Vol.2018-HPC-165 No.39
2018/8/1

SPGenのC言語フロントエンドによるループ最適化と性能評価

李 珍泌[1,a)]　上野 知洋[1]　佐藤 三久[1]　佐野 健太郎[1]

概要：半導体技術の停滞によってチップ内の限られたトランジスタを有効に活用して高い性能を達成するためにアプリケーションに特化した専用アーキテクチャが注目を集めている。従来、専用アーキテクチャは Application Specific Integrated Circuit (ASIC) によって実現されていたが、開発期間とコストが増大してきていることから、一部ではプログラミング可能な集積回路を持つ Field Programmable Gate Array (FPGA) が用いられるようになった。しかし、従来のプログラミングモデルであるハードウェア記述言語や高位合成コンパイラでは高い生産性と性能を同時に達成することが困難である。本稿は C 言語の一部を FPGA 化する C 言語ベースのプログラミングモデル C2SPD の提案を行う。FPGA ハードウェアを生成する処理系のバックエンドとしてストリーム計算に特化した FPGA 開発プラットフォームである SPGen を用いる。C2SPD は独自の指示文を含む C 言語コードを受け取り、SPGen の独自言語のコードに変換する。SPGen 処理系はコードを Verilog に変換することで FPGA ハードウェアを生成する。C2SPD の指示文は FPGA 化を行うコード領域の指定やハードウェア最適化に関するヒントを含む。2 次元ステンシル計算を C2SPD で記述し、Intel Arria 10 を用いた性能評価で 175.41 GFLOPS の性能を達成した。この結果は C2SPD の指示文によってベクトル化や Loop Unroll の最適化を施したものであり、最適化なしのハードウェアに比べて 220 倍の性能改善を達成している。

Loop Optimization and Performance Evaluation with C2SPD: SPGen C Front-end

Jinpil Lee[1,a)]　Tomohiro Ueno[1]　Mitsuhisa Sato[1]　Kentaro Sano[1]

1. はじめに

近年のシリコン技術の停滞とチップの消費電力の増加により、限られたトランジスタを有効に活用する手段としてアプリケーションに特化した専用アーキテクチャが注目を集めている。従来、専用アーキテクチャは ASIC で実現されるものが多かったが、設計と実装コストの増加により近年は FPGA を用いた実装が増えてきている。FPGA は様々なアプリケーションをサポートする必要がある HPC の分野で専用アーキテクチャを実現する手段として注目を集めており、様々な研究が進められているが、プログラミングコストの高さが問題とされている。

[1] 理化学研究所 計算科学研究センター
RIKEN Center for Computational Science
[a)] jinpil.lee@riken.jp

本研究は FPGA 上でストリーム計算を実現する C 言語ベースのプログラミングモデル C2SPD を提案する。C2SPD は C 言語のコードに指示文を追加することで FPGA 化するコード領域の指定を行う。処理系は指定されたコード領域の解析を行い、FPGA で実行可能なストリーム計算コアやデータストリームの生成を行う。ベクトル化や Loop Unroll のような FPGA 向け最適化を指示文の節 (clause) として記述することで高度にパイプライン化された FPGA カーネルを生成することができる。数行の C2SPD 指示文を追加することで 2 次元ステンシル計算の最適化された FPGA カーネルが得られることを性能評価で示す。

C2SPD はストリーム計算という限られた分野を対象にすることで効率のいいハードウエアを生成できる反面、対象のコード領域内では様々な制約が存在する。その多くは

制約を超えて記述されたコードが本質的にストリーム計算として実現できないからであるが、今後の FPGA バックエンドの機能拡張や処理系のコード最適化によって対象範囲を広げることも考えられる。

本稿の構成は次のようである。第 2 章では関連研究を述べる。第 3 章では背景知識としてストリーム計算向け FPGA プログラミングプラットフォームである SPGen について述べる。第 4 章では SPGen の C 言語フロントエンドである C2SPD のプログラミングモデルと性能最適化手法について説明を行う。第 5 章では C2SPD 処理系のコード変換の実装について述べる。第 6 章では 2 次元ステンシルコードを C2SPD を用いて記述することで FPGA 化と性能最適化を行い、Intel FPGA を用いた性能評価の結果を示す。第 7 章では処理系によるループ文の FPGA 向け最適化スケジュールの生成に関する課題を述べ、第 8 章では結論と今後の課題について述べる。

2. 関連研究

SystemC[1]、Xilinx Vivado HLS[2]、NEC CyberWorkBench[3] のような従来の High Level Synthesis (HLS) 言語は C 言語などの高レベル言語による FPGA プログラミングを可能にするが、ターゲットアプリケーションは組み込みシステムであるため、HPC アプリケーションの開発には向かない。OpenCL[4] は CPU、GPU 向けに提案されたオープンかつポータブルな並列プログラミングモデルであり、近年は Xilinx や Intel[5] による FPGA 向け処理系が提供されている。C 言語を用いる OpenCL は FPGA のハードウエアの詳細を知らなくても利用可能であるため高い生産性を実現する。汎用プログラミングモデルであるため記述された C 言語コードがハードウエアに変換されることは保証されるが、ハードウエア生成が処理系によって隠蔽されるので性能最適化が困難である。Oak Ridge National Lab で開発されている OpenARC コンパイラ [6] は OpenACC 指示文を OpenCL コードに変換するため、性能最適化に関する OpenCL の課題を共有する。Maxeler 社の MaxCompiler[7] はデータフローを記述する Java ベースの Domain Specific Language (DSL) を提供する。記述範囲をデータフローアプリケーションに限定させることで高い性能を達成するハードウェアを生成するが、独自言語とハードウエアモジュールとして実装されたライブラリを用いてプログラミングするため、既存のプログラミング言語から移行する場合はコードの大幅な修正が必要である。C2SPD は C 言語をベースにストリーム計算というアプリケーション領域に特化することで高い性能を達成するハードウェアを生成する。指示文によるプログラミングモデルを持つため、ユーザが持つ従来のコードから FPGA への移植が容易である。

```
1 Name       core;
2 Main_In    {Mi::x1, x2, x3, x4, sop, eop};
3 Main_Out   {Mo::z1, z2, sop, eop};
4 EQU        equ1, t1 = x1 * x2;
5 EQU        equ2, t2 = x3 + x4;
6 EQU        equ3, z1 = t1 + t2;
7 EQU        equ4, z2 = t1 - t2;
8 DRCT       (Mo::sop, Mo::eop) \\
9          = (Mi::sop, Mi::eop);
```

図 1 SPD コード例

3. ストリーム計算プラットフォーム SPGen の概要

SPGen[8][9] はストリーム計算に特化したハードウェアを FPGA 上で実現する開発プラットフォームである。SPGen のストリーム計算コアは独自の Stream Processing Description (SPD) 言語 [10] で記述される。図 1 に SPD のサンプルコードを示す。SPD コードは 4 つの値 (*x1*-*x4*) を入力として受け取り、2 つの値を出力 (*z1*、*z2*) するストリーム計算モジュールを記述する。計算は *EQU* ラベルと持つ式に Single Static Assignment (SSA) 形式で記述する。*DRCT* ラベルの式で SPGen ハードウエアの制御信号である *sop*、*eop* を入力から出力へ直接つなげる。

図 1 のコードから生成されるハードウェアのイメージを図 2 に示す。足し算の命令レイテンシを 2、掛け算の命令レイテンシを 4 とする。命令レイテンシを隠すために SPGen の処理系はパイプライン化されたハードウェアを生成する。SPGen のハードウェアは計算のレイテンシをすべて静的に解析することでストールしないパイプラインを生成する。途中の計算が異なるレイテンシを持つときは早い方の計算結果を遅延させることで次の演算器に入力データが同時に入るように調整する。図 2 では *t1* の計算に 4 サイクル必要で *t2* の計算は 2 サイクルで済むため、データが揃うまで *t2* の結果を 2 サイクル分遅延させる。

パイプライン化と遅延ノードの挿入を行うことによって SPGen のハードウェアは毎サイクル一つの入力データセット (図 2 の *x1*-*x4*) を受け取り、一つの出力データセット (図 2 の *z1*、*z2*) を出力する。入出力のデータの流れをデータストリームと呼び、このような計算手法をストリーム計算と呼ぶ。SPGen はストリーム計算に特化した DSL と処理系を実現し、効率の高いハードウェア生成を行うが、計算式を SSA 形式で記述することや、ホスト側の制御をランタイム関数呼び出しによって行うなど、プログラミングコストに改善の余地がある。本研究は SPGen を FPGA ハードウェア生成のバックエンドとして用いて C 言語と指示文によるプログラミングモデルを提供することである。

4. C 言語フロントエンド C2SPD の概要

著者らは C 言語のコードを受け取り、SPD コードに変

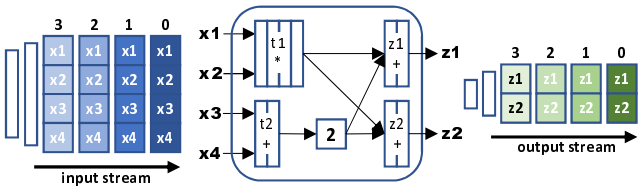

図 2　生成されたハードウェアのイメージ

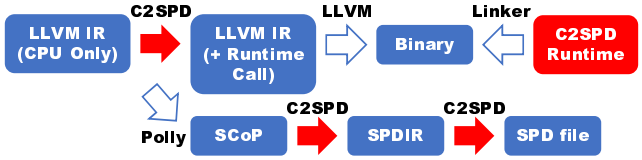

図 3　C2SPD 処理系によるコード変換

換する SPGen の C 言語フロントエンド C2SPD の開発を行った [11]。図 3 に C2SPD によるコード変換の流れを示す。LLVM の C 言語フロントエンド Clang によって C 言語コードが LLVM の中間形式である LLVM IR に変換される。C2SPD は Polly[12] を用いてループ文を SPD コードに変換し、LLVM IR からループ文を削除する。また、FPGA カーネルを実行するランタイム関数呼び出しが LLVM IR に追加される。今までの実装では処理系が見つけたすべてのループ文に対して SPD コードの生成を行った。今回の実装で指示文によるプログラミングモデルを導入することで、対象ループ文を限定し、ベクトル化などの最適化を明示的に指定できるようにした。

リスト 4 に C2SPD の指示文によって記述された 2 次元ステンシルコードを示す。C2SPD は *region* と *offload* の二つの指示文を用いて SPD コード生成を明示的に指定する。*offload* 指示文は処理系に対して C 言語のループ文の SPD コード生成を明示的に指定する。処理系は *offload* で指定されたループ文にのみ、図 3 で示されたコード変換を行う。*offload* 指示文はストリーム計算の性能最適化のために *vector_length* と *unroll_count* の二つの節を提供する。これらの役割とコード変換は次の章で説明する。

region 指示文は *offload* 指示文によって FPGA 化されるカーネルが持つデータ環境のスコープを指定する。リスト 4 からもわかるように、ステンシル計算などは反復法によるデータの更新を行うため、対象ループ文の前後で素直にデータバッファーの生成と CPU と FPGA 間のデータ転送を行うと反復回数に比例するオーバーヘッドが発生する。*region* 指示文によってこれらの処理を時間反復ループ文の前後で行うようにすることでオーバーヘッドを最小限に抑えることができる。*offload* 指示文の *connect* 節は反復法で必要なデータの更新を FPGA 上で行うために与える。リスト 4 では 1 回の反復が終わると *out* のデータを *in* に代入して次の反復に備えるデータの更新を行う。処理系は C2SPD の *connect* 指示文で与えられたデータのペアをストリーム計算の入出力と認識し、次の反復で出力のデータ

```
1  float in[M][N];
2  float out[M][N];
3
4  #pragma spd region
5  for (int t = 0; t < TIME_STEP/UC; t++) {
6  #pragma spd offload vector_length(VL) \\
7          unroll_count(UC) connect(in:out)
8    for (int i = 1; i < M-1; i++) {
9      for (int j = 1; j < N-1; j++) {
10       out[i][j]
11         = (in[i][j-1] + in[i][j+1]
12         +  in[i-1][j] + in[i+1][j]) / 4.0f;
13     }
14   }
15   ...
```

図 4　ステンシル計算の C 言語コード

```
1  Name       kernel0;
2  Main_In    {Mi::src0, iattr, sop, eop};
3  Main_Out   {Mo::dst0, oattr, sop, eop};
4  HDL hdl0, 2, (xxxv00)() = \\
5    mStreamBackward(src0,..,.pBwdCycles(1)>;
6  HDL hdl1, 2+1, (xxxv10)() = \\
7    mStreamFoward(src0,..,.pFwdCycles(1)>;
8  EQU equ0, add110 = xxxv00 + xxxv10;
9  HDL hdl2, 2, (xxxv20)() = \\
10   mStreamBackward(src0,..,.pBwdCycles(N)>;
11 EQU equ1, add170 = add110 + xxxv20;
12 HDL hdl3, 2+N, (xxxv30)() = \\
13   mStreamFoward(src0,..,.pFwdCycles(N)>;
14 EQU equ2, add230 = add170 + xxxv30;
15 EQU equ3, conv240 = add230 * 0.250000;
16 EQU equ4, dst0 = mux(src0, conv240, iattr
       [0]);
17 DRCT (oattr, Mo::sop, Mo::eop) = \\
18       (iattr, Mi::sop, Mi::eop);
```

図 5　ステンシル計算の SPD コード

が入力として使われるようにする。

5. 処理系の実装

本章では C2SPD の処理系の実装について述べる。ベクトル化されていない SPD コードの生成や CPU 側コードのランタイム関数呼び出し生成は [11] で発表済みであるため、本稿ではベクトル化や Loop Unroll が指定されたときの最適化コード生成について述べる。図 5 に図 4 の *offload* 指示文から生成される SPD コードを示す。このコードはまだ最適化されていないものである。ベクトル化を指定する *vector_length* 節や Loop Unroll を指定する *unroll_count* 節にパラメータを与えることで最適化された SPD コードが生成される。

5.1 ベクトル化

CPU プログラミングにおけるベクトル化はコードが持つ並列性を抽出し、複数のデータを一つの命令で処理するベクトル命令を生成することで性能最適化を行う。SPGen にお

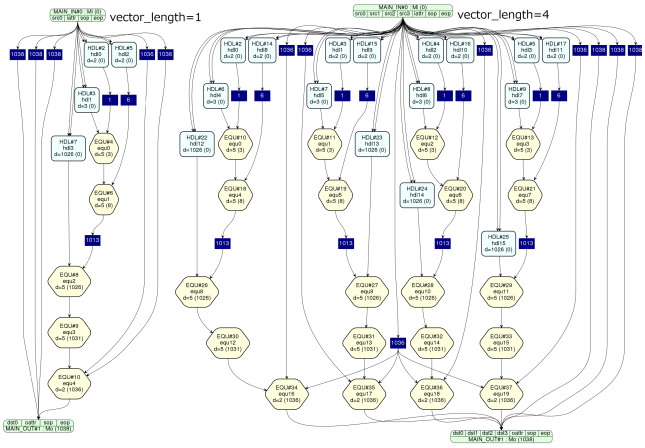

図 6 ベクトル化された FPGA カーネル

```
1  Name       kernel0;
2  Main_In    {Mi::src0,src1,src2,src3,\\
3              iattr,sop,eop};
4  Main_Out   {Mo::dst0,dst1,dst2,dst3,\\
5              oattr,sop,eop};
6  HDL hdl0, 2, (xxxv00)() = \\
7      mStreamBackward(src0,..,.pBwdCycles(1)>;
8  HDL hdl1, 2, (xxxv01)() = \\
9      mStreamBackward(src1,..,.pBwdCycles(1)>;
10 HDL hdl2, 2, (xxxv02)() = \\
11     mStreamBackward(src2,..,.pBwdCycles(1)>;
12 HDL hdl3, 2, (xxxv03)() = \\
13     mStreamBackward(src3,..,.pBwdCycles(1)>;
14       . . . <<SKIPPED>> . . .
15 EQU equ16, dst0 = mux(src0,conv240,iattr
       [0]);
16 EQU equ17, dst1 = mux(src1,conv241,iattr
       [1]);
17 EQU equ18, dst2 = mux(src2,conv242,iattr
       [2]);
18 EQU equ19, dst3 = mux(src3,conv243,iattr
       [3]);
19 DRCT (oattr,Mo::sop,Mo::eop) = \\
20       (iattr,Mi::sop,Mi::eop);
```

図 7 ベクトル化された FPGA カーネル

けるベクトル化はコードが持つ並列性を用いて複数のデータを一つのサイクルで処理するハードウェアを生成することで演算スループットを向上させる。ベクトル化された SPGen のストリーム計算コアを図 6 に示す (*vector_length* = 4)。入出力ポートと計算ロジックを複製することで同じクロックサイクルで高いスループットを達成するハードウェアを生成する。

図 7 に図 4 から生成されるベクトル化された SPD コードを示す。スループットを上げるため、入出力ポートと計算式が *vector_length* で与えられた数の分複製される。*iattr* はデータストリームの各要素に対して実行制御を行うための 32-bit 整数データである。多くのステンシル計算ではデータの全体領域と書き込みが行われる領域が異なる (e.g. 境界要素は参照のみを行う) ため、ビットマスクによる書き込み制御が必要である。データストリームの各要素に対して CPU 側で *iattr* のビットマスクの値を生成し、FPGA 側では値による書き込み制御を行う。ベクトル化を行う場合は同一 Vector Lane の中では一つの *iattr* を共有して使うように処理系が値を生成する。処理系はループ文の中のデータ依存の解析を行い、依存する反復間の距離を計算する。*vector_length* の値が計算された値を超えない限り、ベクトル化を安全に行うことができる。

5.2 Loop Unroll

CPU プログラミングにおける Loop Unroll はループ文の一部をインライン展開し、ループ制御のオーバーヘッドの軽減と並列性を高めることで性能最適化を行う。SPGen における Loop Unroll は対象ループ文が反復法で繰り返し実行されることを想定し、反復の一部をハードウェア化することで性能を最適化する。図 8 に Loop Unroll された FPGA カーネルを示す (*unroll_count* = 4)。対象 FPGA カーネルがループ文によって **N** 回実行されると仮定し、4 回文の反復を行うハードウェアを生成することでループ文の反復回数を **N**/4 回になる。これによって FPGA カーネルの実行に必要な DMA メモリ転送を 1/4 に減らすことや、4 回文の反復を計算することによってより深いパイプラインを持つハードウエアを合成することができる。

P 段のステージを持つパイプラインアーキテクチャで **M** 個の要素を処理する場合、(**P** + **M** - 1) サイクルが必要である。そのようなハードウェアを用いて反復法により **UC** 回計算を行うと (**P** + **M** - 1) × **UC** サイクルが必要である。Loop Unroll によって **UC** 回の反復を一つのパイプラインアーキテクチャとして合成すると、実行サイクル数は ((**P** × **UC**) + **M** - 1) になる。要素数 **M** を無限大に大きくできると仮定すると Loop Unroll による性能改善は **UC** 倍になる。

Loop Unroll を指定するためには図 4 のように *offload* 指示文に *unroll_count* 節を与える。処理系は対象ループ文を *unroll_count* 節で与えられた数の分だけ複製し、図 8 のようにカスケード接続を行う。また、対象ループ文の SPD コードを呼び出すコードを別途生成する。複製した数だけ外側のループ文の反復回数を減らさないといけないが、現在のプログラミングモデルではプログラマが外側のループ文を手動で変更することにしている。

6. 性能評価

本章では 2 次元ステンシル計算コードを用いて、合成されたハードウェアの消費リソースや性能の評価を行う。表 1 に評価環境を示す。図 4 で示したステンシル計算コードを用いて性能評価を行う。データパラメータは **M** と **N** が 1024、**TIME_STEP** が 10240 である。最適化パラメータである **VL** と **UC** に 1 から 16 までの値を与えてリソース

```
Name        UC4_kernel0;
Main_In     {Mi::src0, iattr, sop, eop};
Main_Out    {Mo::dst0, oattr, sop, eop};
HDL core0, 1038, (xxxt0, xxxt1, xxxt2, xxxt3) =\\
  kernel0(src0, iattr, Mi::sop, Mi::eop);
HDL core1, 1038, (xxxt4, xxxt5, xxxt6, xxxt7) =\\
  kernel0(xxxt0, xxxt1, xxxt2, xxxt3);
HDL core2, 1038, (xxxt8, xxxt9, xxxt10, xxxt11) =\\
  kernel0(xxxt4, xxxt5, xxxt6, xxxt7);
HDL core3, 1038, (dst0, oattr, Mo::sop, Mo::eop) =\\
  kernel0(xxxt8, xxxt9, xxxt10, xxxt11);
```

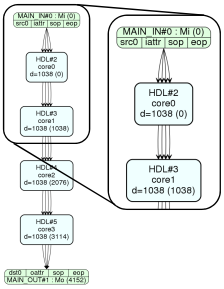

図 8 Loop Unroll による FPGA カーネルのカスケード接続

Item	Specification
FPGA	Intel Arria 10 GX FPGA
Memory	DDR3-2133 ×2
	17.067 GB/s ×2
Interface	PCIe Gen3 ×8 Edge
Compiler	Intel Quartus Prime 16.1

表 1 評価環境

VL	ALM	REG	BMEM	DSP
1	7.89 (0.21)	4.30 (0.05)	9.83 (0.37)	0.26 (0.26)
2	8.19 (0.40)	4.32 (0.09)	10.0 (0.61)	0.52 (0.52)
4	8.84 (0.79)	4.50 (0.17)	10.5 (1.10)	1.05 (1.05)
8	9.94 (1.49)	4.91 (0.32)	11.5 (2.07)	2.10 (2.10)
16	11.0 (2.98)	5.24 (0.66)	13.4 (4.01)	4.21 (4.21)

表 2 ベクトル化による消費リソース量の変化 (%)

消費量や性能の変化を評価した。性能評価は FPGA カーネルの実行サイクルをハードウェアカウンターによって測定することで行ったため、FPGA と CPU 間のメモリ転送などのオーバーヘッドは考慮していない。

6.1 消費リソースの評価

表 2 に **VL** を変化させてベクトル化を行ったときのハードウェアのリソース消費量を示す。表の値は FPGA チップのリソース量に対する消費リソース量の割合を表したものである。括弧の中の値は SPD コードによって生成されるストリーム計算コアの消費リソース量である。SPGen は SPD コードで記述されたロジックの他に、DMA 転送などを行う共有ロジックを生成するため、トータルの FPGA リソース消費は括弧の値より高くなる。

VL を増加させることで合成されるハードウェアのクロック当たりのスループットが増加する。しかし、FPGA のリソース量とメモリバンド幅の制限があるため、**VL** の値を無限に増やすことはできない。表 2 の結果から **VL** を増やすことでストリーム計算コアのリソース量が高くなることがわかる。しかし、評価環境のメモリバンド幅を十分に活用できると予測される 16 の場合でも計算コアの消費量が最大 4.01% (BMEM の場合) に収まっており、共有ロジックのリソース量も **VL** に大きく比例せず、9%程度 (BMEM の場合) を維持している。

UC を増加させ、モジュールをカスケード接続すること

UC	ALM	REG	BMEM	DSP
1	7.89 (0.21)	4.30 (0.05)	9.83 (0.37)	0.26 (0.26)
2	8.01 (0.42)	4.28 (0.10)	10.2 (0.74)	0.52 (0.52)
4	8.32 (0.82)	4.34 (0.20)	10.9 (1.48)	1.05 (1.05)
8	9.01 (1.64)	4.62 (0.41)	12.4 (2.97)	2.10 (2.10)
16	10.3 (3.25)	5.05 (0.81)	15.4 (5.94)	4.21 (4.21)

表 3 Loop Unroll による消費リソース量の変化 (%)

	1	2	4	8	16
fmax (MHz)	220.5	224.6	221.4	220.9	209.1
GFLOPS (fmax)	0.877	1.788	3.525	7.034	13.31
GFLOPS (200)	0.796	1.592	3.184	6.368	12.73

表 4 ベクトル化された FPGA カーネルの予想性能

	1	2	4	8	16
fmax (MHz)	220.5	220.2	240.2	220.7	212.8
GFLOPS (fmax)	0.877	1.751	3.814	6.979	13.35
GFLOPS (200)	0.796	1.590	3.174	6.324	12.55

表 5 Loop Unroll を適用した FPGA カーネルの予想性能

でより深いパイプラインを持つハードウェアを合成することができる。メモリバンド幅による制限があるベクトル化とは違って Loop Unroll は FPGA リソース量とデータサイズに制限される。データサイズが十分大きい場合、FPGA リソースを限界まで利用することで性能を最適化することができる。

表 3 に **UC** を変化させて Loop Unroll を行ったときのハードウェアのリソース消費量を示す。ベクトル化を行った表 2 と比べると Loop Unroll によるリソース消費量の変化がわずかに高いことがわかる。演算部分 (HDL、EQU 式) とポートのみを複製するのベクトル化に対して Loop Unroll はモジュール全体を複製し、カスケード接続を行うため、リソースの消費量が高くなる。しかし、**UC** を 16 にしてもトータルリソース消費量は最大 15.4% (BMEM の場合) であり、今回のステンシル計算のようなシンプルなコードではさらに **UC** の値を増やして性能を向上させる余地がある。

6.2 実行性能の評価

生成された FPGA カーネルの予測性能を表 4 と表 5 に示す。パイプライン実行によって予想される実行サイクル数と入出力の幅 (ベクトル幅) を考慮した性能予測を行ったものである。FPGA 処理系が生成した最大動作周波数と性能評価で用いた 200MHz の値を両方挙げている。表の結果からベクトル化と Loop Unroll によって最適化パラメータの変化に比例した性能向上が見られると予想する。

ベクトル化と Loop Unroll を組み合わせて最適化したハードウェアを用いてステンシル計算を評価した結果を表 6 に示す。括弧の中の値は **VL** と **UC** の値を 1 にしたハードウェアの性能を 1 とした場合の相対性能である。動作周

UC \ VL	1	8	16
1	0.7960 (1)	6.3248 (7.94)	11.127 (13.9)
8	6.3248 (7.94)	50.253 (63.1)	88.425 (111)
16	12.551 (15.7)	99.729 (125)	175.41 (220)

表 6 FPGA カーネルの評価結果 (GFLOPS)

波数は 200MHz に固定している。

ベクトル化による性能改善は表 4 に示した値より低い。合成したハードウェアでは単純に参照するデータの幅が増えるだけでなく、データストリームを遅延させ、隣接領域を参照するモジュールの利用が増えるため、追加のオーバーヘッドが発生する。これにより予測した性能より低い性能を達成しているものの、**VL** を増加させることで性能が向上することを確認した。

Loop Unroll による性能改善は表 5 に示した値とほぼ同じ結果を達成した。SPGen は命令レイテンシを静的に解析するため、パイプラインの段数を増やす Loop Unroll に対しては予測性能と近い性能向上が期待できるためである。データのサイズがパイプラインのオーバーヘッドを隠すほど十分大きいため、複製されたストリーム計算コアの数に比例した性能向上が得られている。

ベクトル化と Loop Unroll を組み合わせて、256 ストリーム計算コアを生成する最適化により (ベクトル化; 16、Loop Unroll: 16) 220 倍の性能向上率を達成している。性能評価の結果は C 言語の計算コードに数行の指示文を追加することで達成されたものである。性能パラメータの **VL** と **UC** はマクロで定義されているため、簡単に値を修正してハードウエアを合成し直すことができる。これによって C2SPD がストリーム計算に対して高い生産性と性能を同時に達成することを示した。

7. ループスケジュール最適化の課題

C2SPD の SPD コード生成に用いた Polly はループ文の反復空間やメモリアドレスへの参照を解析し、変形するためのフレームワークである。ループ文の反復空間に対する問題を整数計画法を用いて最適化することでベクトル化やスレッド並列化に適したスケジュールを生成することを主な目的とする。FPGA 向けの最適なループスケジュールを探すことは従来の CPU 向けの最適化とは異なる部分がある。CPU のようなランダムなメモリアクセスができないため、メモリ参照がサイクル毎に一定の間隔で変化するアクセスパターンになるようにループ文を変形しなければならない。そのような参照が複数存在する場合、ステンシル計算で用いるようなストリーム参照モジュールを利用する。

また、最適なストリーム計算のためにはメモリアクセスが逐次になるように Loop Interchange などのループ文の変形が必要である。今後の課題として 1 より大きいステップで反復を行うループ文の変形や、反復間の依存性が存在するループ文を変形し、並列化可能にするテクニック (Loop Skewing) の FPGA への適用について検討を行う。

現在のプログラミングモデルでは Loop Unroll のためのループ文の変形をプログラマが手動で行うようにしているが、Unroll 対象になるループ文も Polly の解析範囲に含むことで最適な変形を行うようなスケジュール方法の検討を行う。例えば時間反復のループ文を展開することにより最適な Systolic array アーキテクチャ [13] を生成することなどが考えられる。

8. おわりに

本稿ではストリーム計算向け FPGA 開発プラットフォーム SPGen の C 言語フロントエンド C2SPD の指示文を用いたプログラミングモデルを提案した。C2SPD の指示文は FPGA 化の対象となるループ文にベクトル化や Loop Unroll のようなストリーム計算向け最適化を記述することができる。ステンシル計算を用いた性能評価で最適化なしの性能に対して 256 個のストリーム計算コアを生成し、最適化したハードウェアが 220 倍の性能向上を達成した。

今後はループ文の変形やスケジュールによる性能最適化アルゴリズムを検討するとともに、それを簡潔に記述できるプログラミングモデルの設計を行う。また、アプリケーションによる性能評価から得られたフィードバックで SPGen の機能拡張を提案し、C2SPD で扱うことのできるコードの範囲を拡大することでさらなる生産性の向上を目指す。

参考文献

[1] SystemC: http://www.accellera.org/downloads/standards/systemc.
[2] Xilinx Vivado HLS: https://japan.xilinx.com/products/design-tools/vivado/integration/esl-design.html.
[3] CyberWorkBench: http://jpn.nec.com/cyberworkbench/index.html.
[4] OpenCL Overview: https://www.khronos.org/opencl.
[5] Intel FPGA SDK for OpenCL: https://www.altera.co.jp/products/design-software/embedded-software-developers/opencl/overview.html.
[6] Lee, S., Kim, J. and Vetter, J. S.: OpenACC to FPGA: A Framework for Directive-Based High-Performance Reconfigurable Computing, *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 544–554 (2016).
[7] Milutinovic, V., Salom, J., Trifunovic, N. and Giorgi, R.: *Guide to DataFlow Supercomputing: Basic Concepts, Case Studies, and a Detailed Example*, Springer Publishing Company, Incorporated (2015).
[8] Sano, K., Suzuki, H., Ito, R., Ueno, T. and Yamamoto, S.: Stream Processor Generator for HPC to Embedded Applications on FPGA-based System Platform, *CoRR*, Vol. abs/1408.5386 (online), available from ⟨http://arxiv.org/abs/1408.5386⟩ (2014).
[9] 航平長洲，健太郎佐野：FPGA によるデータフロー計算

機におけるハードウェア資源割当て最適化，技術報告 25，東北大学大学院情報科学研究科, 東北大学大学院情報科学研究科 (2018).
[10] 健太郎佐野，涼　伊藤，啓介菅原：階層的モジュール設計を可能とするストリーム計算コア高位合成コンパイラ (リコンフィギャラブルシステム)，電子情報通信学会技術研究報告 = IEICE technical report : 信学技報，Vol. 115, No. 109, pp. 159–164（オンライン），入手先 ⟨https://ci.nii.ac.jp/naid/40020523875/⟩ (2015).
[11] 珍泌　李，知洋上野，三久佐藤，健太郎佐野：FPGA 向けストリームプロセッサ生成のための C 言語フロントエンドの開発，技術報告 25，理化学研究所計算科学研究機構, 理化学研究所計算科学研究機構, 理化学研究所計算科学研究機構, 理化学研究所計算科学研究機構 (2018).
[12] GROSSER, T., GROESSLINGER, A. and LENGAUER, C.: POLLY - PERFORMING POLYHEDRAL OPTIMIZATIONS ON A LOW-LEVEL INTERMEDIATE REPRESENTATION, *Parallel Processing Letters*, Vol. 22, No. 04, p. 1250010 (online), DOI: 10.1142/S0129626412500107 (2012).
[13] Kung, H. T.: Why Systolic Architectures?, *Computer*, Vol. 15, No. 1, pp. 37–46 (online), DOI: 10.1109/MC.1982.1653825 (1982).