Vol.2017-HPC-158 No.16
2017/3/9

OpenCLとVerilog HDLの混合記述によるFPGAプログラミング

藤田 典久[1,a)]　大畠 佑真[2]　小林 諒平[1]　山口 佳樹[3]　朴 泰祐[1]

概要：我々はGPUに代表されるアクセラレータを持つPCクラスタにおいて，アクセラレータ同士のノード間通信性能を向上させる機構として，TCA (Tightly Coupled Accelerators) と呼ばれるコンセプトを提案してきた．また，そのプロトタイプ実装をFPGA (Field Programmable Gate Array) を用いて行うことにより，演算加速と通信の融合だけでなく，アプリケーションに特化した演算機能を通信機構内に組み込むという新コンセプトとしてAccelerator in Switchを提唱している．近年，FPGAのハードウェアおよび開発環境の進歩により，より一般的な環境でAccelerator in Switchを実現できるようになってきた．ハードウェア面では40Gb/100Gb Ethernetのような高速な外部リンクが搭載され，またFPGA開発に用いられる言語としてC言語，C++言語，OpenCL言語などを利用可能な，高位合成と呼ばれる手法が広まりつつある．これらの背景の下，Accelerator in Switchをアプリケーションユーザにまで広める環境が固まりつつある．本稿では，Accelerator in Switchにおいて，OpenCLでは記述できない機能を補完するためにVerilog HDL記述を平行して用い，OpenCLとVerilog HDLを併用してプログラミングする方法について検討を行う．通信機構などの外部ペリフェラルとOpenCLを接続する方法の検討や，メモリアクセスやコアとなる演算をVerilog HDLで代替し，ライブラリすることで，より高性能・高効率な回路実装を目指す．一例として，内積計算をライブラリ化したところ，混合記述を行ったプログラムで理論ピーク性能の約90%の実効性能を達成し，OpenCLのみで記述したプログラムの性能を上回った．また，外部ペリフェラルの操作として，ボード上に搭載されているハードウェアの制御をOpenCLから行えることを確認した．

1. はじめに

筑波大学 計算科学研究センターでは，TCA (Tightly Coupled Accelerators) と呼ばれるコンセプトを提唱している．TCAはGPUなどのアクセラレータ間を通信ネットワークで密に接続し，低レイテンシで通信を行うというものである．また，NVIDIA GPU (Graphics Processing Unit) 向けのTCA実装としてPEACH2 (PCI Express Adaptive Communication Hub Ver.2) [1] という通信機構を開発しており，PEACH2を搭載したシステムとしてHA-PACS/TCA (Highly Accelerated Parallel Advanced System for Computational Sciences/TCA) [2] を運用している．

PEACH2はFPGA (Field Programmable Gate Array) を用いて実装されており回路の動作を変更できるため，アプリケーションに特化した演算機能を追加 [3], [4] でき，通信機構内に演算機構を組込む手法を，我々はAccelerator in Switchと呼んでいる．しかし，PEACH2はVerilog HDL (Hardware Description Language) のみで実装されているため，Accelerator in Switchとして演算機能を追加する場合，その開発コストが大きく，FPGAの専門家でなければその開発ができない問題があった．

近年，FPGAのハードウェアおよび開発環境の進歩により，より高度かつ柔軟なソフトウェア環境でFPGAプログラミングが可能になってきた．ハードウェア面では40Gb/100Gb Ethernetのような高速な外部リンクが搭載され，また，ソフトウェア面ではソフトウェア開発で用いられる言語 (C言語，C++言語，OpenCL言語など) から，FPGAの回路を生成する高位合成と呼ばれる手法が利用可能になってきており，より低コストにFPGAプログラミングが行える環境が整いつつある [5], [6]．しかしながら，RTL (Register Transferring Level) で手動設計された回路と比較して，性能が大きく悪化する [7] 場合や，FPGA内のリソースの利用量が増える [8] 場合があるなど，FPGA開発の全ての領域で高位合成がRTLを代替できる訳ではない．

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

本研究の目的は，Intel FPGA を対象に，高位合成言語の一つである OpenCL を用いて，高位合成で Accelerator in Switch の開発が可能かどうか検討することである．これにより，アプリケーションに直結する部分をユーザが OpenCL によって記述し，低レベルのコア演算やハードウェアを直接駆動する部分を OpenCL から呼び出し可能な Verilog HDL ライブラリとして提供し，Accelerator in Switch の概念を高性能並列計算向けに実用化することを目指す．本稿では， OpenCL では記述できない機能を補完するために Verilog HDL 記述も併せて用い，FPGA をプログラミングする方法について検討を行う．通信機構などの外部ペリフェラルと OpenCL を接続する方法の検討や，メモリアクセスや演算を Verilog HDL で代替することで，より性能の良い回路実装を行う．

2. FPGA の開発手法

本章では，FPGA 内の回路を開発する手法について述べる．FPGA の回路を作成する際の手法には，大きく分けて 2 つの手法がある．1 つは，ハードウェア記述言語 HDL (Hardware Description Language) と呼ばれる言語を用いて記述する方法，もう 1 つは，高位合成言語を用いて記述する方法である．高位合成言語は，一般的なソフトウェア開発に用いられる言語を用いて回路開発を行うものである．高位合成言語を用いて記述された回路は HDL に変換され，FPGA ベンダーが提供する論理合成ツールを用いて，変換された HDL から回路イメージを生成する．本稿では，HDL として Verilog HDL を，高位合成言語として OpenCL を用いて FPGA の回路を作成する．

2.1 Intel Quartus Prime FPGA 開発環境

Intel 社は Intel FPGA 向け総合開発環境として Quartus Prime と呼ばれるソフトウェアを開発している．Quartus Prime は Verilog HDL コンパイラだけではなく，FPGA の開発に必要なツールが複数含まれる総合開発環境である．OpenCL から FPGA 回路を生成する Intel FPGA SDK for OpenCL 開発環境も Quartus Prime に含まれている．

Quartus Prime 開発環境の機能のひとつとして Qsys と呼ばれるツールがある．Qsys は複雑な配線を持つ回路を図 1 のような GUI を通して設計できるツールである．Qsys システムは複数の Qsys モジュールおよびモジュール間の配線指定から構成され，どのモジュール間を接続するかの指定が GUI で直感的に指定できる．Qsys システムは最終的に Verilog HDL に変換され回路に合成される．

Qsys で取り扱う配線には次の 3 つの種類がある．1 つ目は Qsys 上で特別なプロトコルのない配線であり，たとえばクロック・リセット信号や，DDR メモリ用配線などの Qsys システムの外部との配線がこれに相当する．2 つ目は Avalon-MM (Avalon Memory Mapped) 配線である．

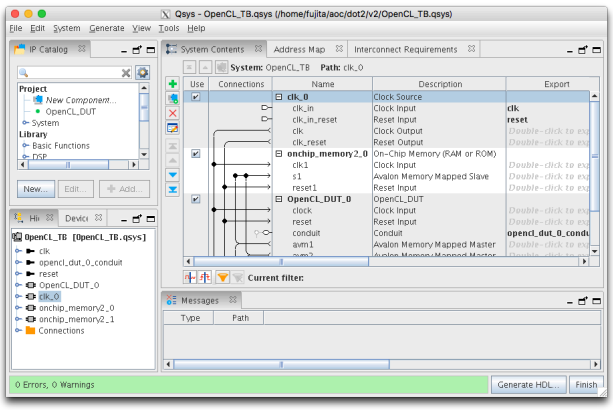

図 1: Qsys の編集画面.

Avalon-MM はメモリバス用のプロトコルであり，アドレスを指定して書き込み・読み込みができる．マスタとスレーブの区別があり，1 つのバスに複数のマスタとスレーブを配置してメモリ空間を共有できる．3 つ目は Avalon-ST (Avalon STream) 配線である．Avalon-ST は point-to-point のデータ通信用のプロトコルであり，高効率なデータ転送を行える．ただし，Avalon-MM とは異なり 1 対 1 の固定配線であるため，通信相手を動的に変化させることはできない．

Qsys は，Intel FPGA やサードパーティのベンダから提供される IP (Intellectual Property) コアをベースとしたシステムを構築する際に，標準的に利用される開発ツールである．設計者は提供されている IP コアを Qsys 上で Qsys モジュールとして作成し，ユーザの回路と接続することで利用する．Qsys モジュールとして提供されている IP コアの例としては，DDR メモリコントローラ，PCI Express コントローラ，NIOS II 組み込み CPU などがある．また，ユーザが自作した Verilog HDL のモジュールを Qsys モジュール化することもでき，その場合でも Qsys の GUI 画面を通じて配線を設計できる．

2.2 Intel FPGA SDK for OpenCL

Intel 社は OpenCL を用いて FPGA 回路を設計できる Intel FPGA SDK for OpenCL というパッケージを提供しており，その SDK を用いることで，OpenCL で記述したプログラムを FPGA 回路にコンパイルできる．Intel OpenCL は all-in-one 型の開発環境であり，OpenCL を用いて生成された回路は，それだけで FPGA を動作させることができ，HDL の知識が全く無くとも FPGA を利用できる．また，ホストからの FPGA 制御については，OpenCL 標準の API 群を用いて行え，ホスト OS から FPGA デバイスを操作するための Windows および Linux 用のドライバが提供される．

Intel FPGA SDK for OpenCL を用いて FPGA を開発する際の開発手順を図 2 に示す．OpenCL のコードは aoc コマンドを用いてコンパイルを行う (図 2 の (1) 部分)．.cl

ファイルを aoc コマンドにてコンパイルすると aocx ファイルが生成され，aocx ファイル内に FPGA の回路データが含まれる．

ホスト側で動作するコードは通常の OpenCL ソフトウェア開発と同じく，ホスト用のコンパイラを利用してコンパイルを行い，Intel FPGA SDK for OpenCL 用のライブラリをリンクする (図 2 の (2) 部分)．ただし，図 2 では，Linux 上で開発することを想定しホストコンパイラに gcc を用いているが，Windows 環境で開発を行う場合は Visual Studio の C/C++言語コンパイラを利用する．

OpenCL では，ソースコード (.cl ファイル) を実行時に読み込みコンパイルする方式 (オンラインコンパイル) と，事前にソースコードをオブジェクトファイルに変換しておき，そのファイルを実行時に読み込む方式 (オフラインコンパイル) の 2 つの利用方法があるが，OpenCL コードを FPGA の回路データにコンパイルする作業は時間がかかるため，Intel FPGA SDK for OpenCL ではオフラインコンパイル方式のみ利用できる．

OpenCL 利用時に FPGA の書き換えを行う方法は，aocx ファイルをホストプログラムの実行時に読み込み (図 2 の (3) 部分)，aocx ファイルのデータを引数にして `clCreateProgramWithBinary` 関数を呼び出すことで行う．FPGA の書き換え，メモリ管理，カーネル実行管理などの FPGA の制御全般は OpenCL の API のみで完結して行え，FPGA 固有の API やツールを利用する必要はない．

Intel FPGA SDK for OpenCL 開発環境固有の要素として，Board Support Package (BSP) がある．FPGA の場合，CPU 等とは異なり，FPGA チップのメモリ，IO などの周辺回路の構成はボード毎に異なる．ボード間の差異を吸収するために，ボード固有のパラメータや回路は BSP という形で提供され，OpenCL コードのコンパイル時に BSP を読み込み利用する．一般的に，OpenCL 対応の FPGA ボードを利用する場合，ボードの開発元から BSP が提供され，ユーザはその BSP を利用して OpenCL を用いた回路開発を行う．

BSP の構築には，前小節で述べた Qsys を利用しており，aoc コマンドによって OpenCL コードが Qsys システムに変換される．したがって，BSP は Qsys システムのテンプレートのようなものと見做すこともできる．

3. OpenCL+Verilog HDL 混合記述

本章では，OpenCL と Verilog HDL を併用して FPGA プログラミングを行う方法について述べる．Intel FPGA SDK for OpenCL で OpenCL と Verilog HDL の混合記述を行う場合には，「ヘルパー関数方式」と「I/O Channel」方式の 2 つの方式があり，以下の小節で，それぞれの方式について述べる．

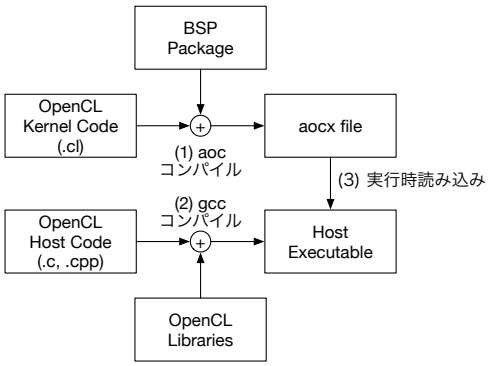

図 2: Intel FPGA SDK for OpenCL を用いて FPGA を開発する際の流れ．

3.1 ヘルパー関数方式

ヘルパー関数方式は OpenCL のコード内で関数を呼び出す形式で利用する方式である．OpenCL コード中では関数を呼び出す記述を行い，その関数の実装は OpenCL ではなく Verilog HDL で記述する．Verilog HDL で記述された関数の呼び出しをコンパイラが検出し，その部分に Verilog HDL モジュールが接続され，モジュールは OpenCL が生成するパイプラインの中に組込まれ動作する．

aoc コマンドによるコンパイルを行う際に，XML ファイルに必要なパラメータを記述しコンパイラに渡すことで，ヘルパー関数方式の実装が Verilog HDL で記述するとコンパイラに指示できる．設定用の XML ファイルには，引数の個数・データ幅，利用する Verilog HDL ファイルのパス，Verilog HDL で実装されるパイプラインのレイテンシは何サイクルなのか，パイプラインストールの有無などの情報を記述する．

後述する I/O Channel 方式と比べると，実装に必要な記述が少なく簡便な方式であるものの，OpenCL が生成するパイプライン中に組込まれるため，Verilog HDL モジュール内で実装できる機能に制約がある．Verilog HDL 内で利用できる入出力手段に制限があり，引数，返り値，メモリアクセスによる入出力しか行えず，例えば，ボード上の LED を操作したり，ネットワーク通信を行うなどの外部 I/O 機能を実装したりできない．したがって，ヘルパー関数方式の主な利用方法は，演算やメモリアクセスを行う機能の実装となる．

3.2 I/O Channel 方式

Intel FPGA SDK for OpenCL による OpenCL 言語への独自拡張のひとつとして Channel 機能があり，複数の OpenCL カーネルが同時に動いている場合にカーネル間通信を行える機能である．外部メモリを経由することなく，FPGA 内部でカーネル間通信が完了するため，高速なカーネル間通信が可能となる．また，Channel 機能には I/O Channel と呼ばれる動作モードがある．通常の Channel は

送信側と受信側どちらもも OpenCL によって実装されたカーネルであるが，I/O Channel は送信側もしくは受信側の一方が Verilog HDL によって実装された Qsys モジュールとなり，通信プロトコルは Avalon-ST を用いる．

ヘルパー関数方式は OpenCL が生成するパイプラインの中に組み込まれるのに対して，外部モジュールは，OpenCL パイプラインの外の Qsys ネットワークに接続され，独立して動作できる利点がある．I/O Channel の接続は BSP によって定義されており，I/O Channel を用いて何を実現可能かはボード毎に異なる．I/O Channel の拡張や動作変更を行うには BSP の修正が必要であり，ヘルパー関数方式よりもコストがかかる．しかしながら，I/O Channel 方式ではヘルパー関数方式とは異なり，例えば Ethernet 通信の制御を OpenCL で処理を行うといった外部のペリフェラルとの接続が可能となる．

4. I/O Channel を用いた外部ペリフェラル操作

本章では，I/O Channel を用いて OpenCL プログラムから外部ペリフェラルを操作する方法について述べる．ペリフェラル操作の実験として，FPGA 開発用ボードに搭載されている LED を I/O Channel を経由して OpenCL から操作する．

4.1 BSP の変更と LED の接続

I/O Channel を用いた外部ペリフェラル操作を行うには，外部ペリフェラルを操作するコードだけでなく，その機能を OpenCL から使えるように BSP を適切に設定する必要がある．

最初に，BSP の Qsys システムを編集し，LED と OpenCL の橋渡しを行うモジュールを追加する．このモジュールが必要な理由は，OpenCL から I/O Channel を通じてデータを受け取る際のプロトコルは Avalon-ST であるが，Avalon-ST の信号をそのまま LED 接続することはできないため，プロトコルを変換する回路が必要となるからである．プロトコル変換の役割を持つモジュールが図 3 の led_st_0 モジュールである．led_st_0 モジュールでは，Avalon-ST 経由で OpenCL から受け取ったデータをレジスタに保存し，レジスタの値をボード上の LED に出力する．

次に，OpenCL の I/O Channel と Avalon-ST モジュールの間の関連付を行うために，BSP に含まれている board_spec.xml ファイルを編集する．board_spec.xml ファイルは OpenCL から扱う FPGA ボード上のリソースに関する情報が記述されているファイルである．

board_spec.xml ファイルへの追加記述を図 4 に示す．interface タグの name 属性で Qsys モジュールの名前，port 属性でそのモジュールのどのポートに接続するか，chan_id で OpenCL 上での I/O Channel の名前，type 属性と width

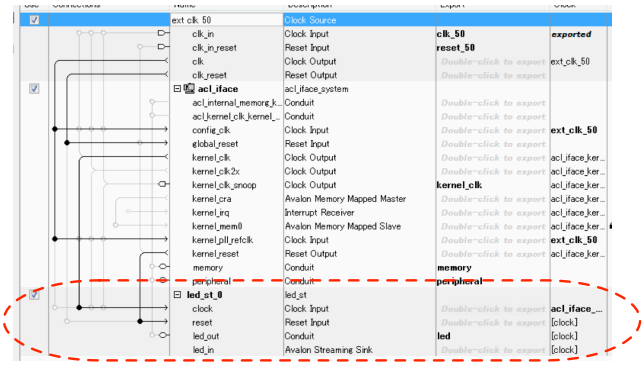

図 3: 赤枠が LED を OpenCL から操作するために追加した Qsys モジュール．

```
<channels>
  <interface name="led_st_0" port="led_in"
    type="streamsink" width="32" chan_id="led0" />
</channels>
```

図 4: board_spec.xml に追加した記述．

属性で信号の入出力方向とビット幅を指定する．図 4 の様に記述すると，Qsys に追加した led_st_0 モジュールの led_in ポートが OpenCL 上から “led0” という名前の I/O Channel として利用できる．

4.2 LED を操作する OpenCL コード

本小節では，前小節の手順を踏まえて OpenCL から LED を操作するプログラムについて述べる．I/O Channel を用いて LED を制御する OpenCL プログラムを図 5 に示す．1 行目の pragma は Intel FPGA SDK for OpenCL の独自拡張である channel の有効化をコンパイラに指示するためのものであり，2 行目で I/O Channel 変数 `outLED` を定義し，9 行目で `outLED` channel に対してデータを書き込む．`write_channel_altera` 関数は第一引数の Channel に第二引数のデータを書き込む組み込み関数である．

2 行目で宣言している channel 変数 `outLED` の接続先は io 属性で指定する．図 5 では，4 行目に `io("led0")` という属性を記述しており，これは，前述した board_spec.xml に追加した interface タグの chan_id 属性の値と対応している．

4.3 実験環境と結果

LED 操作の実験には Terasic 社が販売している DE1-SoC Board [9] を利用する．DE1-SoC ボードは FPGA 開発用のボードであり，OpenCL をサポートするボードである．FPGA として Cyclone V SoC チップを搭載しており，このチップは FPGA だけでなく，ARM Cortex-A9 プロセッサを Hard Processor System (HPS) として内蔵している．ただし，ARM プロセッサは FPGA のプログラマブルな

```
1  #pragma OPENCL EXTENSION cl_altera_channels :
       enable
2  channel uint outLED
3    __attribute__((depth(0)))
4    __attribute__((io("led0")));
5
6  __kernel
7  __attribute((reqd_work_group_size(1,1,1)))
8  void led(int N) {
9    write_channel_altera(outLED, N);
10 }
```

図 5: LED を操作する OpenCL コード.

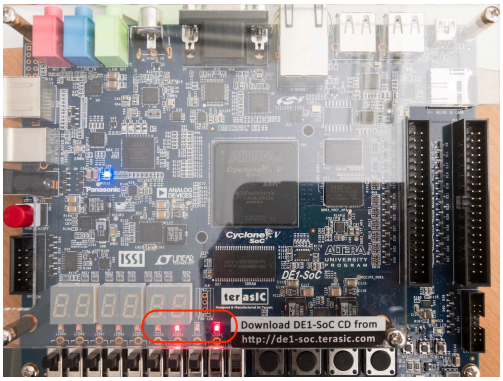

図 6: DE1-SoC ボードにて，LED の制御を OpenCL から行うプログラムの動作例．図中左下の赤枠で囲われた部分にある LED を制御する．

ロジックを利用しているのではなく，固定機能回路として実装されており，ユーザのカスタマイズなどはできない．Cyclone V SoC チップのような，OpenCL をサポートし，プロセッサと FPGA が同一チップ上にある SoC システムで OpenCL を利用する場合は，OpenCL の制御は内蔵プロセッサ上で動いている Linux OS から行う．

LED 操作の実験結果を図 6 に示す．図 6 の左下の赤枠内部の 4 個の赤色 LED を OpenCL プログラムから制御している．図 6 の写真は，I/O Channel に 5 (=2 進数で 0101) の値を書き込んだ場合のものであり，I/O Channel に書き込んだ値が LED 表示に正しく反映されていることがわかる．

5. ヘルパー関数方式を用いた内積計算の最適化

内積計算はメモリバンド幅ボトルネックな計算であり，限られたメモリバンド幅を効率良く利用することが高性能を達成する上で重要となる．本章では，計算およびメモリアクセスを両方 OpenCL で実装したものと，計算は OpenCL で行うがメモリアクセスを行う部分を Verilog HDL で実装したものについて性能評価を行う．

表 1: DE5-Net FPGA Development Kit の性能諸元.

FPGA	Intel Stratix V GX (5SGXEA7N2F45C2)
LE (Logic Element)	622K
ALMs (Adaptive Logic Module)	234,720
Registers	938,880
DSP (Digital Signal Processor)	256
M20K memory blocks	2,640
M20K memory size	50M bit
Memory	DDR3 SO-DIMM, 2GiB, 800MHz x 2ch
PCI Express	Gen3, x8 lanes

5.1 実験環境

ヘルパー関数方式を用いた内積計算の最適化に関する実験には，Terasic 社が販売している DE5-Net FPGA Development Kit [10] を使用する．表 1 に DE5-Net ボードのスペックを示す．DE5-Net は，FPGA として Stratix V チップを搭載しており，またメモリとして 800MHz 駆動の DDR3 メモリが 64bit 幅 ×2 チャンネル搭載されており，DDR3 メモリの理論ピークバンド幅は 25.6GB/s となる．

DE5-Net は PCI Express のボードとして設計されており，図 7 の写真にあるようにホストのシステムとは PCI Express を経由して接続される．前章で利用した DE1-SoC ボード搭載の FPGA とは違い，DE5-Net の FPGA には汎用プロセッサコアは含まれておらず，制御などは全てホスト OS から PCI Express を経由して行い，OpenCL を GPU で利用する際と似た制御モデルとなる．

DE5-Net を接続しているホストマシンには，CPU として Intel Xeon E5-2670 v3 を搭載したマシンを使用し，OS には CentOS 6.8 を用いる．また，開発用ソフトウェア環境は Intel Quartus Prime Standard Edition, Version 16.0.2 Build 222 を使用し，Intel FPGA SDK for OpenCL も同一のバージョンのものを利用する．

5.2 OpenCL 利用時のメモリ構成

複数のメモリチャンネルが利用できるボードでは，マルチチャンネルの動作についてコンパイル時にオプションを付与することで切り替えられる．Intel FPGA SDK for OpenCL ではメモリモードが 2 つあり，1 つ目のモードは，メモリ空間を 1KiB 毎にインターリーブし，それぞれを異なるメモリチャンネルへ割り当てる．2 つ目のモードは，インターリーブをしないモードであり，例えば 2GiB のメモリを 2 枚搭載している環境では，4GiB のメモリ空間中，下位 2GiB がチャンネル 1 に，上位 2GiB がチャンネル 2 に割り当てられる動作となる．

OpenCL を用いて FPGA をプログラミングした際の，メモリ周辺回路の構成図を図 8 に示す．ただし，FPGA の場合，ボード毎に FPGA チップの種類やメモリの構成が異っ

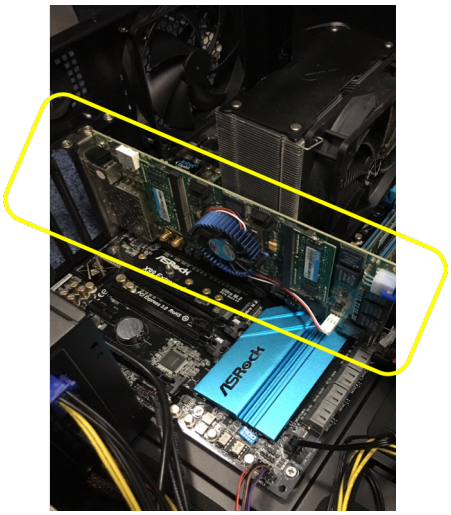

図 7: DE5-net ボードを搭載したシステムの写真．黄枠で囲われた部分がボード本体であり，マザーボードとは PCI Express スロットで接続．

ており，図 8 は本章の実験で用いる DE5-Net ボードを想定したものであり，FPGA には Intel Stratix V を搭載し，外部メモリとして DDR3 メモリとして 2 チャンネル搭載するボードを想定している．

OpenCL カーネルとメモリ間のデータのやりとりは，図 8 で OpenCL Original Bus と示している OpenCL 用メモリネットワークと Avalon バスを通過する．また，図 8 にあるように，1 メモリチャンネルあたり 512bit のデータ幅で接続されている．Avalon バス側は Avalon 標準のコンポーネントで構成されているが，OpenCL のメモリネットワーク側の回路は aoc コンパイラが生成する．OpenCL のメモリアクセス間の調停や前述したインターリーブの処理はこの段階で行われる．

Intel FPGA SDK for OpenCL が生成する回路では，OpenCL カーネルから生成されるロジックが動作するクロック周波数と，メモリコントローラが動作するクロック周波数は独立している．OpenCL カーネル側のクロック周波数は生成された回路によって可変であり OpenCL コードの内容によって変化する．一方で，メモリコントローラ側のクロック周波数は接続するメモリの規格によって固定である．DDR3 メモリを利用する場合は，メモリ動作周波数の 4 分の 1 固定となり，例えばメモリが 800MHz 動作の場合，コントローラクロックは 200MHz である．メモリコントローラ側と，カーネル側の周波数の境界には Avalon-MM 用の Clock Crossing Bridge が挿入され，異なる周波数で動作する回路間でのデータ交換を行う．

5.3 OpenCL 版の実装

OpenCL のみで内積計算を行うコードを図 9 に示す．メモリアクセスおよび演算には，OpenCL の組み込みベクタ型を用いており，float16 型は float 型の変数が 16 要素

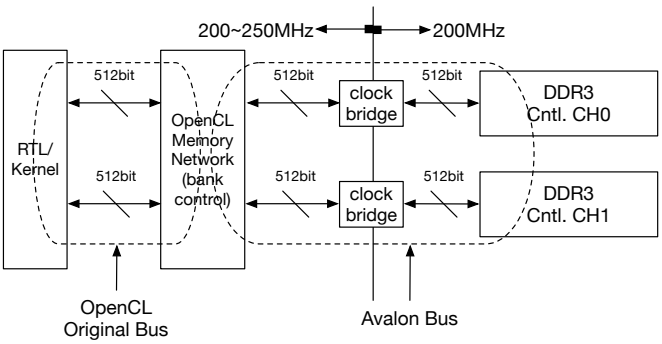

図 8: aoc コンパイラによって生成される回路のメモリ関係の構成図．ただし，周波数やバス幅などは DE5-Net ボードを想定．

パックされた型である．float16 型の各要素には順番に 0〜9, a〜f の名前が付いており，その名前でアクセスする要素を指定する．また，`dot` 関数は OpenCL の組込み関数であり 4 要素までのベクタの内積を求められる．

ループイテレーションをまたいで総和を計算する部分については，Shift Register による最適化を適用する．この手法は Programming Guide [11] で推奨されており，また，Zohouri らもこの Shift Register の最適化は有用だと述べている [12]．Shift Register を用いて総和を計算することで，動作周波数の向上および演算ループのパイプラインストールを回避する効果があり，性能が向上する．

メモリアクセスに Verilog HDL で実装するヘルパー関数を用いる場合には，演算部はそのままに，配列 a, b へのメモリアクセスの部分を置き換える．ただし，演算結果を変数 `out` に書き出す部分については，1 回の関数呼び出しあたり 1 回しかメモリアクセスを行わず，性能にあたえる影響が小さいため，Verilog HDL による最適化の対象とせず OpenCL による実装のままである．

5.4 Verilog HDL による回路設計

OpenCL に組込む Verilog HDL 回路におけるメモリ関係の制限の 1 つに，メモリバスへの 1 系統あたりのデータバスの幅が BSP によって指定された幅に固定であるという制限がある．DE5-Net ボード用の BSP の場合，図 8 にあるようにデータバスの幅は 512bit 固定である．ただし，バスの動作周波数はカーネルの動作周波数と同じであり，回路の合成結果によって変化するため OpenCL コードによって異なる．

データバスの幅が 512bit ということは，すなわち，1 クロック毎に 512bit のデータしか転送できないということであり，DE5-Net ボードの DDR メモリの理論ピークバンド幅 25.6GB/s の帯域を得るためには，400MHz でデータバスが動作しなければならない．DE5-Net ボード用に作成した OpenCL カーネル回路はコードによって変化するが，一般的に 200〜250MHz の範囲の動作周波数となり，その

```
__kernel __attribute__((max_global_work_dim(0)))
void mydot(
  __global float16 const* restrict a,
  __global float16 const* restrict b,
  __global float* restrict out) {
  float acc[7];
  for (size_t i = 0; i < 7; i++) acc[i] = 0;

  for (uint i = 0; i < (N / 16); i++) {
    float16 v = a[i] * b[i];
    acc[6] = acc[0] +
      (dot(v.s0123, v.s0123) +
       dot(v.s4567, v.s4567) +
       dot(v.s89ab, v.s89ab) +
       dot(v.scdef, v.scdef));
    for (size_t i = 0; i < 6; i++)acc[i]=acc[i+1];
  }

  float tmp = 0;
  for (size_t i = 0; i < 6; i++) tmp += acc[i];
  *out = tmp;
}
```

図 9: OpenCL のみで内積計算を行うコード．Verilog HDL 併用実装では下線部のメモリアクセスを Verilog HDL による実装に置き換える．

動作周波数では DDR メモリの帯域をフルに使うことはできない．したがって，Verilog HDL でメモリ帯域をフルに利用する回路を作成する場合は，図 8 にあるように少なくとも 2 系統のメモリバス接続を持たなければならない．

本実験では，以下の 2 パターンの回路を Verilog HDL で記述し，性能を評価する．1KiB Interleave は OpenCL によるメモリインターリーブを利用する場合，Separate はインターリーブを利用しない場合を表す．なお，Verilog HDL で実装された回路から OpenCL の管理下のメモリにアクセスする場合は，OpenCL のメモリネットワークを経由してアクセスするため，OpenCL 側のメモリインターリーブ設定の影響を受ける．

- 1 Reader, 1KiB Interleave
- 2 Readers, 1KiB Interleave
- 2 Readers, Seperate

1 Reader あたり，1 系統の 512bit 幅のメモリバスとの接続を持ち，各 reader が独立して動作する．各パターンにおける，reader のメモリ読み出しリクエストのアクセス順を図 10 に示す．図 10 において，a, b はそれぞれ内積計算対象の配列を示し，配列要素のデータ型は float である．

Verilog HDL モジュールに接続されるメモリバスはバースト読み出し機能をもっており，最大のバースト長は 16 である．reader はバンド幅を最大化するために，常に 16 連続のバースト読み出しのリクエストを発行するため，図 10 にあるように，各リクエストのメモリアクセスの粒度は $512 \times 16/32 = 256$ 要素となる．

Separate 時は変数単位でどのバンクに割り当てるかを指定できるため，配列 a をバンク 0 に配列 b をバンク 1 に割り当てる．Interleave 利用時には，各配列の先頭は 1KiB にアライメントが取られており，それぞれの配列の中で 256 要素単位毎にバンク割り当てが順々に切り替わる．ただし，各変数の先頭の要素の割り当てバンクは実行時までわからないため，図 10 では，各配列の先頭バンクを *a_0*, *b_0*，それらのバンクの反対側のバンクを *a_1*, *b_1* とする．

5.5 性能評価

性能評価の結果を表 2 および表 3 に示す．配列の大きさは float 型で 524,288 個とし，2 つの配列をあわせて 1,048,576 要素の 4MiB とする．性能の測定には aoc コンパイラが提供しているプロファイラ機能を利用する．aoc のプロファイラ機能はメモリ関係の性能測定に特化しており，OpenCL で記述したプログラムの場合は，コード中の各メモリアクセス点に関する情報を収集できる．今回の実験のように Verilog HDL で自作した回路でメモリアクセスを行う場合は，OpenCL の場合のような細かいデータ収集はできないものの，メモリチャンネルあたりの全体性能は測定できるため，その結果を用いる．なお，aoc のプロファイラでメモリバンド幅を測定する際は，メモリアクセスのない部分も含めて OpenCL カーネル全体の実行時間と，データ転送量を元にしてバンド幅を計算される．

まず，メモリインターリーブを有効にした場合で，OpenCL による実装 *OpenCL, 1KiB interleave* と，Verilog HDL による実装 *Verilog HDL (2 reader), 1KiB interleave* で性能を比較すると，OpenCL で記述した回路と Verilog HDL で記述した回路はどちらも同程度の性能が得られているとわかる．しかしながら，どちらの言語で記述した場合でも，理論ピーク性能と比較すると約 65%の効率であり，2 チャンネルある DDR3 メモリの帯域を使い切れていないことがわかる．

次に，メモリインターリーブを無効にした場合で比較する．OpenCL による実装 *OpenCL, Separate* と，Verilog HDL による実装 *Verilog HDL (2 Reader), Separate* で性能を比較すると，OpenCL で実装したものが理論ピーク性能比 82.6%なのに対して，Verilog HDL で実装したものが 89.8%であり，Verilog HDL で実装したものの方が高性能であることがわかる．

OpenCL による実装と Verilog HDL による実装どちらの場合でも，メモリインターリーブが無効の場合の方が有効な場合よりも性能が良く，aoc コンパイラが生成するインターリーブを制御するロジックによって性能が律速されていることがわかる．

Verilog HDL (1 Reader), 1KiB interleave の性能に注目すると，62.41 byte/cycle の性能が得られているとわかる．1 Reader あたり 512bit のデータバスを持つこと

1 Reader / Interleaved

	Reader 1
req 0	a[0:255], bank=a_0
req 1	b[0:255], bank=b_0
req 2	a[256:511], bank=a_1
req 3	b[256:511], bank=b_1
…	…

2 Reader / Interleaved

	Reader 1	Reader 2
req 0	a[0:255], bank=a_0	a[256:511], bank=a_1
req 1	b[0:255], bank=b_0	b[256:511], bank=b_1
req 2	a[512:767], bank=a_0	a[768:1023], bank=a_1
req 3	b[512:767], bank=b_0	b[768:1023], bank=b_1
…	…	…

2 Reader / Separate

	Reader 1	Reader 2
req 0	a[0:255], bank=0	b[0:255], bank=1
req 1	a[256:511], bank=0	b[256:511], bank=1
req 2	a[512:767], bank=0	b[512:767], bank=1
req 3	a[768:1023], bank=0	b[768:1023], bank=1
…	…	…

図 10: Verilog HDL を用いる場合の各設計パターン時におけるそれぞれの reader のリクエストの順番.

から，byte/cycle の理論ピークは 64 byte/cycle であり，62.41byte/cycle は 97.5%の効率となる．したがって，インターリーブを利用している場合でも，単一のメモリバスでメモリにアクセスする場合は効率が良いことから，複数の回路がメモリに同時にアクセスする際の調停機構に性能低下の原因があると考えられる．また，インターリーブを利用しない場合には 2 つの Reader がそれぞれ異なるバンクにアクセスし調停の必要がなく，このこともインターリーブを利用しない場合に性能が良い理由として考えられる．

6. おわりに

本稿では，ヘルパー関数方式と I/O Channel 方式の 2 つの手法を用いて OpenCL と Verilog HDL の混合記述により FPGA をプログラミングする方法について述べた．

ヘルパー関数を用いて混合記述を行い，内積計算コードの最適化を行なった．OpenCL によるメモリインターリーブを有効にした場合では，OpenCL のみで記述したプログラムと混合記述を行ったプログラムで性能に差がなかったが，メモリインターリーブを無効にした場合では，混合記述を行ったプログラムで理論ピーク性能の約 90%の実効性能を達成し，OpenCL のみで記述したプログラムの性能を上まわった．メモリアクセスの効率が重要なプログラムでは，メモリアクセスの部分を Verilog HDL で記述し最適化することで，性能が改善することがわかった．また，今後の課題として，OpenCL によるメモリインターリーブ部分の性能が悪いことが判明したため，メモリインターリーブ部分を Verilog HDL で自作する方式を検討する．

I/O Channel を用いる方式では，ヘルパー関数方式よりもより低レイヤーな部分で FPGA の外部ペリフェラル接続にアクセスでき，LED の制御を OpenCL コードから行えることを確認した．OpenCL から外部ペリフェラルが制御できるということは，FPGA に搭載されている高速な通信機構や，PCI Express を経由して他のデイバスと通信できるということであり，Accelerator in Switch の実現に重要な要素である．

今後は，Accelerator in Switch のコンセプトを実証するために，OpenCL を用いて FPGA 間の通信する方法および実アプリケーションへの適用方法について検討を進める．また，Intel の最新の FPGA である Arria 10 を用いた性能評価を進める．Arria 10 を用いることで，より高速な回路の実現や，より多くのリソースが利用できるだけでなく，Arria 10 から DSP に追加された IEEE 互換の単精度浮動小数点数の演算ユニットを利用することで単精度浮動小数点数を用いるアプリケーションの性能向上が期待できる．

筑波大学計算科学研究センターでは，Accelerator in Switch のコンセプトの実証実験と実アプリケーションへの適用を目指し，同大学で開発が続けられている PACS シリーズ・スーパーコンピュータの次世代機として PACS-X (PACS version 10) の開発を計画しており，そのプロトタイプとして PPX (Pre-PACS-X) の導入を進めている．同システムは Arria 10 に 2 channel の 40Gb Ethernet ポートを備えた FPGA ボード，NVIDIA P100 GPU を 2 基搭載しており，このプラットフォームを用いた Accelerator in Switch の実証実験を今後推進していく予定である．

謝辞 本研究の一部は，JST-CREST 研究領域「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」研究課題「ポストペタスケール時代に向けた演算加速機構・通信機構統合環境の研究開発」，及び文部科学省研究予算「次世代計算技術開拓による学際計算科学連携拠点の創出」による．また，本研究の一部は，「Intel University Program」を通じてハードウェアおよびソフトウェアの提供を受けており，Intel の支援に謝意を表する．

参考文献

[1] Hanawa, T., Kodama, Y., Boku, T. and Sato, M.: Interconnection Network for Tightly Coupled Accelerators Architecture, *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*, pp. 79–82 (online), DOI: 10.1109/HOTI.2013.15 (2013).

[2] 筑波大学：HA-PACS プロジェクト，筑波大学（オンライン），入手先 ⟨http://www.ccs.tsukuba.ac.jp/research/research_promotion/project/ha-pacs⟩（参照 2017-02-01）.

[3] Kuhara, T., Tsuruta, C., Hanawa, T. and Amano, H.: Reduction calculator in an FPGA based switching Hub for high performance clusters, *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–4 (online), DOI: 10.1109/FPL.2015.7293985 (2015).

[4] Tsuruta, C., Miki, Y., Kuhara, T., Amano, H. and Umemura, M.: Off-Loading LET Generation to PEACH2: A Switching Hub for High Performance GPU Clusters, *SIGARCH Comput. Archit. News*, Vol. 43, No. 4, pp. 3–8 (online), DOI: 10.1145/2927964.2927966 (2016).

[5] 大島聡史，塙 敏博，片桐孝洋，中島研吾：FPGA を用

実装	Interleave	周波数 [MHz]	Logic[%]	DSP[%]	Memory[%]	RAM[%]	帯域 [MB/s]	効率 [%]	B/cycle
Verilog HDL (1 Reader)	1KiB	236.35	21.67	12.50	3.06	12.48	14751.2	57.6	62.41
Verilog HDL (2 Reader)	1KiB	247.83	22.67	12.50	3.30	15.04	16668.4	65.1	67.26
OpenCL	1KiB	241.02	22.05	12.50	3.18	13.95	16817.1	65.7	69.77
Verilog HDL (2 Reader)	Separate	242.77	21.21	12.50	3.07	13.01	22976.6	89.8	94.64
OpenCL	Separate	247.21	21.63	12.50	2.95	12.07	21150.2	82.6	85.56

表 2: 内積計算の実装毎の使用リソースおよび性能比較．周波数: OpenCL カーネル回路の動作周波数，Logic: LE の使用率，DSP: DSP Block の利用率，Memory: 内蔵 SRAM の使用率 (bit 単位)，RAM: 内蔵 SRAM の使用率 (SRAM ブロック単位)，帯域: 実効メモリバンド幅，効率: DE5-Net ボードの理論ピークに対する性能比，B/cycle: 1 クロックあたりの転送性能．

表 3: 内積計算の実装毎の性能比較．

実装	Interleave	性能 [M FLOPS]
Verilog HDL (1 Reader)	1KiB	3687.8
Verilog HDL (2 Reader)	1KiB	4167.1
OpenCL	1KiB	4204.3
Verilog HDL (2 Reader)	Separate	5744.2
OpenCL	Separate	5287.6

いた疎行列数値計算の性能評価，技術報告 1，東京大学情報基盤センター, 東京大学情報基盤センター, 東京大学情報基盤センター, 東京大学情報基盤センター (2016).

[6] 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，技術報告 40，東京大学情報基盤センター, 東京大学情報基盤センター, 東京大学情報基盤センター, 東京大学情報基盤センター (2016).

[7] Winterstein, F., Bayliss, S. and Constantinides, G. A.: High-level synthesis of dynamic data structures: A case study using Vivado HLS, *2013 International Conference on Field-Programmable Technology (FPT)*, pp. 362–365 (online), DOI: 10.1109/FPT.2013.6718388 (2013).

[8] Hill, K., Craciun, S., George, A. and Lam, H.: Comparative analysis of OpenCL vs. HDL with image-processing kernels on Stratix-V FPGA, *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 189–193 (online), DOI: 10.1109/ASAP.2015.7245733 (2015).

[9] Terasic: DE1-SoC Board, Terasic (online), available from ⟨`http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=205&No=836&PartNo=1`⟩ (accessed 2017-02-01).

[10] Terasic: DE5-Net FPGA Development Kit, Terasic (online), available from ⟨`http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=158&No=526`⟩ (accessed 2017-02-01).

[11] Intel: Intel FPGA SDK for OpenCL Programming Guide, Intel (online), available from ⟨`https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/opencl-sdk/aocl_programming_guide.pdf`⟩ (accessed 2017-02-01).

[12] Zohouri, H. R., Maruyama, N., Smith, A., Matsuda, M. and Matsuoka, S.: 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, Piscataway, NJ, USA, IEEE Press, pp. 35:1–35:12 (online), available from ⟨http://dl.acm.org/citation.cfm?id=3014904.3014951⟩ (2016).