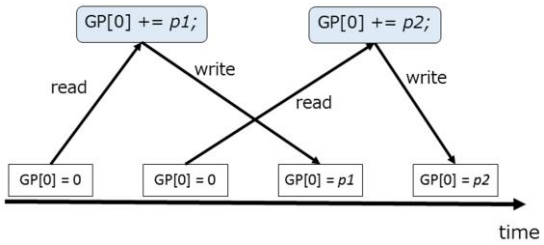

図 4. 割付処理における RAW ハザード

2.4. NSRU-Full-PIC のプロファイリング

NSRU-Full-PIC における各フェイズの処理時間を比較するため、CPU を用いて NSRU-Full-PIC のプロファイリングを行った。評価には、京都大学学術情報メディアセンターが所有するスーパーコンピュータ、CRAY XE6[8] 128 プロセスを用いた。評価環境を以下に示す。

- CPU: AMD Opteron 6200 系 2.5GHz
- Memory: 64GB/node
- OS: Cray Compute Node Linux
- Process: 128

図 5 に、CRAY XE6 128 プロセスによる NSRU-Full-PIC の全体評価の結果を示す。ここで、「MPI 通信」は更新した粒子の情報を全セル共有するための処理である。図 5 によると、割付処理を含むフェイズが全処理時間に対して占める割合は約 4 割となった。これより、割付処理は NSRU-Full-PIC において高負荷な処理であることがわかる。

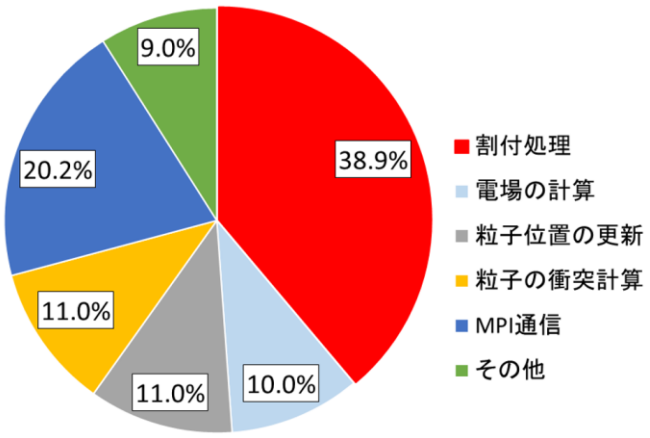

図 5. NSRU-Full-PIC のプロファイリング

3. Altera SDK for OpenCL

Altera SDK for OpenCL は、Altera 社が提供する FPGA 向け OpenCL ベース高位合成環境であり、短期間で高性能な FPGA 回路の記述が可能である。OpenCL は、CPU・GPU・FPGA などの様々なプロセッサによって構成されるマルチプロセッサ環境において利用可能な、並列プログラミング用フ

レームワークである。Altera SDK for OpenCL では、演算用プロセッサ(OpenCL デバイス)の動作を決定するカーネルコードと、制御用プロセッサ(ホスト)の動作を決定するホストコードの 2 種類を用いる。

カーネルコードおよびホストコードは、OpenCL C 言語と OpenCL ランタイム API の二つの仕様を用いて記述する。OpenCL C 言語は、C 言語を拡張した並列プログラミング言語であり、カーネルコードにおいて用いられる。OpenCL ランタイム API は、制御用プロセッサ(ホスト)を動作させるホストコードの記述に用いられる Application Programming Interface(API)である。ホストコードは C++ベースで記述し、その中で OpenCL ランタイム API が用いられる。

また、Altera SDK for OpenCL では、カーネルとホストをつなぐ PCIe や外部メモリとのインタフェースなど周辺回路をサポートする Board Support Package(BSP)が提供されている。これを用いることで、ユーザは周辺インタフェースの設計をせずとも FPGA を動作させることが可能となる。

図 5 に、OpenCL のメモリモデルを示す。OpenCL には、work-group と work-item の 2 階層のスレッド空間が存在する[9]。work-group は work-item の集合であり、カーネルは work-item 単位で処理を実行する。グローバルメモリとコンスタントメモリは、全 work-group からアクセス可能なメモリであり、グローバルメモリは読み書き可能であるのに対してコンスタントメモリは読み出し専用である。一方、ローカルメモリは work-group 固有のメモリで、プライベートメモリは work-item 固有のメモリである。

Altera SDK for OpenCL には、2 種類の並列モデルが存在する[9]。1 つ目が Single work-item kernel であり、2 つ目が NDRange kernel である。Single work-item kernel はタスク並列モデルに分類される。これは work-group および work-item がそれぞれ 1 つしか存在しないため、逐次プログラミングと同じようにカーネルコードの記述が可能である。コンパイラがカーネルコード内の並列性を抽出し、ループをパイプライン化する。一方、NDRange kernel はデータ並列モデルに対応する。work-group および work-item は複数存在する。各 work-item はスレッド空間に対応し、パイプライン実行される。NDRange kernel では、複数の work-item に対してカーネルパイプラインの多重化やベクトル化を指定できる。これらの処理を指定することでスループットの向上が可能であるが、リソース使用量の増加を招く。

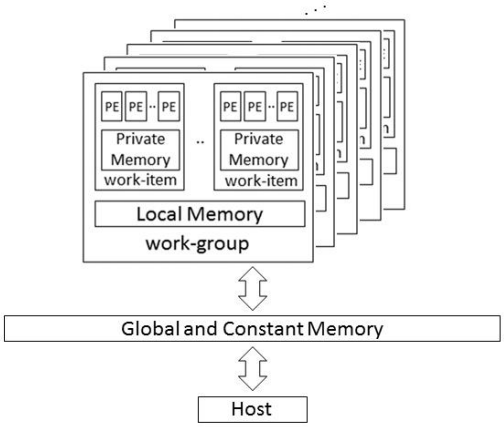

図 5. OpenCL のメモリモデル

4. 割付処理の Arria 10 SoC へのオフロード

第 2 章で記述した通り、NSRU-Full-PIC において割付処理を含むフェイズは高負荷である。また、割付処理の並列化には RAW ハザードの回避が必要である。本研究では、Altera SDK for OpenCL を用いて、Altera 社のミッドエンド SoC FPGA である Arria 10 SoC に割付処理をオフロードすることで、割付処理の高速化を行う。

4.1. Arria 10 SoC

Arria 10 SoC は、Altera 社が提供するミッドレンジ SoC FPGA である[10]。Arria 10 SoC は、デュアルコア ARM Cortex-A9 MPCore を備えた Hard Processor System(HPS)と、FPGA ロジックから構成される。図 7 に、Arria 10 SoC の外観を示す。HPS 部は、CPU を含むプロセッサ・ユニット、キャッシュ、オンチップ・メモリ、外部メモリ・インタフェース、タイマおよび I/O、セキュリティ、通信インタフェース・コントローラ、AXI インタコネクトから構成される。また、Arria 10 SoC の FPGA ロジック部にはハード化された単精度浮動小数点演算用の Digital Signal Processing(DSP)が搭載されている。このため、Arria 10 SoC は Zynq などのローエンド SoC と比較して演算性能が高い。

図 7. Arria 10 SoC の概観

4.2. 実装

図 8 に、本実装の概要を示す。本実装では、RAW ハザードを回避しつつ効率のよい処理を行うため、粒子単位で並列化を行わず、セル単位で並列化を行う。つまり、粒子がもつ情報をセル単位でリダクションの形でまとめて、その値をセル四隅へ値を加算する。図 8 では、一時変数を用いてリダクションの形でもって情報をまとめ、それを GP[0]に加算することで GP[0]の値を一度に更新している。GP[1]-GP[3]に対しても、同様の処理を行う。

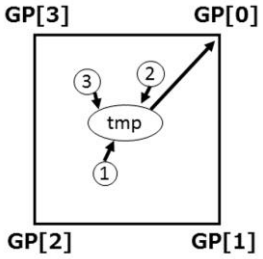

図 8. 本実装の概要

図 9 に、本実装におけるリダクション演算部の処理の概要を示す。本実装のカーネルコードでは、Single work-item kernel を用いる。要素を一つにまとめる処理を行うループに対して、#pragma unroll 修飾子を用いてアンローリングを行い、フィードバックループを全て展開することで、レイテンシを縮める。また、そのリダクション回路に入力データを次々と投入することで、効率のよいパイプライン処理を行う。

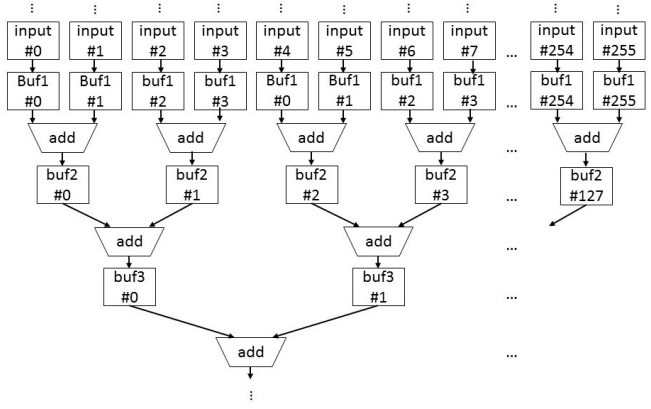

図 9. リダクション演算部の処理の概要

5. 評価

オフロード結果を、Arria 10 SoC 上の ARM Cortex-A9 1.5GHz および Xeon E5-2667 0 2.90GHz の 2 種類の CPU 上での実行結果と比較し、評価を行った。評価環境を表 1 に示す。

図 10 に、インプットデータとして扱うデータサイズが 100MB の時の、Arria 10 SoC での処理時間と 2 種類の CPU 実行の処理時間の比較を示す。

表 1. 評価環境

Arria 10 SoC	FPGA	SX660(10AS660)
	CPU	ARM Cortex-A9 @ 1.5GHz
	Host Memory	1GB
	OS	Linaro Ubuntu 14.04.5 LTS
	Kernel	4.1.22-ltsi
	Compiler(host code)	GCC 4.8.3 added -O3 option
	Compiler(kernel code)	Altera SDK for OpenCL 64-bit Offline Compiler 16.0.2 222 added -O3, -v and -g option
Comparison 1	CPU	ARM Cortex-A9 @ 1.5GHz
	Host Memory	1GB
	OS	Linaro Ubuntu 14.04.5 LTS
	Compiler	GCC 4.8.4 added -O3 option
Comparison 2	CPU	Intel Xeon E5-2667 0 @ 2.9GHz
	Host Memory	128GB
	OS	CentOS 6.5
	Compiler	GCC 4.4.7 added -O3 option

結果として、本実装は ARM Cortex-A9 1.5GHz での実行と比較して約 11.4 倍の高速化を達成し、Intel Xeon E5-2667 0 2.9GHz での実行と比較して約 2.1 倍の高速化を達成した。

また、図 11 にデータサイズを 1KB、10KB、100KB、1MB、10MB、100MB に変化させた時の実行速度向上比の遷移を示す。データサイズが 10KB 以下の処理では実行速度向上比が CPU 実行に対して 1.00 を下回るが、データサイズが 100KB 以上の処理では 2 種類の CPU を上回る処理速度を達成した。また、データサイズが上がるにつれて実行速度向上比が頭打ちになる。この理由は、インプットデータをホストから FPGA のオンチップメモリに転送する際のメモリバンド幅がボトルネックとなり、実行速度の向上を妨げていることが原因であると考えられる。

表 2 に、本実装で使用した Arria 10 SoC の FPGA リソース量を示す。本実装は Altera SDK for OpenCL における single work-item kernel を用いたため、カーネルパイプラインの多重化やベクトル化を行わず、最低限の FPGA リソース使用率で高性能な演算を行うことが出来た。

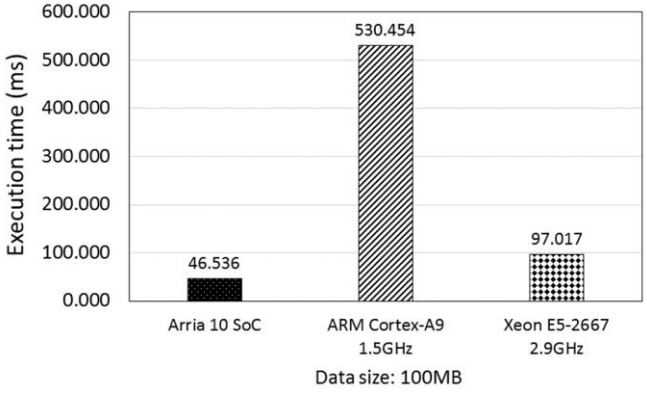

図 10. 100MB での実行時間の比較

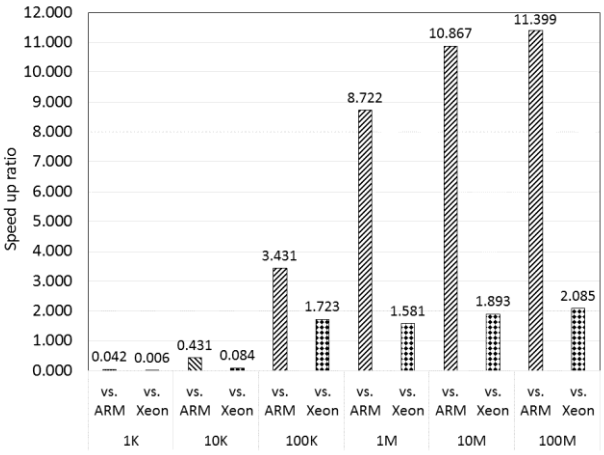

図 11. 実行速度向上比の遷移

表 2. 本実装における FPGA リソース使用量

ALUTs	Registers	Memory blocks	DSPs
4%	5%	8%	16%

6. 結論

本研究では、Arria 10 SoC を複数用いたホールスラスタ・シミュレーション用マルチ FPGA クラスタ構築を見据え、NSRU-Full-PIC において特に高負荷である割付処理の高速化のため、Altera SDK for OpenCL を用いて Arria 10 SoC に割付処理をオフロードした。本実装においては、RAW ハザードを回避しつつ効率のよい演算を行うため、セル単位での並列化を行った。

オフロードの結果、ARM Cortex-A9 1.5GHz での実行と比較して最大で約 11.4 倍の高速化を達成し、Xeon E5-2667 0 2.9GHz での実行と比較して最大で約 2.1 倍の高速化を達成した。

今後の課題として、第一に複数のセルを考慮して割付処理を行うことを挙げる。セルの格子点は合計 4 つのセルに囲まれるため、複数のセルを考慮して本研究で提案した割付処理を行うと、RAW ハザードが発生する可能性がある。これを回避するには、格子点を含む 4 つのセルに対して物質量ごとにリダクションの形でまとめる手法が考えられる。第二に、本研究の実装を NSRU-Full-PIC へ組み込み、全体的な性能評価を行うことを挙げる。本実装では、メモリバンド幅がボトルネックとなったと考えられる。そのため、今後はメモリバンド幅に合わせて回路を設計することが望ましい。

謝辞

本研究は JSPS 科研費 基盤研究 (C) 16K00078 の助成による。

文　献

[1] 栗木恭一, 荒川義博. 電気推進ロケット入門. 東京大学出版会, 2003.

[2] Yokota Shigeru, Komurasaki Kimiya, and Arakawa Yoshihiro, Plasma Density Fluctuation Inside a Hollow Anode in an Anode-layer Hall Thruser. In 42th Joint Propulsion Conference and Exhibit, AIAA-2006-5170, 2006.

[3] Hirakawa Miharu ,“Electron Transport Mechanism in a Hall Thruster", IEPC-97-021 (1997).

[4] Justin M. Fox, “Advances in Fully-Kinetic PIC Simulation of a Near-Vacuum Hall Thruster and Other Plasma Systems", PhD thesis (2007).

[5] James Joseph Szabo, “Fully Kinetic Numerical Modeling of a Plasma Thruster ", PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA (2001).

[6] Takaaki Miyajima, Shinatora Cho, and Naoyuki Fujita. A study of gpu acceleration of “source” part in hall-thruster simulation. In IEICE Tech. Rep., Vol. 115 of CPSY2015-62, pp.7-12, Dec. 2015.

[7] R. Sakai, N. Sugimoto, T. Miyajima, N. Fujita, and H. Amano. Acceleration of full-pic simulation on a cpu-fpga tightly coupled environment. In 2016 IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSOC), pp. 8–14, Sept 2016.

[8] システム A (愛称:Camphor) – スーパーコンピュータシステム– 京都大学情報環境機構 http://www.iimc.kyoto-u.ac.jp/ja/services/comp/supercomputer/.2016/12/29/20:50.

[9] Intel Corporation. Intel FPGA SDK for OpenCL Programming Guide: https://www.altera.com/en_US/pdfs/literature/hb/opencl-sdk/aocl_programming_guide.pdf. 2017/01/29/13:00.

[10] Intel Corporation. Arria 10 SoC - Features:.https://www.altera.com/products/soc/portfolio/arria-10-soc/features.html. 2017/01/29/14:04.