Vol.2019-HPC-168 No.11
2019/3/6

FPGAへのオフロード最適化のための SPGenとOpenCLの統合の検討

渡部 裕[1,a)]　李 珍泌[2]　佐野 健太郎[2]　朴 泰祐[3,1]　佐藤 三久[2,1]

概要：高性能計算向け FPGA (Field-Programmable Gate Arrays) 利用が注目を集めている．FPGA とは書き換え可能なハードウェアであり，計算に特化した効率的な回路を生成可能である．高性能計算で FPGA を使用するための問題の 1 つはどのようにして必要な回路を記述するかである。従来の HDL (Hardware Description Language) を用いた複雑な回路設計を緩和するために導入された高位合成も依然として複雑であり，さらなる改善が必要とされる．本研究では OpenMP プログラミングモデルを用いることを想定し、OpenMP により指定された部分を OpenCL を用いて FPGA にオフロードする。本稿では、FPGA の回路を記述するための言語としてストリーム計算用フレームワークである SPGen に注目し，FPGA へのオフロードされた OpenCL の一部を SPGen を用いて最適化することを提案する。ラプラス方程式を用いた評価では，OpenCL のみを使用し，ディレクティブによる最適化を行う場合に対し本手法は高い演算性能を示した．また，今後 OpenCL と SPGen に変換を行う OpenMP コンパイラを実装することで OpenCL の最適化の可能性を示した．

1. 序論

高性能計算の分野において，FPGA (Field-Programmable Gate Arrays) が注目を集めている．FPGA とは再構成可能なハードウェアであり，計算に特化した効率的な回路を生成することが可能である．近年の半導体技術の停滞により汎用プロセッサの性能向上率が低下しており，今後のポストムーア時代においては計算に特化した効率的なアーキテクチャが主流の一つとなることが考えられる．また FPGA は消費電力性能に優れていることから，近年の大規模計算クラスタにおいて問題となっている膨大な消費電力の解決策の一つとしても期待されている．なお，筑波大学　計算科学研究センターでは FPGA と GPU を併用することでそれぞれの弱点を補うことなどを目的とした研究が行われており，Intel SkyLake-SP CPU，NVIDIA Volta GPU，Intel Stratix 10 FPGA から構成される次世代スーパーコンピュータ “Cygnus” が 2019 年 4 月より運用される予定である．

FPGA を高性能計算で利用するための大きな問題がプログラミングモデルと FPGA のプログラミングである．従来の HDL (Hardware Description Language) を用いた回路設計は複雑な作業となるため，OpenCL (Open Computing Language)[1] などの高級言語を用いる高位合成が導入された．しかし、ベンダー独自の拡張や FPGA に特化した最適化手法，さらにはコンパイラの最適過不足などのため，依然として FPGA のプログラミングは複雑なままである．また，高位言語から HDL への変換アルゴリズムは非公開となっており，OpenCL を用いる先行研究では経験に基づく最適化が多く行われている状況となっている．これらの複雑さを解決するため，先行研究では OpenCL をバックエンドとする OpenACC[2] プログラミングを用いた Intel FPGA 向け開発環境に関する研究が行われているが，FPGA 向けの最適化手法に対応するための OpenACC 独自拡張や独自ディレクティブの追加が行われているため使いこなすためには十分な経験が必要となる．そのため OpenCL プログラミングにおいても、その複雑さが十分に解決されているとはいえない

我々は、FPGA プログラミングのための OpenMP[3] コンパイラを検討している．基本的には OpenMP の既存ディレクティブのみで FPGA に対応することを目標とする．OpenMP により指定された部分を OpenCL を用いて FPGA にオフロードする．本稿では、FPGA の回路を記述するための言語としてストリーム計算用フレームワークである SPGen に注目し，FPGA へのオフロードされた OpenCL の一部を SPGen を用いて最適化することを提案する．現在の SPGen は FPGA におけるメモリアクセスに柔軟性がな

[1] 筑波大学 システム情報工学研究科
[2] 理化学研究所 計算科学研究センター
[3] 筑波大学 計算科学研究センター
[a)] ywatanabe@hpcs.cs.tsukuba.ac.jp

く，SPGen で使用されるデータ形式に従いホスト側でデータ構造の変換を行わなければならない，といった問題がある．そこで OpenCL RTL module として SPGen から生成された IP コアを組み込むことで FPGA におけるメモリアクセスの柔軟性の向上を図る．従って，OpenMP をフロントエンドとし，ホスト側のランタイム呼び出しおよび FPGA での処理を記述する OpenCL + SPGen に変換を行うコンパイラの設計を行うこととする．

本稿ではその前段階として OpenCL に SPGen を組み込むことによる最適化について検討・評価を行い，その後 OpenMP での記述方法に関する考察を行う．

本稿は次の章で構成される．はじめに第 3 章については FPGA のプログラミングについて述べ，第 3 章では提案手法である OpenCL に SPGen を組み込む手法について述べる．第 4 章では関連研究を紹介する．第 5 章では提案手法の評価を行い，また第 6 章では Intel 社の最新の HPC 向け FPGA である Stratix10 への移植に関する評価を行う．最後に第 7 章で結論を述べる．

2. FPGA のプログラミング

2.1 Intel FPGA SDK for OpenCL

Intel FPGA SDK for OpenCL[4], [5], [6] とは，Intel 社が提供する同社 FPGA 向けの OpenCL を用いた高位合成開発環境である．ユーザは OpenCL フレームワークを使用し FPGA をプログラム及びホストからの操作を行うことが可能であり，従来の一般的なプログラミングモデルである HDL を用いた開発の複雑さを緩和している．ただし Intel 社により FPGA 向け OpenCL 拡張が加えられており，FPGA プログラムの最適化にこれらは必須である．また CPU や GPU 向けの最適化手法と FPGA 向けの最適化手法は異なり，FPGA を意識したプログラミングが必須となる．OpenCL のみを利用した FPGA プログラミングでは，回路構造を明示的に決定することは困難であり基本的にコンパイラ依存となる．channel を使用することでパイプラインを明示的に生成することは可能であるが，Stratix10 においては channel の高いオーバーヘッドのため非推奨となっている [7].

2.2 SPGen

SPGen とは, FPGA においてストリーム計算を行う回路を生成するためのフレームワークである. SPGen では浮動小数点演算のみをサポートする. spd とよばれる言語を用いてアプリケーションの記述を行い, コンパイラによって自動的にパイプライン化された HDL モジュールが生成される. また, 必要に応じて HDL モジュールを呼び出すことが可能である. このモジュールを複製およびカスケードに接続することで演算の並列化を行うことが可能である. 生成されたモジュールは, shell とよばれる, DDR などのペリフェラルなどを制御する回路に組み込まれたのち bitstream の生成が行われる. SPGen を用いて実装された流体シミュレーション [8] では, Intel 社の 1 世代前の HPC 向け FPGA である Arria 10[9] を使用し 519 GFLOPS の性能を達成している. このときの消費電力性能は 9.67 GFLOPS/W であり, SPGen を用いて開発することで高い消費電力性能を得られることが示されている. また最新の Intel 社の HPC 向け FPGA である “Stratix 10 FPGA”[10] において 6844 GFLOPS 達成できることが見込まれている. Stratix10 の消費電力から計算すれば, およそ 24 - 49 GFLOPS/W の消費電力性能が達成可能と考えられる.

プログラム 1: spd で記述した SAXPY

```
Name saxpy;
Main_In {Mi::in0, in1, sop, eop};
Main_Out {Mo::out0, sop, eop};
EQU equ0, tmp = 3.1337*in0;
EQU equ1, out0 = tmp*in1;
DRCT (Mo::sop, Mo::eop) = (Mi::sop, Mi::eop);
```

プログラム 2: C で記述した SAXPY

```
float in0[SIZE];
float in1[SIZE];
float out0[SIZE];

void saxpy()
{
    for (unsigned int i=0; i<SIZE; i++) {
        out0[i] = in0[i] + 3.1337f*in1[i];
    }
}
```

spd プログラムの例をプログラム 1 に示す. プログラム 2 はこれと等価な C 言語のプログラムである. このように, spd では基本的に EQU 構文を用いて式を記述することで計算ロジックの実装を行う. なお, Static Single Assignment 形式で記述する必要がある. この spd プログラムはコンパイラにより依存性の解析が行われ, 図 1 のようなデータフローグラフが生成される. なお, 各パスのパイプライン長を調整するため, 青の四角で示されているような遅延が挿入される. 実際の回路はこのような構造となる.

このように HDL などと比較し抽象化されており，式を用いることで計算ロジックの記述を行う非常にシンプルな構造となっている．ただし現状の SPGen では FPGA 上での for loop の記述やランダムアクセスなどが不可能であり，SPGen を用いて記述可能なアプリケーションは限定される．なお，ユーザ独自の HDL モジュールによる拡張がサポートされている．

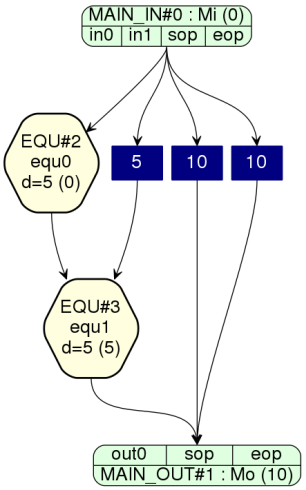

図 1: 生成されたデータフロー図

ただし SPGen は FPGA 上におけるメモリアクセスに自由がなく，ホスト側でデータの整形や順序の入れ替えといった操作を行ったうえでデータ転送を行わなければならない．また SPGen shell がサポートする FPGA ボードは限られているといった問題もある．

そこで，我々は SPGen によって生成された，パイプライン化された HDL モジュールを OpenCL に RTL Module として組み込むことで FPGA にオフロードする計算の最適化を行うことを目標とする．OpenCL のみで記述する場合における最適化の複雑さ等に対し，SPGen で計算を記述することでパイプライン化されることを保証することが可能である．また，OpenCL の柔軟なメモリアクセスの機構を活用可能であるため前述した FPGA 上でのメモリアクセスの制約を緩和することが可能である．さらには OpenCL を用いた開発に対応する Intel FPGA は多く存在するためそれらに容易に移植可能である．

2.3 RTL Module の組み込み

Intel FPGA SDK for OpenCL では，OpenCL での記述に加え，HDL で記述された IP コアを組み込むことが可能である．ユーザが記述する OpenCL プログラムと IP コアは Avalon ST インターフェースを用いて接続される．また，Avalon MM インターフェースを用いた接続も提供されており，IP コア内部から Avalon MM を介したメモリアクセスが可能である．Avalon ST とはデータをストリームの形でやり取りするための．Avalon MM とは，メモリ空間にマップされた領域に対しアドレスを用いて読み書きを行うためのインターフェースである．

Avalon ST コンポーネントに準拠した RTL Module を組み込む場合の記述例を示す．この例では，プログラム 3 は OpenCL に組み込む Verilog モジュール，プログラム 4 は RTL Module が組み込まれる OpenCL プログラムである．また，プログラム 5 では RTL Module との接続について記

プログラム 3: 組み込まれる Verilog プログラム

```
module modTest (
 input  wire         clock,
 input  wire         resetn,
 input  wire         ivalid,
 output wire         oready,
 input  wire         iready,
 output wire         ovalid,
 input  wire [31:0]  in,
 output wire [31:0]  out
 )
    assign ovalid = 1'b1; // ignored
    assign oready = 1'b1; // ignored
    assign out = in;
endmodule
```

プログラム 4: Module が組み込まれる OpenCL プログラム

```
extern float modTest(float in);

__kernel
void kernel_test(
    __global float* __restrict in,
    __global float* __restrict out
) {
    for (unsigned int i=0; i<SIZE; i++) {
        out[i] = modTest(in[i]);
    }
}
```

述する XML ファイルである．ATTRIBUTES タグで IP コアの特性，INTERFACE タグでポートの接続，REQUIREMENTS で必要な HDL ファイル，C_MODEL タグで CPU エミュレーションに使用する関数の定義を行う．この例では入力をそのまま出力しており，出力に必要なレイテンシは 1 クロックとなる．

RTL Module を組み込む場合の合成の流れを図 2 に示す．はじめに aocl コマンドを用いて，Verilog モジュールと xml ファイルの結合，ライブラリ化を行う．この際，XML ファイルの記述に漏れがないかなどの確認が行われる．次に，aoc コマンドを用いて RTL Module を呼び出す OpenCL デバイスプログラムと先述したライブラリの結合および合成が行われ，bitstream が生成される．その後は通常の OpenCL と同様，プログラム実行時または事前に FPGA にコンフィグレーションを行い計算をオフロードすることが可能である．

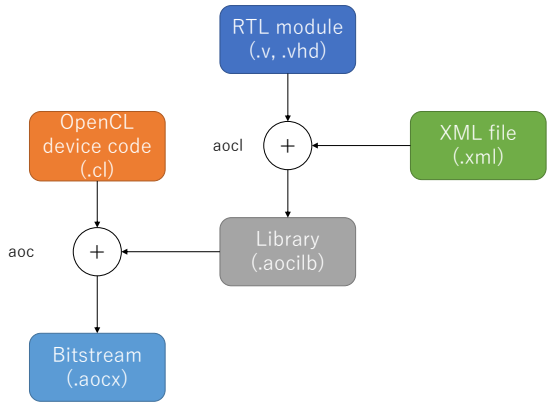

図 2: RTL module を組み込む場合の合成の流れ

プログラム 5: XML の記述

```
<RTL_SPEC>
  <FUNCTION name="modTest" module="modTest" >
    <ATTRIBUTES>
      <IS_STALL_FREE value="yes" />
      <IS_FIXED_LATENCY value="yes" />
      <EXPECTED_LATENCY value="1" />
      <CAPACITY value="1" />
      <HAS_SIDE_EFFECTS value="no" />
      <ALLOW_MERGING value="yes" />
    </ATTRIBUTES>
    <INTERFACE>
      <AVALON port="clock" type="clock" />
      <AVALON port="resetn" type="resetn" />
      <AVALON port="ivalid" type="ivalid" />
      <AVALON port="oready" type="oready" />
      <AVALON port="ovalid" type="ovalid" />
      <AVALON port="iready" type="iready" />
      <INPUT  port="in" width="32" />
      <OUTPUT port="out" width="32" />
    </INTERFACE>
    <REQUIREMENTS>
      <FILE name="verilog/modTest.v" />
    </REQUIREMENTS>
    <C_MODEL>
      <FILE name="c_model.cl" />
    </C_MODEL>
  </FUNCTION>
</RTL_SPEC>
```

3. OpenCL と SPGen の統合による最適化

3.1 OpenCL と SPGen の問題点

FPGA のプログラミングとして OpenCL を用いる場合のメリットはその簡便さにある．高位合成技術により，きわめて通常のプログラミングに近い形で FPGA への回路を生成することができる．しかし，どのような回路が生成されるかについては明らかではない．特に，FPGA 効率的な計算に重要なパイプライン化されるかどうかはユーザの記述方法に依存し、その記述方法については経験が必要となる．明示的に行うためには，Channel による記述が有効であるが，channel を用いて明示的に（擬似的に）パイプラインを作ろうとするとプログラムを大きく変更する必要がある．また，Stratix 10 において channel の仕様が推奨されない．このような背景があり，OpenCL を用いた FPGA プログラミングは複雑となってしまう．

また、ループの unrolling を行う場合，ループ長さが unroll 段数で割り切れない場合に ii(initiation interval, 何クロックごとにパイプラインにデータを 1 つ流し込むか) が 1 にならない場合がある．これはループ内の処理に依存しており，単なるメモリコピーの場合は発生せず，計算を行っている場合に発生する．Intel 社の FPGA 向け最適化に関するドキュメントでは full unroll が推奨されているが，loop 長が長い場合に full unrolling を行うと膨大なリソースを必要とするため適用不可能な場合も存在する．そのため，ii を 1 に保証するためにはユーザが手動で最適化を行う必要がある．unroll を用いずに OpenCL の vector type を利用する方法もあるが，プログラムを大きく変更する可能性や，vector 長が 16 までといった制約が存在するため，unroll を用いるのがより汎用的である．

一方，SPGen によるプログラミングでは、パイプラインを直接記述するため明示的な効率的な記述が可能である．ただし、SPGen は FPGA 上におけるメモリアクセスに自由がなく，ホスト側でデータの整形や順序の入れ替えといった操作を行ったうえでデータ転送を行わなければならない．また SPGen shell がサポートする FPGA ボードは限られているといった問題もある．

3.2 SPGen による OpenCL の最適化

そこで，我々は SPGen によって生成された，パイプライン化された HDL モジュールを OpenCL に RTL Module として組み込むことで FPGA にオフロードする計算の最適化を行うことを目標とする．OpenCL のみで記述する場合における最適化の複雑さ等に対し，SPGen で計算を記述することでパイプライン化されることを保証することが可能である．また，OpenCL の柔軟なメモリアクセスの機構を活用可能であるため前述した FPGA 上でのメモリアクセスの制約を緩和することが可能である．さらには OpenCL を用いた開発に対応する Intel FPGA は多く存在するためそれらに容易に移植可能である．

最適化の例として，loop unrolling を SPGen 内部で行うことにより，ループ間のデータ依存性がない限り ii を 1 にすることが保証できる．これは，SPGen で生成される IP コアは必ずうパイプライン化されるためである．この場合，ループ長が割り切れないケースに対応する必要があるが，これは OpenCL 側で mask を生成し，SPGen 側でそれに応じて処理を行えばよい．この操作自体はループ間のデータ依存性に影響を与えないため，ii を 1 にすることが可能で

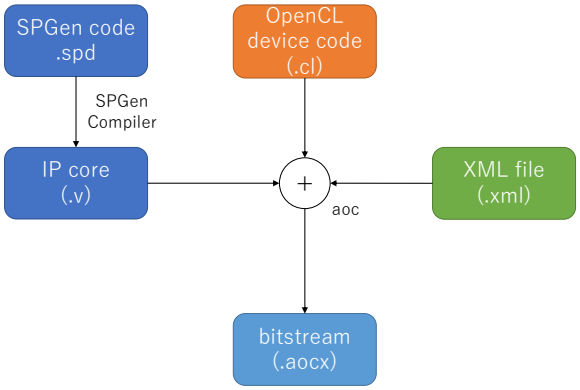

図 3: SPGen を組み込む場合の合成フロー

ある．

また，本研究では計算パイプラインから SPGen に変換を行うためのフレームワークの一部として SPGenC2SPD [11] を使用する予定である．C2SPD ではステンシル計算における時間発展部分のループの unrolling を用いた最適化について触れており，これは複数イテレーションの計算を行う 1 つのパイプラインを生成することで実現している．結合されたイテレーション間でのメモリアクセスは不必要となるため，メモリバンド幅に影響を与えることなくより高い性能を得られることが可能である．

序論でも述べた通り，我々は将来的には SPGen+SPGen に変換を行う OpenMP コンパイラの設計・実装を計画しており，これによってユーザが FPGA 向けに最適化を行うコストを削減できる可能性がある．

3.3 RTL Module 機構を用いた SPGen の組み込み

SPGen フレームワークを使用して生成した IP コアは Avalon ST もしくは Avalon ST インターフェースに準拠している．そのため前述した RTL Module として SPGen を容易に組み込むことが可能である．ただし sop，eop 信号の扱い方が問題となる．SPGen shell に SPGen IP コアを組見込む場合，SPGen IP コアに入力データを供給する Avalon ST インターフェースから供給される startofpacket，endofpacket の信号が使用される．startofpacket はストリームとして供給されるデータの先頭を示す信号である．endofpacket はストリームとして供給されるデータの終端を示す信号である．しかし RTL Module の機構を用いて OpenCL に SPGen を組み込む場合，Avalon ST から供給される startofpacket，endofpacket 信号の接続がサポートされていない．そこで OpenCL 側でそれらの信号を生成し，SPGen IP コアにデータとして供給することで対応する．なお，SPGen モジュールを組み込む場合のコンパイルの流れは図 3 のようになる．

4. 関連研究

4.1 C2SPD

C2SPD[11] とは，SPGen 向けの C 言語フロントエンドであり，独自の指示文ベースで記述されたプログラムを spd プログラムおよびランタイム呼び出しを含めたホストプログラムに変換を行う LLVM ベースの高位合成フレームワークである．ユーザは指示文を用いてオフロード対象とするループを指定し，コンパイラはループの依存解析を行い SPGen に変換可能であれば変換する．なお，変換不可能なものについては本フレームワークでは扱うことができない．我々はこの C2SPD の処理系を活用し，OpenMP target で記述された計算のうち，ユーザによって指定された，もしくは OpenMP コンパイラによって自動検出された SPGen に変換可能な処理を変換することを想定している．また，それ以外の部分については OpenCL に変換することで FPGA 上での回路における柔軟性を確保する．

4.2 Open Accelerator Research Compiler

OpenARC (Open Accerelator Research Compiler) とは，米オークリッジ国立研究所により開発が行われている，GPU・FPGA・Xeon Phi Coprocessor などのアクセラレータに対応するコンパイラである．FPGA については，OpenACC プログラミングを使用し Intel FPGA 向け OpenCL に変換することでサポートしている [12], [13]．ただし，FPGA 向け最適化に対応するために OpenACC の独自の拡張をおよび独自の openarc directive が使用される．FPGA への最適化を行うためにはそれらの独自拡張を理解する必要があり，最適化という点については依然として複雑である．

5. 評価

5.1 評価環境

評価では，筑波大学計算科学研究センターで運用されている PPX (Pre-PACS Version X) システムの 1 ノードを使用する．評価環境を表 1 に示す．使用する FPGA ボードは Bittware 製の A10PL4[14] であり，本ボードは Arria10 GX FPGA を搭載している．なお，後述するラプラス方程式を用いた評価では SPGen のみを用いた場合との比較を行う．SPGen は A10PL4 をサポートしていないため DE5A-Net FPGA を使用する．DE5A-Net の詳細は表 2 のとおりである．DE5A-Net は A10PL4 同様 Arria 10 FPGA を搭載している．チップは異なるが，スピードグレードは同一であることから周波数に影響を与えるハードウェアの条件は同じである．

5.2 ラプラス方程式を用いた評価

4 点ステンシル計算であるラプラス方程式を使用し，

表 1: 評価環境

CPU	Xeon E5-2660 v4 @ 2.00GHz x 2
RAM	DDR4-2400 8GB x 8
GPU	NVIDIA Tesla P100 PCIe x 2
FPGA Board	BittWare A10PL4
FPGA	Intel Arria10 FPGA GX115N3F40E2SG
InfiniBand	Mellanox ConnectX-4 EDR
OS	CentOS 7.3 64bit
FPGA Compiler	Intel FPGA SDK for OpenCL 17.1.2.203
FPGA Compiler Option	-no-interleaving=default
Host Compiler	GNU C Compiler 4.8.5

表 2: SPGen との比較に使用する評価環境

FPGA Board	Terasic DE5A-Net
FPGA	Intel Arria10 FPGA 10AX115N3F45I2SG
FPGA Compiler	Intel Quartus Prime 16.1

プログラム 6: ラプラス方程式

```
#pragma coalesce // for opencl
for (unsigned int i=1; i<SIZE-1; i++) {
    for (unsigned int j=1; j<SIZE-1; j++) {
        out[i*SIZE+j] =
            (
                in[(i+1)*SIZE+(j)] +
                in[(i-1)*SIZE+(j)] +
                in[(i)*SIZE+(j+1)] +
                in[(i)*SIZE+(j-1)]
            );
    }
}
```

SPGen だけで記述した場合，OpenCL のみで記述した場合，OpenCL に SPGen を組み込んだ場合の 3 つの比較を行う．問題サイズは 1024×1024 とする．プログラム 6 は C 言語を用いた場合のラプラス方程式の実装である．SPGen のみで記述する場合，もしくは OpenCL に SPGen を組み込む場合は，このプログラムにディレクティブを追加したものを C2SPD を用いて生成した IP コアを使用する．OpenCL のみで記述する場合は channel などを用いた, プログラムの構造を大きく変更する必要のある最適化は行っていない．本研究では OpenCL+SPGen に変換を行う OpenMP コンパイラの実装を行う予定であり，OpenMP で記述した場合と同等のプログラミングコストで記述された OpenCL プログラムの性能を評価するためである．OpenCL を FPGA に最適化する場合については今後評価を行う．

5.2.1 演算のベクトル化に関する評価

はじめにベクトル化に関する評価を行う．ベクトル化を行うループはプログラム 6 の j ループであり，OpenCL のみで記述する場合においては unroll ディレクティブの追加により行う．unroll されたループにおいて iteration 間の依存性がない場合，unroll された処理は並列に実行されることが期待される．なおベクトル長は 1, 4, 8, 16 の 4 つとする．

はじめに SPGen のみを使用する場合，OpenCL のみを使用する場合，OpenCL に SPGen を組み込む場合の 3 種類のリソース内訳を示す．表 3 は動作周波数，表 4 は ALM（Adaptive Logic Module）の使用率，表 5 は M20K の使用率を示している．

動作周波数では，それぞれベクトル長を長くすることで周波数の低下が確認される．特にに OpenCL に SPGen を組み込んだ場合において，ベクトル長が 1 と 16 の場合において約 2 割程度低下している．ただし SPGen のみを使用した場合の周波数は上回っていることが確認できる．

ALM では，3 種類ともにベクトル長さに応じて若干増加していることが確認できる．なおベクトル長が 16 の場合，OpenCL のみの場合は他と比較し ALM が 3 4%ほど多く使用されている．M20K では，ベクトル長が 8 までの場合は OpenCL のみの場合の使用率がより高くなっているが，16 の場合に逆転しており，OpenCL に SPGen を組み込んだ場合に使用率が 4%ほど高くなっていることが確認できる．

次に実行性能について評価を行う．計測については計算が開始してから終了するまでとし，Host-FPGA 間の通信は含まないものとする．ベクトル長と性能の関係を図 4 に示す．はじめに SPGen のみを用いる場合と OpenCL に SPGen で生成された IP コアを組み込む場合に注目する．ベクトル長が長くなるにつれ性能が向上している一方，性能向上率が下がっていることが確認できる．これは，ベクトル長が長くなることにより要求メモリバンド幅が増幅し，メモリアクセスにおけるストールが増えているためであると考えられる．また，動作周波数が低下していることも要因である．

一方，OpenCL のみを用いて記述した場合，性能が低下していることが確認できる．これは，j ループにおける initiation interval が大きくなっているためである．表 6 にベクトル長と j ループの initiation interval の関係を示す．例えばベクトル長が 16 の場合の理論性能は以下のように計算することが可能である．なお実際の性能はより低くなっており，要因としては要求メモリバンド幅の増幅によるストールの増加などが考えられる．

$$237.50(MHz) * 4 * 16(FLOPS)/1024/165 \simeq 0.089(GFLOPS)$$

このようにループのイテレーション数を unroll 数で割り切れない場合はコンパイラの最適化がうまく行われない．j ループの ii を 1 にするためには，ユーザによる手動最適化が必要となる．

5.2.2 複数タイムステップを行うカーネルに関する評価

次に，複数タイムステップの実行を 1 度に行うカーネルに関する評価を行う．例えば UC が 4 の場合，4 イテレー

表 3: VL frequency (MHz)

	1	4	8	16
SPGen Only	220.56	221.43	220.9	209.07
OpenCL Only	275.00	237.75	255	245.83
OpenCL + SPGen	294.37	281.35	262.50	226.39

表 4: VL ALMs (%)

	1	4	8	16
SPGen Only	7.9	8.8	10.0	11.1
OpenCL Only	10.6	12.7	14.0	15.6
OpenCL + SPGen	10.7	10.8	11.2	12.0

表 5: VL M20K (%)

	1	4	8	16
SPGen Only	-	-	-	-
OpenCL Only	16.3	20.4	21.9	24.8
OpenCL + SPGen	11.6	14.3	19.1	29.1

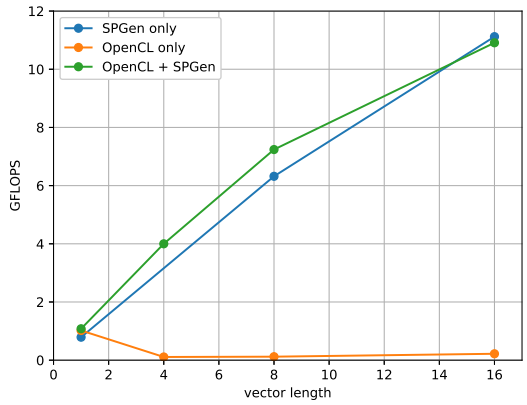

図 4: laplace vector

表 6: initiation interval

vector length	1	4	8	16
ii	1	30	101	165

ション分のラプラス方程式の計算を行うカーネルとなる．SPGen で記述する場合はディレクティブの追加により 4 イテレーションの計算を行うパイプラインの生成を行う．OpenCL で記述する場合は，プログラム 6 のループを 4 つカーネル内に記述する．in，out については適宜入れ替えるものとする．

はじめに，周波数，ALM の使用率，M20K の使用率についてそれぞれを表 7，8，9 に示す．M20K とは FPGA 上に実装されているオンチップメモリである．外部メモリである DRAM と比較し高いバンド幅を持ち，固定レイテンシでメモリの読み書きを行うことが可能である．

動作周波数については，ベクトル化を行った場合と同様に unroll 段数が大きくなるに連れ周波数が低下することが確認できる．SPGen のみの場合は低下は僅かである一方，OpenCL のみの場合，OpenCL に SPGen を組み込む場合の低下率は大きく，それぞれ最大で約 4 割，約 2 割低下している．ただし，OpenCL に SPGen を組み込んだ場合の周波数は SPGen のみの場合の周波数を上回っており，これはベクトル化においても同様である．

ALM についてもベクトル化における評価と同様，OpenCL のみを使用する場合に対し SPGen のみを使用する場合および OpenCL に SPGen を組み込む場合において使用率の増加が抑えられていることが確認できる．unroll 段数が 1 と 16 の場合に注目すると，OpenCL のみの場合は 15.2%も増加している一方，SPGen のみ，OpenCL に SPGen を組み込む場合はそれぞれ 2.4%, 1.5%と抑えられていることが確認できる．したがって，タイプステップのループを unroll する場合において，SPGen を組み込むことでリソース使用量を抑えることが可能であるとい言える．

M20K についても同様，OpenCL のみを使用する場合に対し OpenCL に SPGen を組み込む場合において使用量を抑えられていることが確認できる．ベクトル長が 16 の場合，OpenCL+SPGen の場合には約 20%節約できていることが確認できる．

次にそれぞれの実行性能について図 5 に注目する．縦軸は性能，横軸は unroll 段数である．結果では，OpenCL に SPGen を組み込んだ場合と SPGen のみの場合は unroll 段数が大きくなるごとに性能も向上していることが確認できる．これは，複数イテレーションの計算を行う 1 つのパイプラインが生成されるためで，イテレーション間の同期のようなものが不必要であるためである．また周波数の差から OpenCL に SPGen を組み込んだ場合により高い性能を得られている．

一方，OpenCL のみを利用した場合は性能が全く向上せず，unroll 段数を大きくすることで性能が低下していることが確認できる．例えば OpenCL で 4 つのイテレーションを行うように記述を複製した場合，コンパイラによってイテレーションが 1 つのブロックとなる．それぞれのブロックは前のイテレーションの計算のブロックに依存しており，依存するブロックでの計算が終わらないと実行できないためである．したがって FPGA 上には 4 つのブロックの回路が実装されている一方逐次的に動作するため，性能は向上しない．コンパイラのループ解析やメモリ依存性解析が不十分であり，このような結果になると考えられる．なお channel などをつかって手動で最適化することで擬似的に 1 つのパイプラインを形成することは可能であると考えられるが未確認である．ただし，より多くのリソースが必要になる可能性が高く，その場合においても OpenCL に SPGen を組み込む場合が有利であると考えられる．

表 7: UC frequency (MHz)

	1	4	8	16
SPGen Only	220.56	240.27	220.70	212.86
OpenCL Only	275.00	237.50	177.78	179.63
OpenCL + SPGen	294.34	268.38	267.50	238.66

表 8: UC ALMs (%)

	1	4	8	16
SPGen Only	7.9	8.3	9.0	10.3
OpenCL Only	10.6	13.6	17.7	25.8
OpenCL + SPGen	10.7	10.8	11.2	12.2

表 9: UC M20K (%)

	1	4	8	16
SPGen Only	-	-	-	-
OpenCL Only	16.3	23.2	32.5	51.1
OpenCL + SPGen	11.6	14.8	20.2	31.2

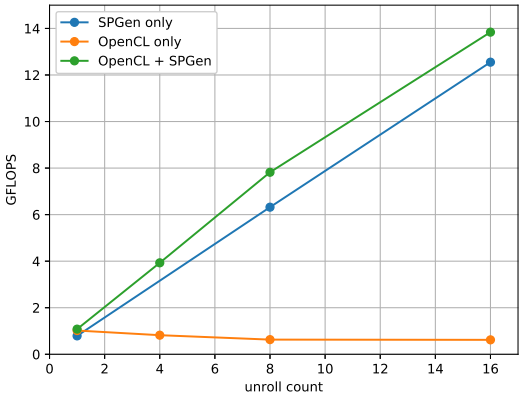

図 5: laplace iteration unrolling

5.3 疎行列ベクトル積を用いた評価

次に疎行列ベクトル積を用いた評価を行う．SPGen を OpenCL に組み込んで使用することにより FPGA 上でのメモリアクセスが柔軟となったため，このようなランダムアクセスや関節参照を必要とする計算も記述することが可能となっている．疎行列の格納方式として CRS 形式を用いる．なお，現時点では内積および加算部分を SPGen で書き換えたのみであり最適化は行えていない．今後の方針としては内積部分のベクトル化による最適化を行い，SPGen で行う場合と OpenCL で行う場合について比較を行う予定である．

評価環境は表 1 と同様である．また使用したデータセットを表 11 に示す．現時点での評価は不十分であり，今後より多くのデータセットを使用し，非ゼロ要素数による影響などを含めて評価する必要がある．

はじめに，OpenCL のみで実装した場合，内積部分を SPGen で記述した場合のそれぞれの回路のリソース内訳について表 10 に示す．各項目とも非常に類似しており大きな差はない．ただし OpenCL のみで実装した場合の動作周波数は OpenCL+SPGen で実装した場合に対し若干高くなっていることが確認できる．

表 10: SpMV の各リソース内訳

	frequency (MHz)	ALM (%)	M20K (%)
OpenCL Only	303.57	10.4	14.6
OpenCL + SPGen	283.33	10.6	13.8

表 11: 使用した疎行列データセット

name	size	num. of non-zeros
dw2048	2048×2048	10114
dw4-96	4096×4096	41746

表 12: SpMV の実行性能 (MFLOPS)

type	dw2048	dw4096
OpenCL only	274	541
OpenCL + SPGen	268	518

次に実行性能を表 12 に示す．それぞれの性能は問題規模に依存しており，問題サイズが大きいほど高い性能を示している．また，若干ではあるが OpenCL のみを用いた場合に高い性能を得られていることが確認できる．ただし何らかの最適化を行わない状態において性能が類似するのはラプラス方程式を用いた評価でも同様であり，内積のベクトル化などを行った場合に OpenCL+SPGen で実装した場合に高い性能を得られる可能性がある．これらについてより評価を進めていく予定である．

6. Stratix10 への移植について

本章では Stratix10 への移植に関する事前評価を行う．比較対象とする Stratix 10 の FPGA ボードは Nallatech 製の 520N L tile[15] および DE10 Pro[16] とし，それぞれの評価環境の概要を表 13，14 に示す．520N のチップのスピードグレードは 2，DE10Pro は 1 である．よって，DE10Pro 向けに合成した回路の周波数は 520N 向けに合成したものより高くなることが予想される．ただし BSP の作り込み方が非常に異なっており，520N では 4 チャンネル分のメモリを扱えるようになっている一方 DE10Pro では 1 チャンネルのみ扱うことができるなどの違いがある．そのためあくまでも参考値であることに注意する必要がある．

はじめに，これらのボードおよび A10PL4 FPGA ボードを用いて，空のデバイスプログラムを合成した場合の周波数を表 15 に示す．OpenCL を用いて高位合成を行う際の最大の周波数がこの値となると考えられる．520N の周波数に注目すると，A10PL4 からの向上率が 1 割にも満たないことが確認できる．その一方，DE10Pro においては 4 割程度向上していることが確認できる．

表 13: 520N

FPGA Board	Nallatech 520N L tile
FPGA	Intel Stratix 10 FPGA 1SG280LN2F43E2VG
FPGA Compiler	Intel FPGA SDK for OpenCL 18.0.1.261

表 14: DE10Pro

FPGA Board	Terasic DE10Pro
FPGA	Intel Stratix 10 FPGA 1SG280LU2F50E1VG
FPGA Compiler	Intel FPGA SDK for OpenCL 18.1.0.222

表 15: 空のデバイスコードを合成したときの fmax

Board	fmax frequency (MHz)
A10PL4	396.19
520N	425.53
DE10Pro	575.7

表 16: 簡易的な spd プログラムを OpenCL に組み込んだ場合の動作周波数

Board	1x frequency (MHz)
A10PL4	315.00
520N	383.33
DE10Pro	575.00

次に，SPGen で生成された IP コアを OpenCL に組み込んで合成した場合の周波数を表 16 に示す．なお，使用する spd プログラムは入力をそのまま出力する簡易的なものである．520N に注目すると A10PL4 からの向上率は 2 割弱である．一方 DE10Pro においては，空のデバイスコードを合成した場合と同様 575 MHz と高い周波数を示している．

これらの結果から，520N への移植においてはあまり周波数の向上が期待できない一方，DE10Pro への移植においては 1.5 倍ほどの周波数向上が期待できると考えられる．ただしこれらはコンパイラの成熟度や BSP の成熟度などに依存するため，今後改善される可能性がある．

7. 結論

本研究では，RTL Module の機構を用いて OpenCL に SPGen を組み込むことによる最適化に関する評価を行った．ラプラス方程式を用いた評価では OpenCL のみで記述した場合，SPGen のみで記述した場合，OpenCL に SPGen を組み込む場合の 3 つを用いて評価を行い，本手法の有用性を示すことができた．具体的には以下のとおりである．

- SPGen のみを使用する場合と同等の性能が得られた
- OpenCL のみで記述した場合に対し高い性能を示した

また，OpenCL のみを用いた場合には FPGA 向け最適化をユーザが十分に行わないと高い性能を得られない一方，今後実装する予定である OpenCL + SPGen に変換を行う OpenMP コンパイラではディレクティブの追加のみで FPGA で高い性能を得られる可能性を示した．

SPGen のみを利用する場合に対しては，既存の OpenCL フレームワークを活用していることにより移植性や FPGA 上でのメモリアクセスの柔軟性において本手法が有利である．

今後の課題としては他の計算を用いた評価や，既存の OpenMP の仕様をどのように割り当てるかに関する考察を行う予定である．それらを行ったのち，omni-compiler へ提案手法の実装を行う予定である．

さらなる課題としては，すでに提案されている OpenCL を用いた FPGA 間直接通信機構 [17] や FPGA-GPU 直接通信機構を持ちいた応用などが考えられる．

8. 謝辞

本研究の一部は，理化学研究所計算科学研究センターと筑波大学計算科学研究センターの共同研究「ポスト京の並列プログラミング環境およびネットワークに関する研究」による．

参考文献

[1] OpenCL Overview. https://www.khronos.org/opencl/
[2] OpenACC https://www.openacc.org/
[3] OpenMP http://www.openmp.org/
[4] Intel FPGA SDK for OpenCL https://www.altera.com/products/design-software/embedded-software-developers/opencl/overview.html
[5] Intel FPGA SDK for OpenCL Programming Guide https://www.altera.com/en_US/pdfs/literature/hb/opencl-sdk/aocl_programming_guide.pdf
[6] Intel FPGA SDK for OpenCL Best Practices Guide https://www.altera.com/en_US/pdfs/literature/hb/opencl-sdk/aocl-best-practices-guide.pdf
[7] Strategies for Optimizing Intel Stratix 10 OpenCL Designs https://www.intel.com/content/www/us/en/programmable/documentation/mwh1391807516407.html#ugg1520272263788
[8] Sano, Kentaro, and Satoru Yamamoto. ”FPGA-Based Scalable and Power-Efficient Fluid Simulation using Floating-Point DSP Blocks.” IEEE Transactions on Parallel and Distributed Systems 28, no. 10 (2017): 2823-2837.
[9] Arria 10 FPGA https://www.altera.com/products/fpga/arria-series/arria-10/overview.html
[10] Intel Stratix 10 FPGA https://www.intel.com/content/www/us/en/products/programmable/fpga/stratix-10.html
[11] Lee, Jinpil, Tomohiro Ueno, Mitsuhisa Sato, and Kentaro Sano. ”High-productivity Programming and Optimization Framework for Stream Processing on FPGA.” In Proceedings of the 9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies, p. 5. ACM, 2018.
[12] Lee, Seyong, Jungwon Kim, and Jeffrey S. Vetter. “OpenACC to FPGA: A Framework for Directive-Based High-Performance Reconfigurable Computing.” In Parallel and Distributed Processing Symposium, 2016 IEEE International, pp. 544-554. IEEE, 2016.
[13] Lambert, Jacob, Seyong Lee, Jungwon Kim, Jeffrey S. Vetter, and Allen D. Malony. ”Directive-Based, High-Level Pro-

gramming and Optimizations for High-Performance Computing with FPGAs." In Proceedings of the 2018 International Conference on Supercomputing, pp. 160-171. ACM, 2018.

[14] A10PL4 PCIe FPGA Board https://www.bittware.com/fpga/intel/boards/a10pl4/

[15] 520N Stratix 10 FPGA - Nallatech https://www.nallatech.com/store/fpga-accelerated-computing/pcie-accelerator-cards/nallatech-520-compute-acceleration-card-stratix-10-fpga/

[16] DE10Pro Stratix 10 FPGA - Terasic https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=13&No=1144&PartNo=1

[17] Kobayashi, Ryohei, Yuma Oobata, Norihisa Fujita, Yoshiki Yamaguchi, and Taisuke Boku. "OpenCL-ready High Speed FPGA Network for Reconfigurable High Performance Computing." In Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region, pp. 192-201. ACM, 2018.