Vol.2019-HPC-170 No.3
2019/7/24

ストリーム計算による最適化を併用する FPGA向けOpenMPコンパイラの試作

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：書き換え可能なハードウェアである FPGA の高性能計算への応用に注目が高まっている中，そのプログラミングの生産性が課題となる．近年では，従来からの一般的なハードウェア記述言語 (Hardware Description Language, HDL) の複雑さ，困難さを緩和するために OpenCL などを用いた高位合成が導入され注目を集めている．OpenCL では platform 間での可搬性などが大きく改善されており，処理の記述自体の困難性は非常に改善された一方，その最適化は非常に困難なものでありさらなる改善が必要である．本研究では，より低レベルでの最適化を行うためにストリーム記述フレームワーク SPGen を用いたストリーム計算を併用することでより柔軟な最適化を行うことを目標とし，また生産性を高めるための OpenMP コンパイラについて試作を行う．本研究では、OpenMP と同等な機能を持つ OpenACC について実装を行い、提案手法である OenCL と SPGen に変換する場合と OpenCL のみの場合の 2 つを omni-compiler へ実装し，同一の OpenACC プログラムを使用し評価を行った．その結果、本提案では、OpenCL と同等な性能を持つコードを生成できることを確認し、さらなる最適化についての検討を行った。

1. 序論

ムーアの法則の終焉を見据え，様々なデバイスの高性能計算への応用に関する研究が行われている．その中の 1 つとして，Field Programmable Gate Arrays (FPGA) が注目を集めている．FPGA とは書き換え可能な回路であり，特定の計算に特化した回路を生成することが可能である．定められた命令セットに従い複数のコアを用いて並列に計算をおこなう CPU・GPU とは異なり，FPGA では計算そのものがハードウェアとなりパイプライン並列に処理されるため高効率な回路を生成することも可能である・これまでにもいくつかのワークロードにおいて FPGA を用いた高性能計算に関する研究が行われている．[1], [2], [3]

FPGA を高性能計算で利用するための課題の 1 つがプログラミング手法となる．高性能計算において FPGA が広く一般的に使用できるようになるためには，FPGA に関する知識が乏しくとも記述可能であり，かつコンパイラによって十分に最適化されるようなプログラミング手法が必須であるが，現時点では存在しない．従来からの一般的な手法であるハードウェア記述言語 (Hardware Description Language: HDL) を用いた記述方法は，詳細な回路の挙動を記述できる一方で非常に複雑であり，長期的な開発を要するため高いコストを伴う．近年では，OpenCL[4] などの高級言語を用いた高位合成が導入され，そのような HDL を用いた開発の複雑性・困難性が改善されつつある．その一方で，本来 OpenCL は GPU などに対する並列プログラミングのためのフレームワークであり FPGA のパイプラインを明示的に記述することは困難である．Oak Ridge National Laboratory の OpenACC[5] を用いて OpenCL プログラミングの生産性を改善する研究 [6], [7] では，OpenARC 独自の pragam を用いてユーザの責任で記述する必要があり，がある程度生産性は改善されているものの依然として FPGA に関する十分な知識が必要となる．また，OpenARC pragma を用いた最適化に対応していない計算については OpenCL 同様生産性の課題が残る．

そこで我々は，より低レベルでの，パイプラインレベルでの最適化が可能な SPGen を用いる OpenMP[8] コンパイラの設計を行っている．SPGen では，OpenCL では困難なパイプラインの明示的な記述が可能となるため，ユーザの記述した OpenMP プログラムを最適化した上で SPGen に変換するような最適化が可能となる．ユーザの記述した OpenMP プログラムをコンパイラによりサイクルレベルでの最適化を行うことが可能であれば，ユーザに対する FPGA の要求知識は格段と削減可能であると考えられる．関連研究の C2SPD[9] では，SPGen に変換を行う pragma ベース

[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

プログラム 1: SPD のプログラム例

```
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);
```

の C コンパイラの提案と最適化手法が提案されているが SPGen のメモリアクセスなどにおける制約により記述可能な計算は限られている．そこで，本研究では SPGen で生成されたモジュールを OpenCL に組み込むことで，SPGen の低レベルでの最適化と OpenCL の表現の豊富さを融合することでより多くの計算に対応できるようにする．

本研究の貢献を以下に示す．

- SPGen を用いたストリーム計算と OpenCL を併用することによる最適化手法の提案を行う
- 上記手法を自動で適用するための OpenACC コンパイラの実装の一例を示す

本原稿の以降の構成は次のとおりである．2 章で OpenCL を SPGen の統合による最適化について，また 3 章で関連研究について述べる．4 章では提案手法の実装の概要について述べ，5 章にて提案手法の評価を行い，最後に 6 章で結論を示す．

2. OpenCL と SPGen の統合による最適化

2.1 SPGen の概要

SPGen とは，理化学研究所の佐野らによって開発が行われている，データフローに基づく回路を生成するためのフレームワークである．SPGen では，Stream Processing Description (SPD) とよばれる Domain Specific Language (DSL) を用いて回路を記述する．SPD では，プログラム 1 のように単一代入形式で計算を記述する．SPGen のコンパイラではこの SPD を入力として Data Flow Graph (DFG) を生成し，DFG に基づき Verilog モジュールを生成する．さきほどの SPD を用いた場合，図 1 のような DFG が生成される．青の四角で表現されている部分はパイプラインの遅延段数を調整するために挿入されるバッファであり，FIFO を用いて実装されている．

2.2 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

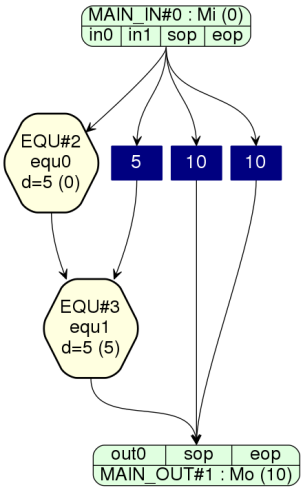

図 1: 生成されたデータフロー図

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．従来は channel を用いて擬似的に pipeline を表現することは可能であったが，Stratix10 向けの最適化においてはその使用が非推奨となっている．

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サイクルレベルでの最適化が実現可能である．SPGen はデータフローに基づく回路を生成するため，FPGA のアーキテクチャと相性がよい．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．

2.3 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データフローを直接記述することでより低レベルでの最適化を行う．データフローを直接的に記述できない OpenCL に対し，直接記述可能な SPGen を用いて最適化した回路を組み込むことで実現する．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．

2.4 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

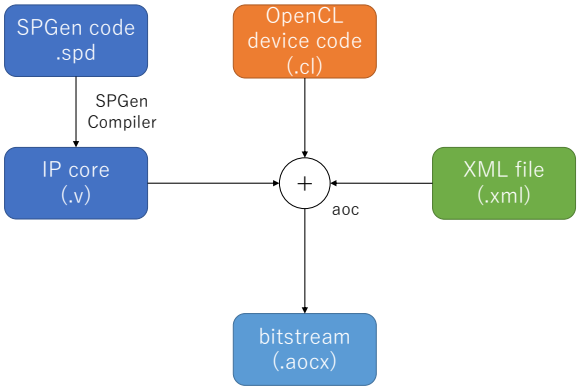

図 2: SPGen を組み込む場合の合成フロー

容易に組み込むことが可能である．ただし SPGen における sop(ストリームの先頭を表す flag)，eop(ストリームの終端を表す flag) 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 2 のようになる．

3. 関連研究

3.1 C2SPD

C2SPD[9] 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．

3.2 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 [6], [7]．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．

4. OpenCL と SPGen への変換の実装

本章では提案手法の実装について概要を述べる．なお，実装は omni-compiler を拡張する形で行う．

4.1 omni-compiler の概要

omni-compiler とは，理化学研究所および筑波大学により開発が行われているコンパイラである．OpenMP や OpenACC への対応に加え，XcalableMP・XcalableACC といった，pragma ベースの PGAS モデルに対応している．

OpenACC においては，GPU や PEZY-SC に対応しており，ユーザの記述したプログラムから CUDA もしくはマルチスレッドモデルの OpenCL プログラムを生成することが可能である．この omni-compiler に対し，FPGA 向けに対応するためシングルスレッドモデルの OpenCL デバイスカーネルの生成および SPGen のプログラムの生成に関する実装の追加を行う．今回は、OpenMP と同等な機能を持つ OpenACC についての変換を行った。

4.2 変換の概要

ユーザの記述した OpenACC プログラムに対し，以下の流れで解析・変換を行う．

(1) SPGen による最適化を行う loop の検出
OpenACC の kernels directive で記述されたオフロード対象領域のうち，loop directive で記述された並列化対象の loop を検出し SPGen を用いた最適化対象とする．loop directive で指定されている for loop では iteration 間のデータ依存性がないことを仮定することが可能であり解析を容易にすることが可能である．

(2) SPGen module との入出力の解析
loop directive で指定された loop の body を解析し，必要な入力および出力を解析する．

(3) SPGen body の vectorize
OpenACC の vector clause が指定されている場合，その値に応じて SPGen を用いた最適化対象の body を vectorize する．

(4) reduction の処理
OpenACC の reduction clause が指定されている場合，指定された変数に対する reduction 処理を追加する．

(5) SPGen コード生成

以上の処理をおこなった loop body を SPGen プログラムに変換する．また，OpenCL デバイスプログラムに SPGen で生成されたモジュールを組み込むための wrapper 関数を追加する．現時点では，SPGen の最適化は行っていない．

4.3 コンパイラによる変換の例

実装を行ったコンパイラを用いたプログラムの変換の例を示す．入力となる OpenACC プログラムはプログラム 2 であり，これは配列 in の総和を求めるプログラムである．このオフロード対象領域に対し kernels directive を用いてオフロード対象であることを示す．オフロード先のデバイスについては device_type clause を用いて指定する．なお，1 つの OpenACC プログラムから複数のデバイス向けに変換を行えるように実装している．次に，並列化対象の for loop に対し loop directive を与える．また，vector clause を用いてベクトル化の指定をしており，この例では 2 を指定している．最後に，reduction clause を用いて変数 acc に対する reduction を指示する．

プログラム 4 は，コンパイラによって変換された OpenCL デバイスカーネルである．入力の OpenACC ではループ長が可変であるため，20 25 行目において bitmask を生成している．36-41 行目では，SPGen module を呼び出すための wrapper 関数の呼び出しを行う．この際，先ほど作成した bitmask とともにデータを引数に与えている．この関数呼び出しにより，実際の回路において SPGen module のパイプラインが組み込まれることになる．

プログラム 3 はコンパイラによって変換された SPGen module である．bitmask の値に応じて入力が有効かどうか判定し，mux (マルチプレクサ) を用いて 0 もしくは有効な値を選択する．13 行目ではベクトルの垂直加算を行う．14 表目では，mALTFP_ACC_PLUS module を呼び出している．この module は入力された値を加算した結果を返す module となっている．この module に対し先ほど垂直加算した値を渡すことで reduction 処理を行う．なお，sop (データストリームの先頭を表す flag) を用いて内部の加算レジスタの値を初期化することが可能である．

なお現時点においては生成れるプログラムは最適なものではなく，生成される OpenCL デバイスカーネルおよび SPGen プログラムは余分な計算を含んでいる状態である．例えば SPGen プログラムにおいては，7 行目のように削除可能な代入が挿入されてしまっている．余分な処理の削除や回路の最適化は今後改善が必要となる．

プログラム 2: 入力となる OpenACC プログラム

```
1  float fpga(
2    float *restrict in,
3    int size
4  )
5  {
6    float acc = 0.0f;
7  #pragma acc kernels device_type(OpenCL_IntelFPGA)
       copyin(in[0:size]) copyout(acc)
8  #pragma acc loop gang(1) worker(1) vector(2) reduction
       (+:acc)
9    for (int i=0; i<size; i++) {
10     acc += in[i];
11   }
12   return acc;
13 }
```

プログラム 3: 生成された SPGen のプログラム

```
1  Name ACC_kernel_fpga_L15_SPGEN;
2
3  Main_In {Mi::in0, in1, in2, sop, eop};
4  Main_Out {Mo::out0, sop, eop};
5  DRCT (Mo::sop, Mo::eop) = (Mi::sop, Mi::eop);
6
7  EQU equ0, local0=in0;
8  EQU equ1, local1=in1;
9  EQU equ2, tmp0=local0;
10 EQU equ3, local2=mux(0.0, tmp0, in2[0]);
11 EQU equ4, tmp1=local1;
12 EQU equ5, local3=mux(0.0, tmp1, in2[1]);
13 EQU equ6, local4=local2+local3;
14 HDL hdl0, 7, out0=mALTFP_ACC_PLUS(local4, Mi::sop[0]);
```

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

プログラム 4: 生成された OpenCL のプログラム

```
#include "acc_cl.h"

__kernel void
    ACC_kernel_fpga_L15_OpenCL_INTELFPGA_DEVICE(
        __global int *size,
        __global float *acc,
        __global float *__restrict__ in
){
  int i;
  int _iter_i_upper_size;
  (_iter_i_upper_size) = (*size);
  for ((i) = (0); (i) < (_iter_i_upper_size); (i) += (1)) {
    float _bitmask_acc;
    unsigned *__p_bitmask_acc;
    float __sopEop;
    unsigned *__p_sopEop;
    extern float ACC_kernel_fpga_L15_SPGEN_wrapper(float attr, float2 in);
    (__p_bitmask_acc) = (((unsigned *) (&_bitmask_acc)));
    (*__p_bitmask_acc) = (0);

    if ((i) < (_iter_i_upper_size)) {
      (*__p_bitmask_acc) |= ((1) << (0));
    }
    if (((i) + (1)) < (_iter_i_upper_size)) {
      (*__p_bitmask_acc) |= ((1) << (1));
    }

    (__p_sopEop) = (((unsigned *) (&__sopEop)));
    (*__p_sopEop) = (0);
    if ((i) == (0)) {
      (*__p_sopEop) |= (1);
    }
    if ((i) == ((_iter_i_upper_size) - (1))) {
      (*__p_sopEop) |= ((1) << (1));
    }

    (*acc) =
    (ACC_kernel_fpga_L15_SPGEN_wrapper (
        __sopEop,
        _bitmask_acc,
        *((in) + ((i) + (1))),
        *((in) + (i))));
  }
}
```

表 1: 評価環境

CPU	Xeon E5-2660 v4 @ 2.00GHz x 2
RAM	DDR4-2400 8GB x 8
FPGA Board	BittWare A10PL4
FPGA	Intel Arria10 FPGA GX115N3F40E2SG
OS	CentOS 7.3 64bit
FPGA Compiler	Intel FPGA SDK for OpenCL 17.1.2.304
Host Compiler	GNU C Compiler 4.8.5

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[11] であり，本ボードは Arria10 GX FPGA を搭載している．また，DDR4-2400 4GB のメモリを 2 つ搭載している．FPGA ボードと Host は PCIe Gen3 を通して接続されており，レーン数は 8 である．

5.2 評価に使用する計算

評価では livermore loops[10] を用いて初期評価を行う．livermore loops とは，コンパイラのベクトル化の性能を検証するためのベンチマークで，実アプリケーションから取り出された 28 の loop から構成される．なお，原稿執筆時点では kernel3, kernel9，および kernel12 のみの評価のみを行っている．これらの計算において，OpenACC から OpenCL のみに変換した場合，OpenACC から OpenCL+SPGen に変換した場合の 2 つに対し，それぞれベクトル帳を 1，4，8 と変化させた場合について合成し評価を行う．

5.3 kernel3 を用いた評価

はじめに kernel3 を用いた評価について示す．表 2 は動作周波数，回路規模を表現する ALM の割合，浮動小数点の演算に使用される DSP の数，ローカルメモリを表す M20K の使用率，また initiation interval の値を示す．ii とは，パイプラインに対し何クロックに 1 度データを供給するかの指標となる．kernel3 においては，OpenCL のみを出力する場合において最適化が不十分であるため ii が 1 とならないケースが存在する．

周波数に着目すると，SPGen を併用する場合は OpenCL のみを使用する場合に対し 1 割弱低くなっていることが確認できる．また，DSP においては 2 倍消費してしまっている．これについては，OpenCL では DSP の FMA モードを使用している一方で SPGen を併用する場合は FMA モードを使用できていないことが要因の 1 つである．M20K についても同様に SPGen を併用する場合により多く使用されていることが確認できる．これは，図 1 の例でも説明したとおり，パイプライン段数を調整するために FIFO が挿入され，ここで M20K が消費されるためである．

実行結果を図 3 に示す．この時の要素数は 1 億である．SPGen を併用する場合は vector 長を長くするにつれ実行

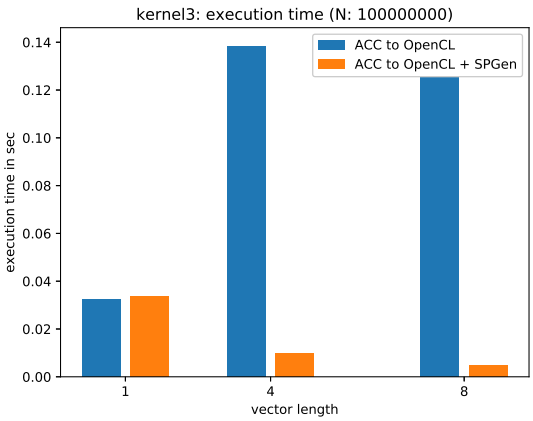

図 3: kernel3 の実行結果

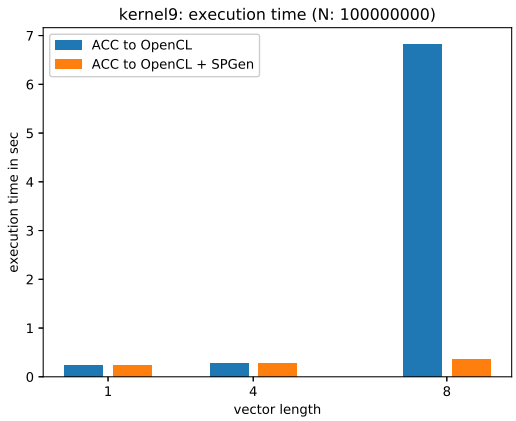

図 4: kernel9 の実行結果

時間が短くなっている一方，OpenCL のみの場合においては長くなっている．OpenCL のみの場合において性能が低下しているのは ii によるものである．なお，先述した通り OpenCL のみを出力する部分の最適化が不十分でないためであり，修正することで同等の性能が得られると考えられる．

5.4 kernel9 を用いた評価

次に kernel9 を用いた評価について示す．リソース内訳を表 3 に示す．周波数については kernel3 同様，SPGen を使用する場合により低い値を示している．とくに vector 長が 4 の場合の差は 2 割弱と，ほかと比較し大きくなっている．一方で vector 長が 8 の場合の差は 1 割弱となっている．また，DSP の使用量にも大きな差が出ている．確認したところ，OpenCL のみを用いる場合については DSP の内積モードを使用している一方，SPGen を併用する場合は内積モードを使用せずにより多くの DSP を使用しているためこのようになっていた．また，M20K についてもより多く使用されているが，kernel3 に対し入力や計算量が多く，より多くの遅延が挿入されるため M20K の使用量も増えたと考えられる．また，OpenCL のみを用いる場合において ii が 2809 と大きな値を示している．vectorize の方法については vector 4 の場合と同様であるため，OpenCL のコンパイラの解析が十分に行われていないと考えられるが詳細は不明である．

実行結果を図 5 に示す．性能については，vector 長が 1, 4 の場合において周波数の差はあるが実行性能の差はほとんどないことが確認できる．理由として，kernel9 ではメモリアクセスが連続ではないため効率が下がっていることが原因の 1 つと考えられるが，詳細は現時点では解析できていない．また，vector 長が 8 の場合において，SPGen を併用する場合は ii が 1 のままであるため性能に変化はないが，一方 OpenCL のみを用いる場合においては先述した通り ii が 2809 となるため西欧が著しく低下している．

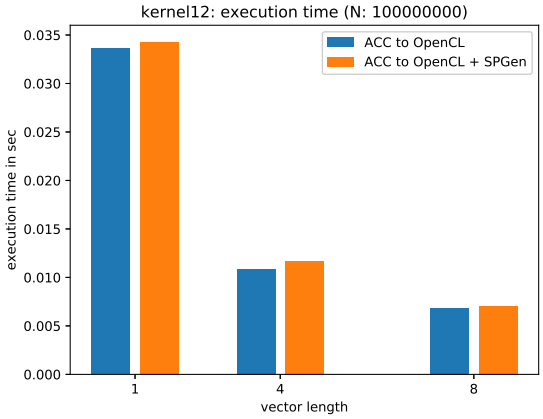

図 5: kernel12 の実行結果

5.5 kernel12 を用いた評価

最後に kernel12 を用いた評価について示す．表 4 に動作周波数，回路規模を表現する ALM の割合，浮動小数点の演算に使用される DSP の数，ローカルメモリを表す M20K を示している．なお，ii についてはすべてとなっている．

周波数については kernel3 同様，若干の差ではあるが SPGen を使用する場合により低い値を示している．また，M20K についても 1%と若干はあるが SPGen を併用する場合により多く消費しており，理由は kernel3 と同様である．

実行結果を図 5 に示す．この時の要素数は 1 億である．どちらも vector 長を長くするにつれて実行時間が短くなっていることが確認できる．ただしわずかではあるが OpenCL のみを出力する場合のほうがより高い性能を示している．

5.6 SPGen の最適化についての検討

3 つのカーネルにおいて，OpenCL のみを出力する場合に対し SPGen による最適化を併用する場合にはより多くのリソースを必要とし，また kernel3 を除いて性能が低いことが確認された．リソースについては，DSP に関しては SPGen を併用する場合において DSP の FMA モードや内積モードが利用できていないことが原因であるが，SPGen

表 2: kernel3 のリソース内訳

	\Vector Length	1	4	8
frequency (MHz)	ACC to OpenCL + SPGen	296.64	259.6	272.75
	ACC to OpenCL	309.98	289.77	288.68
ALMs	ACC to OpenCL + SPGen	11%	11%	11%
	ACC to OpenCL	10%	10%	11%
DSPs	ACC to OpenCL + SPGen	2	8	16
	ACC to OpenCL	1	4	8
M20K	ACC to OpenCL + SPGen	12%	16%	17%
	ACC to OpenCL	12%	14%	14%
initiation interval	ACC to OpenCL + SPGen	1	1	1
	ACC to OpenCL	1	16	32

表 3: kernel9 のリソース内訳

	\Vector Length	1	4	8
frequency (MHz)	ACC to OpenCL + SPGen	263.64	193.72	195.61
	ACC to OpenCL	284.81	271.58	228.67
ALMs	ACC to OpenCL + SPGen	11%	15%	20%
	ACC to OpenCL	12%	14%	22%
DSPs	ACC to OpenCL + SPGen	21	84	168
	ACC to OpenCL	12	48	98
M20K	ACC to OpenCL + SPGen	17%	33%	53%
	ACC to OpenCL	14%	19%	24%
initiation interval	ACC to OpenCL + SPGen	1	1	1
	ACC to OpenCL	1	1	2809

においてもこれらは表現可能である．したがって，実装したコンパイラのコード生成を最適化することでこれら r のリソース使用量を削減可能である．M20K に関しては，SPGen でパイプライン間の遅延をそろえるために FIFO を多用することが原因となっており，これらは shift register に書き換えることで削減可能であると考えられる．

性能については，実装したコンパイラが出力する SPGen のコードの最適化が現状行えていないため，今後の最適化により修正可能と考えられる．現状の実装では，プログラム 3 で示した生成されるコード例の通り，7, 8 行目などのような不必要な代入が生成されてしまっている．これらは SPGen のコンパイラによって削除されることなく回路になってしまうため，このような不必要な代入を削除する必要がある．また，1 つの EQU 式には現在 1 つの演算のみを記述しているが，複数の式を記述可能であるため，複数の EQU 式を統合することでより最適化を行える可能性がある．

6. 結論

SPGen によるストリーム計算を併用する OpenACC コンパイラ，および比較用に OpenCL のみに変換を行う OpenACC コンパイラの実装も行い，livermore loops を用いた評価を行った．評価では，kernel3，また kernel9 の vector 長が 8 の OpenCL のみを用いる場合においてコンパイラの最適過不足により ii が 1 とならず，結果として SPGen を併用する場合により高い性能を得られたケースもあったが，基本的には OpenCL のみを出力する場合に対し 1 割弱の性能低下程度と同等の性能を得られた．また，SPGen を併用する場合により多くのリソースを消費することがわかった．これらは，実装した OpenACC コンパイラの SPD コード生成における最適化を実装することで改善されると考えられる．今後は前述した最適化の実装や，OpenACC だけでなく OpenMP コンパイラの実装を行う．

参考文献

[1] Kenter, Tobias, Gopinath Mahale, Samer Alhaddad, Yevgen Grynko, Christian Schmitt, Ayesha Afzal, Frank Hannig, Jens Frstner, and Christian Plessl. ”OpenCL-based FPGA design to accelerate the nodal Discontinuous Galerkin method for unstructured meshes.” In 2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 189-196. IEEE, 2018.

[2] Fujita, Norihisa, Ryohei Kobayashi, Yoshiki Yamaguchi, Yuma Oobata, Taisuke Boku, Makito Abe, Kohji Yoshikawa, and Masayuki Umemura. ”Accelerating Space Radiative Transfer on FPGA using OpenCL.” In Proceedings of the 9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies, p. 6. ACM, 2018.

[3] 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.

[4] OpenCL Overview. https://www.khronos.org/opencl/

表 4: kernel12 のリソース内訳

	\Vector Length	1	4	8
frequency (MHz)	ACC to OpenCL + SPGen	292.22	277.16	265.45
	ACC to OpenCL	299.04	291.88	277.46
ALMs	ACC to OpenCL + SPGen	11%	11%	11%
	ACC to OpenCL	11%	11%	11%
DSPs	ACC to OpenCL + SPGen	1	4	8
	ACC to OpenCL	1	4	8
M20K	ACC to OpenCL + SPGen	12%	14%	14%
	ACC to OpenCL	12%	13%	13%

[5] OpenACC https://www.openacc.org/
[6] 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.
[7] Lambert, Jacob, Seyong Lee, Jungwon Kim, Jeffrey S. Vetter, and Allen D. Malony. ”Directive-Based, High-Level Programming and Optimizations for High-Performance Computing with FPGAs.” In Proceedings of the 2018 International Conference on Supercomputing, pp. 160-171. ACM, 2018.
[8] OpenMP http://www.openmp.org/
[9] 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.
[10] https://www.netlib.org/benchmark/livermorec
[11] A10PL4 PCIe FPGA Board https://www.bittware.com/fpga/intel/boards/a10pl4/