Vol.2018-HPC-163 No.26
2018/3/2

OpenCLを用いたFPGAによる階層型行列計算

塙 敏博[1,a)]　伊田 明弘[1]　星野 哲也[1]

概要：近年，単精度浮動小数点の演算器を多数内蔵した FPGA(Field Programmable Gate Array) が登場している．加えて，OpenCL を用いた FPGA 実装が可能になったことで，FPGA を用いた HPC アプリケーションが比較的容易に，また効率的に実現できるような環境が整ってきた．
本研究では，階層型行列法ライブラリである $\mathcal{H}$ACApK について，OpenCL を用いた FPGA 向けの実装について検討した．階層型行列は，密行列の部分行列を低ランク行列を用いて近似し，大きな密行列全体を多数の小密行列 と低ランク近似行列の集合として表すことで，計算量と必要メモリを削減することができる．階層型行列生成の際のクラスタリング，階層型行列ベクトル積について検討し，autorun カーネルと channel を用いることで FPGA の持つ性能を活かすことができる．

1. はじめに

科学技術計算において、高速な演算処理が求められる中で、様々なハードウェアが利用されている。今日では，従来のマルチコアプロセッサに加えて．数多くのコアを備えたメニーコアプロセッサや，画像処理用のハードウェアをベースに科学技術計算に転用した GPGPU などにより，多数の演算コアを用いることによりチップ内部の並列度を向上させていくことで性能を高めている．しかし，近い将来半導体プロセスの微細化が限界を迎える「ポストムーア」時代においては，チップ内では，コア単体性能の向上は見込めず，チップ面積の制約から，コア数を増加させることも困難になると考えられている。

そのような中で, 再構成可能なハードウェアとして FPGA (Field Programmable Gate Array) が注目されている．アプリケーションに特化し，カスタマイズされたハードウェアを用いることで汎用プロセッサに比べて効率よく高い性能を発揮しうると考えられる．

そのため様々な用途に対する FPGA の活用が模索されており．例えばデータセンタ内の処理に FPGA を活用する Catapult[1] などが知られている．また国内の HPC 研究分野における FPGA の活用についても．いくつかの例が存在している [5], [6]．しかし．これまで FPGA を用いて特定の処理を実現するためには．Verilog HDL などのハードウェア記述言語 (HDL) を用いて回路レベルで記述を行う必要があった．従って，FPGA 上で動作する一般的な科学技術計算プログラムを作成するには，アルゴリズムを論理回路レベルで詳細に設計する必要があり，大変困難であった．

その中で，回路設計技術に精通していなくても FPGA 上で動作するハードウェアを設計する方法として，OpenCL が利用されるようになってきた．OpenCL は，マルチコアプロセッサや GPU など，様々に異なるプラットフォーム間での並列処理を容易にプログラムするためのプログラミング言語である [2]．実際にいくつかの FPGA 製品においては．Verilog HDL などを用いることなく OpenCL のみを用いて汎用のプログラムを作成することが可能であり，HPC 分野における OpenCL プログラミングについても検討が進められている [7], [8], [9], [10], [11].

さらに、C++言語などのプログラミング言語での記述から直接回路を生成できるような高位合成 (High-Level Synthesis: HLS) コンパイラも FPGA ベンダから提供されるようになってきた [3], [4]。

これまでの HPC における FPGA の利用は，GPU などのアクセラレータと同様に，PCI Express を介してホストに接続し，CPU からの処理をオフロードする形態であった．しかし近年では, Intel 社により Xeon CPU と Arria 10 FPGA を QPI で接続した製品が発表されたり [12], [13]，今後 Xeon プロセッサと FPGA が UPI で接続され 1 チップ化された製品も登場する．一方, IBM Power9 では, NVLink2 インタフェースにより NVIDIA Volta GPU が直接接続可能になっている一方で，CAPI, OpenCAPI のインタフェースも持っており，同じく CAPI, OpenCAPI をサポートする Xilinx FPGA が接続できる．これらの環境では CPU と FPGA との間でキャッシュコヒーレントなメモリ転送が可

[1] 東京大学 情報基盤センター
[a)] hanawa@cc.u-tokyo.ac.jp

能になり，低オーバヘッドで FPGA へのオフローディングが実現できる．従って今後は CPU や GPU の不得手な処理を，比較的細粒度に FPGA 上で実行できるようになると考えられる．

本研究では，文献 [23], [24] における報告に引き続き，階層型行列計算の一部を，OpenCL を用いた FPGA 上での実現について検討する．

2. 階層型行列計算と $\mathcal{H}$ACApK ライブラリ

2.1 階層型行列

本論文では N 次元実正方行列 $\bar{A} \in \mathbb{R}^{N\times N}$ について考える．本論文では，$\bar{A}$ を部分行列に分割した上で，それら部分行列の大半を低ランク行列で近似したものを**階層型行列** A と呼ぶ．ここで，N 次元正方行列の行に関する添え字の集合を $I := 1, \cdots, N$ 列に関する添え字の集合を $J := 1, \cdots, N$ と表す．直積集合 $I \times J$ を重なりなく分割して得られる集合の中で，各要素 m が I と J の連続した部分集合の直積であるものを M とする．すなわち任意の $m \in M$ は $s_m \subseteq I, t_m \subseteq J$ を用いて $m = s_m \times t_m$ と表される．ある m に対応する $\bar{A}$ の部分行列を

$$A|^m_{s_m\times t_m} \in \mathbb{R}^{\#s_m\times\#t_m} \tag{1}$$

と書く．ここで $\#$ は集合の要素数を与える演算子である．階層型行列では、大半の m について $A|^m_{s_m\times t_m}$ の代わりに以下の低ランク表現 $\tilde{A}|^m$ を用いる．

$$\begin{aligned} &\tilde{A}|^m := V_m \cdot W_m \\ &V_m \in \mathbb{R}^{\#s_m\times r_m} \\ &W_m \in \mathbb{R}^{r_m\times\#t_m} \\ &r_m \leq min(\#s_m, \#t_m) \end{aligned} \tag{2}$$

ここで $r_m \in \mathbb{N}$ は行列 $\tilde{A}|^m$ のランクである。すなわち、低ランク行列 $\tilde{A}|^m$ とは、密行列 $A|^m_{s_m\times t_m}$ を V_m と W_m の積により近似した行列である。**図 1** に階層型行列の一例を示す。図 1 の濃く塗りつぶされた部分行列が $A|^m_{s_m\times t_m}$ に、薄く塗りつぶされた部分行列が $\tilde{A}|^m$ に対応する。

本論文では階層型行列に関する演算，特に行列・ベクトル積に関して論じる．ある階層型行列 A に関するデータ量 $N(M)$ は，m に対応する部分行列に関するデータ量 $N(m)$ を用いて以下のように表される。

$$N(M) = \sum_{m\in M} N(m) \tag{3}$$

$$N(m) := \begin{cases} \#s_m \times \#t_m & \text{m が密行列の場合} \\ r_m \times (\#s_m + \#t_m) & \text{m が低ランク行列の場合} \end{cases} \tag{4}$$

r_m が $\#s_m$ や $\#t_m$ と比べて十分に小さい場合，$r_m \times (\#s_m + \#t_m)$ は $\#s_m \times \#t_m$ と比べて小さい値となり，低ランク行列による表現がデータ量の点で有利となる．その結果，密行列を用いる場合と比べ，行列・ベクトル積等の行列演算に必要な演算量や行列の保持に必要なメモリ量を低減することができる．

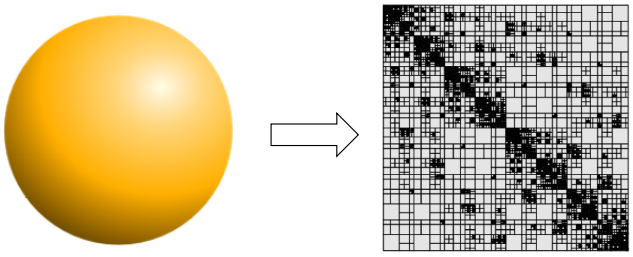

図 1 静電場解析対象と $\mathcal{H}$ACApK により生成される行列分割構造の例

2.2 対象とするコード

本稿では，階層型行列計算ライブラリとして，ppOpen-APPL/BEM ver.0.5.0 [19] に含まれる $\mathcal{H}$ACApK ライブラリ利用版のリファレンス実装を用いる．ppOpen-APPL/BEM とは，JST CREST「自動チューニング機構を有するアプリケーション開発・実行環境: ppOpen-HPC」[20] の構成要素の一つであり，境界要素法 (Boundary Element Method, BEM) 用のソフトウェアフレームワークである．

図 1 は，静電場解析において対象の構造物と $\mathcal{H}$ACApK により生成される行列分割構造の例を示したものである．図 1 中で黒く塗りつぶされた領域は小密行列で表現され，それ以外は低ランク近似行列で表現される．

2.3 階層型行列計算ライブラリ $\mathcal{H}$ACApK の実装概要

以下では，$\mathcal{H}$ACApK ライブラリの実装の概要について述べる．$\mathcal{H}$ACApK は主に，

(1) 階層型行列の生成
(2) 階層型行列を係数行列に持つ線形方程式のための線形ソルバ

の二つのパートから構成される．

2.4 階層型行列の生成

本稿では FPGA に実装する上で必要な部分のみの説明にとどめる．詳細は [31] を参照されたい．

$\mathcal{H}$ACApK の階層型行列の生成は，以下の 3 ステップにより行われる．

(1) 幾何情報に基づくクラスタリング（**図 2**：左）．
(2) 階層型行列構造の作成（図 2：中央）．この時点では部分行列のフレームを作成するだけで，行列要素は計算されない．
(3) 部分行列の計算（図 2：右）．低ランク近似可能な部

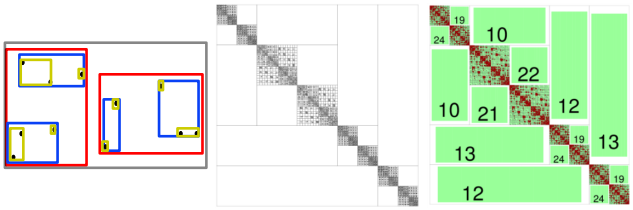

図 2 左：Cluster tree の作成，中央：階層型行列構造の作成，右：部分行列の計算（数字は部分行列の低ランク近似後のランク）

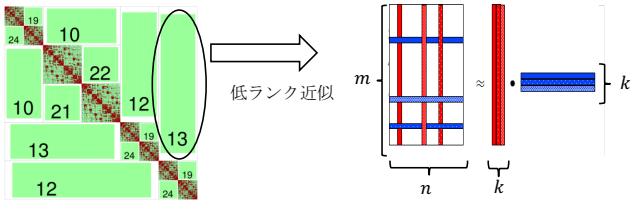

図 3 ACA による低ランク近似．$m \times n$ 行列を k 本の列ベクトルと k 本の行ベクトルの直積により近似する．k が大きくなるほど近似誤差が小さくなることが期待され，k の値により使用メモリ量と近似精度を制御する．

分行列については ACA(Adaptive Cross Approximation [32], **図 3**) を用いて近似部分行列を生成し，近似不可能と判定された部分については密行列として計算される．

このうち，ステップ (1), (2) における処理について具体的に述べる．ここで，低ランク行列による近似行列の組，および小さな密行列それぞれを，リーフと呼ぶ．実際に階層型行列において，近似行列または小さな密行列はツリーにおける葉（リーフ）に相当する，

(1) 近似行列あるいは小さな密行列がある大きさに達するまで分割し，ツリーを構成する．
(2) ツリー情報を元に行列構造を作成する．
(3) 近似行列および小さな密行列をアクセス順に並び替える．

2.5 階層型行列ベクトル積

$$Ax \to y, \quad x, y \in \mathbb{R}^N \tag{5}$$

本論文ではこの演算の実施手順として，各部分行列毎に行列積を実行し，その結果を統合することで最終的な結果 y を得るという，最も自然でかつ効率的と考えられる方法を採用する．密行列により表現されている部分行列 $A|^m_{s_m \times t_m}$ については

$$A|^m_{s_m \times t_m} \cdot x|_{t_m} \to \hat{y}|_{s_m} \tag{6}$$

を計算する。ここで、$x|_{t_m}$ は x の各要素のうち t_m に対応する要素のみを抜き出して生成した $\#t_m$ 個の要素からなるベクトルである。$\hat{y}|_{s_m}$ は $\#s_m$ 個の要素を持つベクトルであり、各要素は y の s_m に対応する要素の部分積の一つとなる。

次に、低ランク表現が用いられている行列 $\tilde{A}|^m$ に関しては、$c \in \mathbb{R}^{r_m}$ として、まず

$$Wm \cdot x|_{t_m} \to c|_{r_m} \tag{7}$$

を計算し、さらに

$$Vm \cdot c|_{r_m} \to \hat{y}|_{s_m} \tag{8}$$

を計算することで、$\tilde{A}|^m \cdot x|_{t_m} = Vm \cdot Wm \cdot x|_{t_m} \to \hat{y}|_{s_m}$ を得る。

それぞれの部分行列について $\hat{y}|_{s_m}$ を計算した後、

$$\sum_{m \in M} \hat{y}|_{s_m} \to y \tag{9}$$

のようにこれらを統合し、最終的な結果とする。

表 1 対象とする階層型行列の構成

行列名	100ts	216h	human_1x1
行数	101250	21600	19664
総リーフ数	222274	50098	46618
近似行列個数	89534	17002	16202
小密行列数	132740	33096	20416

2.6 対象とする行列

本稿では**表 1** に示す 3 つの階層型行列を扱う．これらの行列はいずれも境界要素法を用いた静電場解析において現れる行列である．

3. OpenCL による FPGA プログラミングと性能最適化

3.1 OpenCL を用いた FPGA プログラミング

FPGA 内部の論理を設計するためには，従来は Verilog HDL や VHDL といったハードウェア記述言語を用いて記述するのが一般的であり，求められるアルゴリズムにあわせて人手で論理回路レベルに変換する必要があった．そのため，例えば C 言語や Fortran を用いれば数行で実装できるような単純な処理を行うだけでも，FPGA 上に実装するためには多大な時間と労力が必要であり，様々な HPC アプリケーションに FPGA を活用することは現実的ではなかった．

しかし近年では，OpenCL, C++などを用いた設計ツールが FPGA ベンダーによって提供されるようになってきた．

OpenCL は Khronos グループによって標準化されている並列化プログラミング環境であり，GPU などのアクセラレータ向けに仕様策定や開発が進められたものである．FPGA 向けのコンパイラでは，OpenCL での記述から内部で Verilog HDL を出力し，論理合成を行う．

これまで Intel 社では，ホスト CPU の演算の一部をオフロードするためのアクセラレータとして FPGA を利用

表 2 対象とする FPGA 製品の仕様

FPGA	Intel Arria 10 GX E1 (10AX115N2F45E1SG)
#Logic units (ALMs)	427,200
#RAM blocks (M20K)	2,713
#DSP blocks	1,518 (浮動小数点)
ボード	Terasic DE5a-Net E1
DDR メモリ容量	(4 + 4) GB
DDR メモリバンド幅	17.1 GB/sec
PCIe I/F (OpenCL の場合)	Gen3 x8
ツール	QuartusPrime Pro 16.1.1

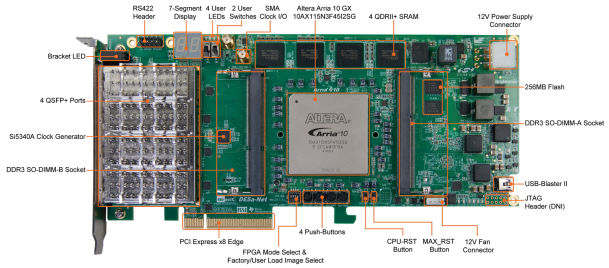

図 4 Terasic 社 DE5A-Net Arria 10 FPGA Development Kit (Teraric 社提供)

できるようなツールを提供している．この場合，FPGA は PCI Express 拡張ボードの形でホストに装着されるのが一般的である．

本研究では，FPGA として Intel 社の FPGA である Arria 10 GX 1150 が搭載された，Terasic 社の PCI Express ボード DE5a-Net Arria 10 FPGA Development Kit (**図 4**) を用いる．

本 FPGA は，**表 2** に示すように，Adaptive Logic Module (ALM) と呼ばれる論理モジュール 427,200 個で構成されており，各モジュールは 4 個のレジスタ，2 個の 8 入力 2 出力 Look Up Table (LUT) および 2 個の全加算器とマルチプレクサから構成されている．さらに，FPGA チップ内部には 2,713 個の 20Kbit からなる RAM ブロック (M20K) が含まれ，それとは別に，640bit からなる Memory Logic Array Block (MLAB) 20,774 個も使用することができる．

また，これらとは別に，固定小数点および浮動小数点算術演算をサポートする，可変精度 DSP (Digital Signal Processor) を備える．DSP 単体で，単精度浮動小数点の乗算，加減算，積和演算をサポートしている．チップ全体としては，1,518 個の DSP を備え，理論ピーク性能としては，単精度で 1.37 TFLOPS の演算能力を持つ．また，倍精度演算を行うためには，DSP を 4 つと付加回路を用いることで実現できる．Arria 10 以降では，浮動小数点演算をサポートする DSP が搭載されたことで，より多くの浮動小数点演算を実現することができる．

OpenCL コードのコンパイルを行うには，ユーザは Intel FPGA SDK for OpenCL Offline Compiler (実際には aoc コマンド) を実行するだけである．それによって，内部では以下のような処理が行われる．

- “aoc -c kernel.cl” で行われる処理
  (1) OpenCL のプログラム構造から，パイプラインステージを切り出し，ステートマシンを構成する．
  (2) 必要な資源を見積もる．(ロジック使用率，レジスタ使用率，メモリ使用率，DSP 使用率)
  (3) PCI Express や DDR3-DRAM インタフェースなどのモジュール，OpenCL を元に Verilog HDL の記述が生成される．
  (4) kernel.aoco ファイルの出力
- “aoc kernel.aoco” で行われる処理
  (1) Quartus (Intel 社の FPGA 向け論理合成ツール) を起動し，論理合成，配置配線等を行う．
  (2) FPGA コンフィグレーションデータのビットストリームを含む kernel.aocx ファイルが出力される．

ツールの使い方としては，非常に容易である反面，現状では依然として以下のような課題がある．

- コンパイルに非常に時間がかかる
  aoco ファイルから aocx ファイルに変換する部分では，コンパイラ内部で Quartus ツールを起動して，論理合成および，FPGA デバイスへの配置配線などが行われる．現状ではどんなに簡単な記述でも，1 回のコンパイルで Intel Xeon E5 (Haswell プロセッサ) を用いても 2 時間近くかかる．本来であれば，インタフェース部分などの回路ブロックは不変なはずであり，OpenCL のコードに対応する部分のみを部分再構成で済むはずである．
  今後の FPGA デバイスやツール群の改良により，必要最小限部分の合成やマッピングなどでコンパイル時間が短縮されることが望まれる．
- 設計時にハードウェア資源，性能の予測が難しい
  FPGA 内部に含まれるハードウェア資源をどの程度使用するかをレポートする機能が提供されており，コンパイラに`--report -c` オプションを与えることで利用することができる．また，パイプラインステージの構成についても，クリティカルパスや，依存関係などのメッセージが表示され，性能を改善するためのヒント情報なども含まれている．しかし，FPGA に収まるかどうかの目安にしかならず，最終的には上記の通り，長時間の論理合成の結果を待つ必要がある．動作周波数や，最終的なステートマシンの情報については，事前に予測することはできないため，結果を見ながらトライ&エラーでソースコードの改良を進める必要がある．
  最新の Quartus 17.1 では，インクリメンタルコンパイルや高速コンパイルなどの機能が実装されているが，

そのバージョンに対応したライブラリ[*1] が必要であり，ボードベンダがそれを提供してくれない限り，移行は困難である．

3.2 FPGA に向けた OpenCL 記述

OpenCL は C++言語を元にした並列化プログラミング環境であり，接頭辞を用いて関数や変数に対してその実行場所や配置場所といった追加情報を与えるという言語拡張が行われている．またデバイス間でのデータ通信などの機能 (API 関数) も提供されている．言語仕様の策定において GPU での利用が強く意識されていたこともあり，OpenCL のプログラム記述方法や実行モデルは CUDA[26] と類似点が多い．OpenCL を用いた並列化プログラミングは，CPU やメニーコアプロセッサにて広く用いられている OpenMP や GPU 向けの主要な並列化プログラミング環境である CUDA と比べるとプログラム記述量などの点で優れているとは言い難い．しかし OpenCL のみを用いて FPGA プログラミングが行えることは科学技術計算に FPGA を使用したい利用者にとっては大きなメリットである．

現在の OpenCL はバージョン 2.0 が最新版であるが，現在の Offline Compiler 16.1 では バージョン 2.0 の一部までをサポートしている．

通常 OpenCL コードは実行時にコンパイルされるが，FPGA では論理合成に長時間を要するので，事前に offline compiler によりバイナリ（実際にはコンフィグレーションデータ）を生成する必要がある．

FPGA 向けの高性能な OpenCL プログラムを作成するためには様々な最適化を行う必要がある．特に FPGA を使う場合には，ハードウェアの構成自体を利用者がある程度自由に指定できる点が特徴的である．また GPU 向けの OpenCL 最適化プログラミングにおいては，GPU 上の大量の演算器を十分に活用できるように非常に高い並列度を持つプログラムを記述することが非常に重要である一方，FPGA はハードウェア資源の制約から GPU のような高い並列度には向いていない．そのため同じ OpenCL を用いるものの，FPGA 向けのプログラムには GPU とは異なる最適化プログラミング戦略が必要である．

3.3 最適化

Intel 社の FPGA に向けた最適化プログラミング手法については Intel 社によるプログラミングガイド [28] や最適化ガイド [29] などの公開情報に詳しく紹介されている．本稿では特に

- メモリの使い分け
- コード記述レベルの最適化
- ループアンローリング
- SIMD 化

に着目し，次章では実際にプログラムを作成してその効果を確認する．

3.3.1 メモリの使い分け

OpenCL においては数種のメモリを使い分けることができる．__global 接頭辞を付けた配列は，FPGA ボード上の DDR メモリ上に確保され，ホストとのやりとりなどに使用することができる．

一方，__local 接頭辞を付けた配列は，FPGA 内部の RAM ブロックなどで構成されるオンチップメモリ上に確保される．しかしながら，OpenCL カーネル内部でしか利用することができない．したがって，一旦__global 配列から__local 配列にデータをコピーした後に，__local 配列を計算に使用し，その結果を__global 配列に書き戻す必要がある．

3.4 コード記述レベルの最適化

前節までに述べた最適化手法はプログラムの構造自体を変化させない最適化であった．本節ではプログラムの構造を変化させるようなコード記述レベルの最適化について述べる．

OpenCL コンパイラでは主に for 文や while 文などのループ構造を解析し，ハードウェアレベルのパイプラインに変換する．

FPGA においては，ハードウェアの使用量を抑える工夫も必要である．通常の CPU プログラムであれば，キャッシュの効率なども考慮して，例えば初回の反復で実行する処理と，その後の残りの反復処理を分離して記述するような場合がある．しかし，ハードウェア資源の制約と，その処理に特化したパイプラインが生成されることを考えると，なるべく共通化できる部分は共通化しておく方がよい場合がある．内部に分岐を含む処理であっても，ハードウェアでは，単にセレクタによって信号線が選択されるだけであり，性能にはほとんど影響がない．また，全体を通してパイプラインの 1 ステージの処理時間が他のステージによって決まるような場合であれば，冗長な計算をしても性能にあまり影響はないため，例えば 0 と掛け算を意図的に行うことで不要な項を削除するなどして，回路を共通化することも可能である．

これらのことから，逐次実行 (single stream) において高い性能を実現するためには，

- 各 for 文の中に含まれる処理量が，おおよそ均等，または整数倍となり，バランスが取れること
- 共通化できそうな文はまとめること
- メモリアクセスは最小化すること

などが挙げられる．

3.4.1 ループアンローリング

一般的な CPU 向けのループアンローリングは，ループ

[*1] 正確には Board Support Package (BSP)

制御のための命令数を削減するとともに分岐無しで連続実行できる命令数を増加させたり，メモリに対してバースト転送を可能にする効果がある．FPGA においても，メモリアクセスを含むループをアンロールすると，coalesced アクセスが期待できる場合には高い性能を得られる可能性がある．

3.4.2 SIMD 化

FPGA は搭載されている資源の制約上，GPU のような非常に高い並列度を持つプログラムがそのまま実行できるわけではない．しかし，同様の並列化自体は可能であり，それによって OpenCL カーネル起動オーバヘッドを低減することができ，資源量にあわせた適切な並列化を行うことで性能向上が期待できる．

OpenCL では clEnqueueNDRangeKernel 関数を用いて FPGA カーネル関数を実行するが，この関数の引数には実行時の並列度を与えることが可能である．FPGA カーネル関数側では実行時に自身の ID を得る API 関数が用意されているため，この ID を用いて自身の計算するべき範囲を決めるなどの方法により並列処理が実現可能である．この実装方法は CUDA を用いた GPU プログラミングなどと類似しており，高い性能が期待される並列度には差があるものの，GPU 向けに実装されたプログラムを FPGA 向けに移植する際には低い移植コストにて利用可能な最適化手法であると考えられる．

さらに OpenCL を用いた FPGA プログラミングにおいては，カーネル関数に対して付加できる attribute 情報を用いて並列実行時の動作を制御することができる．たとえば `num_simd_work_items(4)` の指定をすることで SIMD 長が 4 の計算ユニットが作成される．また，`num_compute_units(4)` を指定すれば 4 つの計算ユニットが作成されるが，この場合，メモリアクセスの必要があるとインタフェースが複数必要になってしまう．

Offline Compiler は，内部でスレッド ID などを獲得する `get_global_id(0)` などの関数が用いられている場合には NDRange として複数スレッドを想定したコンパイルを実行し，そうでない場合には single stream を想定したコンパイルを行う．

3.5 autorun カーネルと channel の使用

カーネルに対して `__attribute__((autorun))` を指定することで，ホストからの制御を必要としない，autorun カーネルを生成，利用することができる．ホストからの起動が必要なく，起動オーバヘッドがほとんどない．その代わり，引数を取ることができないため，後述の channel を使って，他のカーネルから値を受け取って動作する形にする必要がある．

channel は Altera による拡張で，OpenCL カーネル間をホストを介さずに値の受け渡しをする．channel として指定したデータ型に合わせて，書き込んだり読み出したりすることで，1 クロックでそのデータを送受信することができる．構造体とすることで，複数データや付属情報も一括して送受信することができる．

Intel が Arria10 向けに OpenCL で実装した SGEMM[30] では，これらの autorun カーネルと channel とを組み合わせて 約 1 TFLOPS と高い性能を得ている．藤田らの実装 [11] でも同様に autorun カーネルと channel により高い性能が得られている．

まず入力をメモリから読み出さない限り，channel の先のカーネルも動作することはない．入力となる配列のアドレス等はホストから指示する必要があるため，ホスト側から，配列データをメモリから読み出し，channel に書き込むようなカーネルを記述することになる．channel は，それぞれが独立に動作するため，1 つのカーネルから複数 channel に書き込み（アンロールすれば同時に実現できる），多数の autorun カーネルに値を渡して並列動作させることも可能である．

4. 実装と評価

4.1 対象問題

我々は，これまで [23], [24] において，階層型行列ベクトル積について Stratix V GX, Arria10 GX FPGA への実装を試みた．

本論文では，Arria10 GX FPGA を用い，行列ベクトル積の性能改善と階層型行列生成におけるクラスタリングへの適用について検討する．

実験環境として，Intel Xeon E5-2650v3 (Haswell) 2 ソケット搭載のサーバを用い，その PCI Express スロットに 3.1 節で述べた Terasic 社の Arria 10 搭載 FPGA ボード DE5a-Net Arria 10 FPGA Development Kit を接続した．

4.2 階層型行列ベクトル積

$\mathcal{H}$ACApK の行列ベクトル積サブルーチン `HACApK_adot_body_lfmtx` について，改めて実装を行った．**図 5** にベースとなる実装を示す．

今回は，(1) 単にループアンローリングを用いて，並列処理を行うものと，(2) autorun カーネルと channel を用いてパイプライン+並列処理を行うもの，の 2 つのバージョンについて検討を行った．

(1) について，最大限アンロールするためには，演算器リソースを節約する必要がある．図 5 の 7〜13 行目，14〜19 行目，21〜26 行目のように，似たような計算パターンが多く，特に乗算器を複数パターンで使い回すことが有効と考えたため，これらを共通に扱える関数を定義した．また，最内ループにおいて，アンローリングを行った．

しかしながら，予備コンパイルの時点では，64 段アンローリングでもハードウェアに収まるはずであったが，論

```
1    for(ip=0; ip<nlf; ip++){
2      sttmp=st_lf+ip;
3      ndl=sttmp->ndl; ndt=sttmp->ndt;
4      nstrtl=sttmp->nstrtl; nstrtt=sttmp->nstrtt;
5      if(sttmp->ltmtx==1){
6        kt=sttmp->kt;
7        for(il=0; il<kt; il++){
8          zbu[il] = 0.0;
9          for(it=0; it<ndt; it++){
10           itt=it+nstrtt-1;
11           itl=it+il*ndt + sttmp->offset_a1;
12           zbu[il] += a1[itl]*zu[itt];
13       } }
14       for(il=0; il<kt; il++){
15         for(it=0; it<ndl; it++){
16           ill=it+nstrtl-1;
17           itl=it+il*ndl + sttmp->offset_a2;
18           zau[ill] += a2[itl]*zbu[il];
19       } }
20     } else if(sttmp->ltmtx==2){
21       for(il=0; il<ndl; il++){
22         ill=il+nstrtl-1;
23         for(it=0; it<ndt; it++){
24           itt=it+nstrtt-1;
25           itl=it+il*ndt + sttmp->offset_a1;
26           zau[ill] += a1[itl]*zu[itt];
27   } } } }
```

図 5 ベースとなる実装コード

理合成に失敗しており，原因を調査中である．

(2) について，(1) のループアンローリングの代わりに，多数の autorun カーネルによっても並列処理を行うことができる．3.5 節でも述べたように，それぞれの配列を読み出す専用のカーネルを用意し，ホストから順次起動する．これにより，各カーネル内では連続したメモリアクセスになり，float4 などのベクトルとして扱うことで coalesced アクセスになることが期待できる．各 kernel では，channel に書き込むことでその先の channel+autorun カーネルの方が理想的に並列性を引き出せると考えられる．原稿執筆時点では実装を行っている段階である．

4.3 階層型行列生成におけるクラスタリング

2.4 節で述べたように，$\mathcal{H}$ACApK の行列生成においてはクラスタリング処理に際してツリーを構成したり，ソートを行っている．

これらの関数について，オリジナルのコードでは再帰を用いて記述されている．しかしながら，OpenCL では再帰がサポートされていない．スタックを自分で定義して再帰を行わないように書き換えることも可能であるが，channel 文と，同一のカーネルを複数使うことで，再帰と同様の処理を記述することが可能である．

この際には，`__attribute__((autorun))` に加えて，`__attribute__((num_compute_units(N)))`(N は任意) をカーネルに指定し，カーネル内では `get_compute_id(0)` によって ID を取得して動作を切り替えれば良い．

オリジナルの Fortran コードから，C 言語に変換した版[*2]を用いて，現在実装を進めている．

[*2] 京都大・平石助教らによる．

5. おわりに

本稿では，階層型行列計算の FPGA への OpenCL による実装について検討を行った．

現在実装を進めているが，今回用いた Terasic 製のボードにおいて，実験中不安定な部分があったり，オフラインコンパイラでは Verilog HDL に変換できても，論理合成でエラーを出して終了するケースが多々見られた．そのため最新の Quartus 17.1 で動作する Board Support Package が必要であり，既存の 16.1.1 の BSP を移植する必要があると考えられる．

さらに今後は，QPI や OpenCAPI のように，CPU と FPGA とがより緊密に接続された環境も合わせて FPGA の有効活用を検討していく予定である．

謝辞 本研究の一部は，JSPS 科研費 15K00166 の助成を受けたものです．本研究で用いた Quartus II のライセンスの一部は，Intel/Altera 社 University Program によります．

参考文献

[1] Putnam, A. and Caulfield, A.M. and Chung, E.S. and Chiou, D. and Constantinides, K. and Demme, J. and Esmaeilzadeh, H. and Fowers, J. and Gopal, G.P. and Gray, J. and Haselman, M. and Hauck, S. and Heil, S. and Hormati, A. and Kim, J.-Y. and Lanka, S. and Larus, J. and Peterson, E. and Pope, S. and Smith, A. and Thong, J. and Xiao, P.Y. and Burger, D., A reconfigurable fabric for accelerating large-scale datacenter services, 2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA), pp.13-24, 2014.

[2] OpenCL - The open standard for parallel programming of heterogeneous systems `https://www.khronos.org/opencl/`

[3] Intel HLS Compiler: Fast Design, Coding, and Hardware, `https://www.altera.co.jp/content/dam/altera-www/global/en_US/pdfs/literature/wp/wp-01274-intel-hls-compiler-fast-design-coding-and-hardware.pdf`

[4] Vivado Design Suite, `https://japan.xilinx.com/support/documentation/sw_manuals_j/j_ug1197-vivado-high-level-productivity.pdf`

[5] 佐野 健太郎, 河野 郁也, 中里 直人, Alexander Vazhenin, Stanislav Sedukhin: FPGA による津波シミュレーションの専用ストリーム計算ハードウェアと性能評価, 情報処理学会 研究報告 (2015-HPC-149), 2015.

[6] 上野 知洋, 佐野 健太郎, 山本 悟: メモリ帯域圧縮ハードウェアを用いた数値計算の高性能化, 情報処理学会 研究報告 (2015-HPC-151), 2015.

[7] 丸山 直也, Hamid Reza Zohouri, 松田 元彦, 松岡 聡: OpenCL による FPGA の予備評価, 情報処理学会 研究報告 (2015-HPC-150), 2015.

[8] 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.

[9] ウィッデヤスーリヤ ハシタ ムトゥマラ 他：OpenCL を用いたステンシル計算向け FPGA プラットフォーム，情報処理学会 研究報告 (2016-HPC-154)，2016.

[10] Hamid Reza Zohouri, Naoya Maruyama, Aaron Smith, Motohiko Matsuda, and Satoshi Matsuoka, “ 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), 2016.
[11] 藤田典久，他，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 (2016-HPC-161), No. 12, 2016.
[12] PC Watch, Intel、Altera の FPGA「Arria 10 GX」を 1 パッケージに統合した Xeon, http://pc.watch.impress.co.jp/docs/news/752237.html, 2016 年
[13] Gupta, P. K. “Accelerating datacenter workloads,” 26th International Conference on Field Programmable Logic and Applications (FPL), 2016.
[14] S. K. Sadasivam, B. W. Thompto, Ron Kalla, and William J. Starke, “IBM Power9 Processor Architecture,” IEEE Micro, Vol. 37, No. 2 , pp. 40–51, 2017.
[15] OpenCAPI Consortium, http://opencapi.org
[16] CCIX Consortium, Connecting Processor Architecture and Accelerators Seamlessly, https://www.ccixconsortium.com
[17] Gen-Z Consortium, http://genzconsortium.org
[18] K. Nakajima and M. Satoh and T. Furumura and H. Okuda and T. Iwashita and H. Sakaguchi and T. Katagiri and M. Matsumoto and S. Ohshima and H. Jitsumoto and T. Arakawa and F. Mori and T. Kitayama and A. Ida and M. Y. Matsuo and K. Fujisawa and et al., ppOpen-HPC: Open Source Infrastructure for Development and Execution of Large-Scale Scientific Applications on Post-Peta-Scale Supercomputers with Automatic Tuning (AT), Optimization in the Real World, pp.15–35, DOI 10.1007/978-4-431-55420-2_2, 2016.
[19] T. Iwashita, A. Ida, T. Mifune, and Y. Takahashi, Software Framework for Parallel BEM Analyses with H- matrices Using MPI and OpenMP, Procedia Computer Science, Vol. 108, pp. 2200 2209 (2017).
[20] ppOpen-HPC — Open Source Infrastructure for Development and Execution of Large-Scale Scientific Applications on Post-Peta-Scale Supercomputers with Automatic Tuning (AT) http://ppopenhpc.cc.u-tokyo.ac.jp/ppopenhpc/
[21] 塙 敏博, 児玉 祐悦, 朴 泰祐, 佐藤 三久, Tightly Coupled Accelerators アーキテクチャに基づく GPU クラスタの構築と性能予備評価, 情報処理学会論文誌（コンピューティングシステム）, Vol.6, No.4, pp.14-25, 2013.
[22] Yuetsu Kodama, Toshihiro Hanawa, Taisuke Boku and Mitsuhisa Sato, “PEACH2: FPGA based PCIe network device for Tightly Coupled Accelerators,” International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2014), pp. 3-8, Jun. 2014.
[23] 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.
[24] 塙 敏博, 伊田 明弘, 星野 哲也, 階層型行列計算の FPGA への適用，情報処理学会研究報告 (2017-HPC-161), 2017.
[25] Altera Corporation, Floating-Point IP Cores User Guide, UG-01058, 2015.
[26] CUDA Dynamic Parallelism, http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#cuda-dynamic-parallelism
[27] Intel Corporation, Intel FPGA SDK for OpenCL - 概要 https://www.altera.co.jp/products/design-software/embedded-software-developers/opencl/overview.html
[28] Intel Corporation, Intel FPGA SDK for OpenCL Programming Guide 16.1, UG-OCL002, 2017.
[29] Intel Corporation, Intel FPGA SDK for OpenCL Best Practice Guide 16.1, UG-OCL003, 2017.
[30] Amulya Vishwanath, Enabling High-Performance Floating-Point Designs, https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/wp/wp-01267-fpgas-enable-high-performance-floating-point.pdf
[31] A. Ida, T. Iwashita, T. Mifune and Y. Takahashi, “ Parallel Hierarchical Matrices with Adaptive Cross Approx w ima-tion on Symmetric Multiprocessing Clusters, ” Journal of Information Processing Vol. 22, pp.642-650, 2014.
[32] Kurtz S., Rain O. and Rjasanow S.: The Adaptive Cross-Approximation Technique for the 3-D Boundary-Element Method, IEEE Trans. Magn., Vol.38(2), pp.421-424 (2002).