Vol.2018-HPC-167 No.12
2018/12/17

ストリーム計算ハードウェアコンパイラSPGenのための Polyhedral Modelを用いたループスケジュール最適化

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

概要：近年、HPC の分野で主流だったメニーコアアーキテクチャは半導体技術の停滞によってコア数の増加に限界が見えてきている。そのような流れの中で、プログラミング可能なハードウェアである FPGA を用いた専用ハードウェアの利用が注目を集めている。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、ポータブルな HPC 向けのプログラミングモデルはいずれも FPGA の利用における生産性と性能を両立できない問題があり、プログラミングモデルに関する研究が続いている。本研究はデータフローモデルによるストリーム計算に特化したハードウェアコンパイラ SPGen を FPGA のバックエンドにし、C 言語によるプログラミングを可能にするためのプログラミングモデルを提案するものである。著者らが以前提案を行った C2SPD は C 言語と独自の指示文を用いて SPGen のハードウェアを生成し、高いスループットを実現するハードウェアを生成することに成功した。しかし、言語モデルがステンシル計算に特化しており、一部は逐次コードからのコード変更が必要である問題がある、本稿では C2SPD の言語仕様の再設計を行い、より一般的に利用可能なプログラミングモデルの提案を行う。C2SPD はオープンソースコンパイラの LLVM をベースにしており、本稿で Polyhedral Model を用いる Polly によるループ文の解析やのコード変換のアルゴリズムを述べる。提案手法の実装や性能評価には至っておらず、今後の課題を原稿の最後に示す。

Polyhedral Copmilation of Loop Statements for Streaming Processing using SPGen

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

1. はじめに

近年、High Performance Computing (HPC) の分野ではチップ内の計算コア数を増やすことで性能を向上させるメニーコアアーキテクチャが主流である。しかし、半導体技術の停滞によって同じ面積に実装できる計算コアの数にも限界があると考えられ、今後異なるアプローチが求められると考えられる。そのような流れの中で、機械学習のような分野では専用ハードウェアを用いた計算の高速化が注目を集めている。特定のアプリケーションに特化し、従来の汎用プロセッサより限定した機能を持つため、同じ面積により多くの計算コアを実装でき、消費電力も抑えられる。従来、このような専用ハードウェアは Application Specific Integrated Circuit (ASIC) として実装されていたが、ハードウェアの集積度が上がるにつれて開発期間の増加やコストが膨大になるなどの問題がある。

専用ハードウェアを実現する手法として近年 Field-Programmable Gata Array (FPGA) が注目を集めている。FPGA は計算ロジックのハードウェアを持つ論理ブロックと再構築可能なネットワークで構成される。対象アプリケーションが行う計算を論理ブロックで実装し、ネットワークの配線を組み立てることで任意の専用ハードウェアを実現することができる。半導体の利用効率や消費電力は ASIC と比べて劣るものの、プログラミングによるアーキテクチャの変更が可能であるため、用途変更が容易で開発期間が短くなるというメリットがある。

[1] 理化学研究所 計算科学研究機構
RIKEN Advanced Institute for Computational Science
[a)] jinpil.lee@riken.jp

ⓒ 2018 Information Processing Society of Japan

本研究は HPC 分野における FPGA のプログラミングモデルに関するものである。FPGA のプログラミングモデルとして伝統的にハードウェア記述言語である Verilog HDL や VHDL などが用いられてきたが、低レベルのハードウェアに関する知識が必要であるため HPC のアプリケーションプログラマが使うには敷居が高い。著者らは HPC の分野で広く使われている C 言語から FPGA の回路を生成するプログラミングモデル C2SPD を提案し、処理系の実装を行った [1]。C2SPD は C 言語のループ文に指示文を挿入することでコード領域を限定し、ストリーム計算に特化した FPGA ハードウェアを生成する。

C2SPD は FPGA のバックエンドとして理化学研究所の佐野ら [2][3] が開発を行っている Stream Processor Generator (SPGen) を用いる。SPGen はストリーム計算に特化した高度にパイプライン化されたハードウェアを生成するが、計算の記述に Static Single Assignment (SSA) を用いる、CPU 側の制御に専用 API を用いるなど、プログラミングコストの面で改良の必要があった。C2SPD は SPGen 向けの C 言語フロントエンドを提供することでこの問題を解決している。しかし、C2SPD のプログラミングモデルにはいくつか改善が必要な部分がある。本研究では C2SPD のプログラミングモデルの再設計と Polly を用いたループの解析手法について考察を行った。

本稿の構成は次のようである。第 2 章では関連研究を挙げる。第 3 章ではストリーム計算に特化したハードウェアを生成するハードウェアコンパイラ SPGen の概要を述べる。第 4 章では Polyhedral Model を用いてコード最適化を行う LLVM の Polly プロジェクトについて説明する。第 5 章では既存の C2SPD の再設計を行い、改良したプログラミングモデルについて述べる。第 6 章では Polly を用いたループ文の解析と SPGen 向けコード生成について述べる。第 7 章では結論と今後の課題について述べる。

2. 関連研究

C 言語などの高水準言語を用いて FPGA をプログラミングするために多くのプログラミングモデルやツールが提案されている。Xilinx Vivado HLS[4] や SystemC[5]、NEC の CyberWorkBench[6] は FPGA が伝統的に広く使われてきた組み込み機器の開発向けに提案されたプログラミング環境である。C 言語やそれに準ずる水準の言語モデルによるプログラミングが可能であるが、HPC のために必要な大量の浮動小数点演算の処理を記述するには適していない。

OpenCL[7] は Graphic Processing Unit (GPU) やメニーコア CPU の並列処理を記述するために提案されたプログラミングモデルであり、近年 Intel と Xilinx による FPGA 向けの実装が提供されている [8]。OpenCL は CPU、GPU、FPGA などの対象デバイスに共通して利用可能なポータブルなプログラミングモデルを提供するが、並列処理向けに設計された言語モデルで FPGA 向けのパイプライン処理を記述することは困難であり、性能最適化のためにはベンダーが定義した独自の言語仕様を用いる必要がある。また、そのような仕様を用いた場合でも生成されるハードウェアを意識した高度な最適化が必要であるため、HPC のアプリケーションプログラマには敷居が高い。Oak Ridge National Lab の Lee ら [9] は指示文ベースのプログラミングモデル OpenACC[10] の FPGA 向け OpenCL のコンパイラ OpenARC を開発することで OpenCL の生産性を改善する研究を行っている。OpenARC が対象とする最適化が適用可能なコードに関しては最適化された OpenCL のコードが生成されるが、その他のコードに対しては OpenCL と同様の課題が残る。

```
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. SPGen の概要

本章では C2SPD の FPGA バックエンドとして用いられている SPGen の概要を述べる。SPGen は独自の Domain Specific Language (DSL) である Stream Processing Description (SPD) を用いてストリーム計算を記述する。ストリーム計算では入力データが 1 次元のストリームとして表現され、頭から順番に計算ハードウェアに流れる。ストリーム計算コアは入力ストリームから受け取ったデータを用いて計算を行う。サイクル毎に生成される計算結果が出力ストリームを生成する。

図 1 に SPD のコード例を示す。Name は生成される計算モジュールの名前である。Main_In は入力ストリームを構成するデータ変数をあらわす。図 1 では入力ストリームに *x1*、*x2*、*x3*、*x4* の 4 つの変数が含まれる。sop と eop はデータストリームの制御のために用いられる信号であり、計算には用いられない。同様に、Main_Out は出力ストリームを構成する変数をあらわす。EQU 式は計算を記述し、一つの EQU 式で一つの計算ノードを構成する。DRCT 式で入力と出力を直接接続させることができる。図 1 では制御信号を後ろのモジュールに伝達するために用いられている。

SPD コードは SPGen コンパイラによってハードウェア化される。図 2 に図 1 から生成されたハードウェアを示す。EQU 式の演算は FPGA の計算ロジックを用いるようにハードウェア化される。各演算はレイテンシを隠すためにパイプライン化される。レイテンシが異なる演算同士の

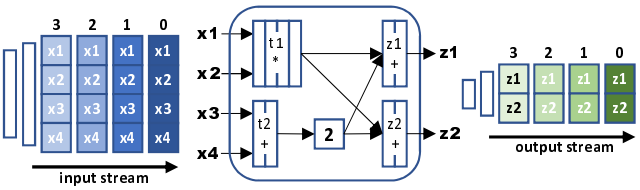

図 2　図 1 から生成されるストリーム演算コア

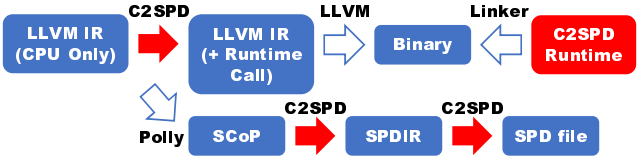

図 3　C2SPD のコード変換の流れ

結果を待ち合わせるために遅延バッファーが挿入される。このようなハードウェアを生成することでサイクル毎に計算結果を生成するハードウェアを実現し、汎用プロセッサと比べて高いスループットを達成することができる。

4. Polly の概要

本章では C2SPD がループ文の解析に用いる Polly[11] について説明を行う。Polly はオープンソースコンパイラの LLVM[12] のモジュールの一つとして開発されているプロジェクトである。LLVM の中間形式 (LLVM IR) を解析し、Polyhedral Model による解析と最適化を行うもので、ベクトル化や OpenMP 並列化、CUDA 向け GPU 並列化などを行う。

図 3 に C2SPD のコード変換の流れを示す。C2SPD はループ文を対象に SPD のコード生成を行うが、その解析と最適化に Polly を用いる。Polly は LLVM IR からループ文を検出すると独自の中間形式である SCoP を生成する。SCoP は以下の要素で構成される。

- **Statement** Polyhedral Model の解析単位である。LLVM では主に BasicBlock 単位の処理が行われる。
- **Instance** ループ文の各イテレーションで実行される statement をあらわす。異なるイテレーションの statement は異なる instance として扱われる。
- **Domain** instance の実行範囲をあらわす。ループ文のイテレーション範囲と同じである。
- **Access Relation** 各 instance と、それが参照 (read、write) するメモリの関係をあらわす。access relation によって配列要素単位の依存性の解析が可能である。
- **Dependency Relation** ループ文の中での instance 同士の依存関係をあわらす。access relation から計算される。Read-After-Write (RAW)、Write-After-Read (WAR)、Write-After-Write (WAW) の 3 種類の依存関係がある。その他に、データの有効範囲を解析するためにあるメモリセルが別の値に書き換えられることをあらわす flow dependency の計算も求めることが可能である。
- **Schedule** instance の実行順序をあらわす。Dependency Relation と他の制約条件などによって計算される。schedule の最適化によってベクトル化や OpenMP、GPU 並列化を行うことができる。

```
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　C2SPD による Laplace 方程式ソルバーのコード (従来)

ループ文のスケジュールは dependency relation を守り、メモリアクセスのレイテンシを最小にするなどの制約条件を満たすように行われる。任意のメモリアドレスを参照することが可能で、イテレーションの順序入れ替えも比較的自由に行える CPU コードとは違って、SPGen 向けのストリーム計算では新しい制約条件が必要である。C2SPD で導入される制約条件などについては第 6 章で述べる。

5. C2SPD のプログラミングモデル

本章では C2SPD のプログラミングモデルの再設計を行い、言語仕様について検討を行う。従来の C2SPD は C 言語のコードに独自の指示文を挿入することで SPGen のハードウェアを生成するコード領域を指定した。図 4 に C2SPD の指示文で記述された Laplace 方程式ソルバーのコード例を示す。SPGen のストリーム計算コアとしてハードウェア化されるコード領域を offload 指示文を用いて指定する。SPGen の性能最適化のためのテクニックであるベクトル化や loop unrolling を行うために vector_length や unroll_count 節が記述されている [1]。時間反復が進んだときに入出力を入れ替えるために、本来の逐次コードには別途のループ文による代入が行われるが、C2SPD コードでは connect 節で同様の処理を指定している。CPU と FPGA メモリ間のデータ転送は region 指示文の前後に行われる。region 指示文が時間反復のループに記述されることで不必要なメモリ転送を抑えることができる。

従来の言語仕様はステンシル計算に特化したものであり、connect 節を導入することで元の逐次コードを変更しなければならないという問題があった。新しい言語モデル案ではこのような問題点を解決し、ステンシル計算より一

```
1  float in[M][N];
2  float out[M][N];
3
4  #pragma spd region
5  for (int t = 0; t < TIME_STEP; t++) {
6  #pragma spd offload
7    for (int i = 1; i < M-1; i++) {
8      for (int j = 1; j < N-1; j++) {
9        out[i][j]
10         = (in[i][j-1] + in[i][j+1]
11         +  in[i-1][j] + in[i+1][j]) / 4.0f;
12     }
13   }
14
15 #pragma spd offload
16   for (int i = 0; i < M; i++) {
17     for (int j = 0; j < N; j++) {
18           in[i][j] = out[i][j];
19     }
20   }
21 }
```

図 5　C2SPD による Laplace 方程式ソルバーのコード (改良案)

般化した言語モデルになるように設計を行った。図 5 に改良案を示す。改良案でも C 言語のコードに指示文を挿入するというプログラミングモデルを採用する。

offload 指示文の意味は従来通りであり、対象ループ文を SPD コードに変更する。しかし、offload 指示文にあった vector_length や unroll_count、connect 節は廃止する。vector_length や unroll_count はコンパイル時の解析によって得られた結果をもとにコマンドライン入力として与えることにする。connect 節は廃止し、元の逐次コードをそのまま利用するアプローチに変更した。その結果、図 5 には図 4 では削除された入出力の入れ替え代入文がそのまま残る。

既存の C2SPD の言語モデルで region 指示文は CPU、FPGA 間のデータ転送のタイミングを指定するものであったが、新しい言語モデルでは意味を拡張する。region 指示文はいくつかの offload 指示文を含むコード領域に対して指定することができ、ループ文またはブロック文を対象とする。対象のループ文またはブロック文の本体には offload 指示文で指定されたループ文のみ記述できる。region 指示文は複数の offload 指示文から生成される SPD カーネルを実行するトップモジュールを生成する役割を果たし、offload 指示文から生成されるカーネル (offload カーネル) 同士を接続する。region 指示文の対象がループ文のとき、中の offload カーネルのカスケード接続による loop unrolling が行われる。したがって、図 4 のような既存の言語モデルで行われていた時間反復ループ文のイテレーションの手動変更は不要である。C2SPD のコンパイラは region 指示文から生成されるトップモジュールに対して CPU、FPGA メモリ間のデータ転送を行い、回路を実行するためのホスト側 API を挿入する。

6. Polly によるコード解析

本章では第 5 章で提案を行った言語モデルからループ文の解析とコード変換を行うコンパイル手法に関する検討を行う。そのために LLVM Polly が提供する Polyhedral Model を用いる。コード変換は Offload カーネルの生成とベクトル化、トップモジュールの生成と loop unrolling の順に行われる。

6.1 Offload カーネルの生成とベクトル化

C2SPD コンパイラは各 offload カーネルの対象ループ文の解析を行う。まず最初にターゲットループ文が SPGen のストリーム計算コアのハードウェアとして実現できるかテストを行う。SPGen の高いスループットを実現するためにはいくつかの制約があり、また現在の SPGen の機能制限からも C 言語で記述できるプログラムに制約が発生する。以下に対象ループ文の制約条件とテスト手法を示す。

- **依存関係の制約** ストリーム計算では一度計算が終了するとその結果が出力ストリームとしてハードウェアの外部 (主に次の演算コアの入力) に移動するので、書き込まれたメモリセルの値を再度変更することはできない。SCoP の WAW dependency が空でない場合はこのような依存関係が存在するため、offload カーネルを生成することはできない。
- **メモリ参照 offset の制約** 図 5 のようなステンシル計算では配列の隣接要素に対するメモリ参照が行われる。SPGen は高いスループットを達成するために各演算のレイテンシは定数で表現できることを前提としている (でない場合ストールが発生する可能性があるため)。したがって、メモリ参照の offset (イテレーションの変数からの差分) も定数で表現されるものでないといけない。SCoP の Access Relation の解析によって制約をテストする。
- **その他の制約** 現在の SPGen ではサポートされない機能に関連した制約が含まれる。例えば同じ配列が入力と出力ストリームに同時に含まれることはできない (HPC のリダクション演算などであらわれるアクセスパターン)。SCoP の Access Relation の解析によってテストする。

入出力ストリームを生成するために、SCoP の情報から live-in、live-out 集合を計算する。live-in とは対象ループ文の最初のイテレーションの前に値が定義されているメモリセルの集合である。Polyhedral Model の flow dependency の計算の中で、source を持たないメモリセルの集合が live-in に該当する。C2SPD は CUDA 向け GPU コードを生成する PPCG と同様のアルゴリズムで live-in 集合を計算する [13][14]。live-in 集合は入力ストリームの生成に用いられる。

ⓒ 2018 Information Processing Society of Japan

live-out とは対象ループ文の最後のイテレーションで有効である書き込みの集合である。メモリセルの書き込み同士の依存関係を解析することで最後のイテレーションで有効なメモリセルの集合を解析できる。live-in と同様、PPCG と同様のアルゴリズムで live-out 集合の計算を行う。live-out 集合は出力ストリームの生成に用いられる。

図 4 のコードから offload カーネルを生成する場合を考える。二つの offload カーネルの live-in と live-out 集合は以下のようになる[*1]。生成された live-in、live-out 集合を用いて offload カーネルの入出力ポートを生成する。ループ文の本体を用いた EQU 式の生成は [1] と同様である。

- **1st offload kernel** live-in → in[0:M:1][0:N:1]、live-out → out[1:M-1:1][1:N-1:1]
- **2nd offload kernel** live-in → out[0:M:1][0:N:1]、live-out → in[0:M:1][0:N:1]

offload カーネルが生成されると、コンパイラは対象カーネルがベクトル化できるか依存関係のテストを行う。RAW dependency と WAR dependency の依存関係の距離を計算することで可能になるが、その最小値が安全にベクトル化できる並列度になる。例えば、図 5 の out を in に変更した場合 (ガウス-ザイデル法や SOR 法にあらわれるアクセスパターン)、依存距離の最小値は 1 になり、ベクトル化は不可能である。並列度は他の offload カーネルの並列度にも影響される。現在の言語モデルではすべてのデータパスの幅を統一するためにベクトル化可能な幅を region 指示文の中にあらわれる offload カーネルの並列度の最小値とし、適切な値をプログラマに選ばせることにする[*2]。

現時点では Polly のスケジュールアルゴリズムを用いたループスケジュールを用いていない。任意のメモリアクセスが可能であることを前提とする CPU や GPU 向けのスケジュールアルゴリズムが SPGen では有効ではないためである。例えば、ループ文の skewing を行うことで依存関係を解消し、ベクトル並列化を可能にすることが CPU 向けのコード生成では可能であるが、メモリ参照の offset がイテレーション毎に動的に変わるため、SPGen 向けハードウェアを生成することは困難である。

6.2 トップモジュールの生成と loop unrolling

offload カーネルの live-in、live-out 集合を用いてトップモジュールの生成を行う。トップモジュールの役割は offload カーネルが逐次コードであらわれる順番と同様の順序で実行されるように接続を行うことである。offload カーネルと同様、トップモジュールでもストールしないストリーム計算を実現するためにいくつかの簡略化が行われる。以下にトップモジュールの生成プロセスについて述べる

- **入出力ストリームのサイズを均一化** ストールしないハードウェアを生成するためにはトップモジュールの中のすべての入出力ストリームの要素数が同じでないといけない (要素数が異なる場合は待ち合わせのための遅延が発生するため)。現在のモデルでは計算が行われた live-in、live-out 集合の各次元に対して最大値を求めて入出力ストリームの形状を決める。その結果余分な要素を持つストリームが出てしまうが、attribute 要素を用いたアクセス制御を行うことで逐次コードと同じ結果を生成することが可能である [1]。
- **ベクトル化パラメータの決定** offload カーネルの生成でも述べたように、データパスの幅を統一するため、ベクトル幅は各 offload カーネルの持つ値の最小値でなければならない。
- **HDL 式の生成** 入出力ストリームの形状とデータパスの幅 (ベクトル幅) が決定したら offload カーネルを呼び出す HDL 式を生成する。呼び出しは逐次のソースコードの順番通りに行われる。

region 指示文の対象がループ文である場合は loop unrolling を適用することができる。CPU の loop unrolling はいくつかのイテレーションを複製して展開する手法であるが、SPGen では offload カーネルのカスケード接続によって実現する。イテレーション間の接続の時に live-in、live-out の重複チェックを行う。live-out 集合の要素が live-in 集合に含まれている場合、計算結果が次のイテレーションで用いられるということを意味する。offload カーネルの出力ポートを次のイテレーションの offload カーネルの入力ポートに接続することで逐次コードと同じ結果が得られるようにコード生成を行う。

6.3 ホスト側 API の生成

region 指示文に対するホスト側 API の生成は既存の C2SPD と同様であるが、live-in、live-out 集合を用いることで配列要素単位のデータ転送が可能である。また、従来 connect 節で行われていた FPGA メモリ上での出力ストリームの入力ストリームへのフィードバック (region 指示文がループ文を指定している場合) も live-in、live-out 集合の重複チェックによってコンパイラが自動的に行う。

7. 結論と今後の課題

本研究では以前提案を行った C2SPD の言語仕様の再設計を行い、ステンシル計算より一般的なアプリケーションに対応したプログラミングモデルの提案を行った。また、提案したプログラミングモデルが LLVM の Polly を用いて解析され、SPD のコードに変換される手法を示した。まだ、提案手法の実装や性能評価には至っておらず、今後以下に示す課題を解決するとともにコンパイラの実装を進

[*1] 配列の各次元に記述された三つ組の整数は最初の要素の index、要素数、要素間の step をあらわす。
[*2] この選択を自動化することも可能であると考えられるが、現在のモデルでは FPGA のメモリバンド幅などのスペックを考慮し、プログラマが明示的に選べるようにする。

ⓒ 2018 Information Processing Society of Japan

める。

- 依存関係の解消やベクトル化を可能にする SPGen 向けループスケジュールのアルゴリズムの設計
- 対象アプリケーション拡大のための SPGen 機能拡張の提案
- offload カーネルの並列接続による性能最適化
- ベクトル化や loop unroll のパラメータの自動チューニング

参考文献

[1] Lee, J., Ueno, T., Sato, M. and Sano, K.: High-productivity Programming and Optimization Framework for Stream Processing on FPGA, *Proceedings of the 9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*, HEART 2018, New York, NY, USA, ACM, pp. 5:1–5:6 (online), DOI: 10.1145/3241793.3241798 (2018).

[2] 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).

[3] 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).

[4] Xilinx Vivado HLS: https://japan.xilinx.com/products/design-tools/vivado/integration/esl-design.html.

[5] SystemC: http://www.accellera.org/downloads/standards/systemc.

[6] CyberWorkBench: http://jpn.nec.com/cyberworkbench/index.html.

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

[8] Intel FPGA SDK for OpenCL: https://www.altera.co.jp/products/design-software/embedded-software-developers/opencl/overview.html.

[9] 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).

[10] OpenACC Specification: https://www.openacc.org/specification.

[11] 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).

[12] LLVM Compiler Infrastructure: https://llvm.org.

[13] Verdoolaege, S., Carlos Juega, J., Cohen, A., Ignacio Gómez, J., Tenllado, C. and Catthoor, F.: Polyhedral Parallel Code Generation for CUDA, *ACM Trans. Archit. Code Optim.*, Vol. 9, No. 4, pp. 54:1–54:23 (online), DOI: 10.1145/2400682.2400713 (2013).

[14] Verdoolaege, S. and Janssens, G.: Scheduling for PPCG (2017).

ⓒ 2018 Information Processing Society of Japan