

# OpenCL 対応 FPGA 間光リンク接続フレームワーク CIRCUS と SMI の性能評価

柏野隆太<sup>1,a)</sup> 小林諒平<sup>2,1</sup> 藤田典久<sup>2,1</sup> 朴泰祐<sup>2,1</sup>

概要：近年、高性能分野において FPGA に対する期待が高まっている。高位合成により開発の障壁が低下し、強力な通信性能をもつことが可能な FPGA は従来のシステムでは高速化できない種類のアプリケーションに対しても効果的に働く可能性がある。これらの FPGA の特徴を最大限に活用するためには、FPGA に特化した通信フレームワークが必要となる。既にこの研究は行われており、筑波大学から CIRCUS、チューリッヒ工科大学から SMI が提案されている。いずれも 40~100Gbps の光リンクを OpenCL から利用可能とするもので、今後の FPGA の HPC 利用において重要なパーツとなると考えられる。本報告では、この 2 つの手法、CIRCUS と SMI について実機性能評価を行い、その特性を比較する。

## 1. はじめに

今日の高性能計算 (High Performance Computing, HPC) 分野において、CPU と併せて演算加速装置を利用する事例が増えている。最も良く利用されている演算加速装置は、高い演算性能とメモリバンド幅をもつ GPU (Graphics Processing Unit) であるが、さらなる選択肢として FPGA (Field Programmable Gate Array) への期待が高まっている。FPGA とはプログラムにより所望の論理回路を実現することを可能にする演算加速装置である。

これまで、FPGA は開発に要するコストが非常に高く、HPC 分野への応用は困難であった。しかし、近年の高位合成 (High Level Synthesis, HLS) 技術の発達により開発の障壁が下がってきている。HLS とは、従来の開発で用いられていたハードウェア記述言語 (Hardware Description Language, HDL) より抽象度の高い言語 (C, C++, OpenCL など) により FPGA の開発を可能にする技術である。HDL はクロック単位・ビット単位という粒度での設計となるため、実装に必要な記述量が膨大になり、加えてハードウェアに対する理解が求められる。これに対して、HLS ではソフトウェアのアルゴリズムと同等の抽象度で開発を行うことができるため、記述量と要求されるハードウェアに関する知識量が少なくなる。すなわち、HLS により、FPGA に精通していない研究者に対しても FPGA を利用することが可能になっている。

最先端の FPGA が他の種類の演算加速装置と比べて優れている特徴として、主に次の 2 つが挙げられる。

- パイプライン並列を実現することができる。
- 高い通信性能を有する。

1 つ目の特徴と対照的なものが、GPU に代表される演算加速装置の特徴であるデータ並列性である。データ並列性とは、同じ処理を異なるデータに対して同時並列に実行することで得られる並列性である。すなわち、高いデータ並列性を実現するためには依存関係の無いデータが十分に存在し、それらに対して粗粒度の処理が行われる状況が求められる。言い換えると、多数の条件分岐や少ないデータ量、細粒度処理といった、これらの条件を満たさない状況ではデータ並列性を発揮することができない。このような状況に対して効果的に働くのが、パイプライン並列である。異なる処理を同時並列に実行されるため、細粒度で依存関係がある処理に対しても並列性を得ることができる。

2 つ目の特徴は、利用する FPGA ボードにより異なる。最新の FPGA ボードでは、最大で 100Gbps × 4 の通信性能をもつ外部通信光リンクを利用できる。この光リンクは FPGA の内部回路と直接接続されているため、低レイテンシで異なる FPGA 間の通信を実現することができる。

これらの 2 つの優れている点を最大限に活用する技術として、FPGA 間通信フレームワークがある。FPGA 間通信フレームワークは、複数の FPGA 間で巨大なパイプラインを構成するというパイプライン通信を実現することを可能にする。パイプライン通信の利点は、通信と演算が同時並列に行われることにある。計算時間により通信時間を隠蔽することが可能になり、通信が多いアプリケーションに

<sup>1</sup> 筑波大学 大学院履行情報生命学術院 システム情報工学研究部 情報理工学位プログラム

<sup>2</sup> 筑波大学 計算科学研究センター

<sup>a)</sup> kashino@hpcs.cs.tsukuba.ac.jp

対しても高速化することが期待できる。また、パイプライン並列を活用することで、主にデータ並列性を活用する従来のシステムでは高速化できない種類のアプリケーションに対しても高速化することが期待できる。すでに、FPGA 間通信フレームワークの実装として Communication Integrated Reconfigurable CompUting System (CIRCUS)[1] と Streaming Message Interface (SMI)[2] の 2 つの手法が提案されている。前者は筑波大学、後者はチューリッヒ工科大学 (ETHZ) による提案である。両者はともに 40~100Gbps の光リンクを HLS を使用した OpenCL のプログラムから利用可能にするというものである。両手法は、多くの共通点を持ち、今後の FPGA の HPC 利用において重要な役割を担うと考えられる。しかしながら、これら 2 つの手法を比較した研究は存在しない。

本報告では、FPGA 間通信フレームワークを実現する 2 つの手法、CIRCUS と SMI の特性を比較し報告する。

## 2. 関連研究

HPC 分野での FPGA を用いた研究として、鶴田らによる [3] がある。この研究では、PEACH2 ボードを搭載した HA-PACS/TCA を利用して、N 対シミュレーションの高速化を行っている。PEACH2 ボードとは、異なるノード間の GPU の直接通信を可能にする FPGA ボードであり、主に転送速度の改善に使用されていた。この FPGA に対して、通信回路のみならず演算回路を組み込むことにより通信と演算を同時に行う仕組みを実装している。この研究では、FPGA に演算をオフロードすることにより CPU に対して 7.2 倍の高速化を実現している。

ネットワークで接続された複数台の FPGA を使用して高速化を目指した研究として、[4] がある。この研究では、データセンター内に FPGA による 2D トーラスネットワークを作成し、Bing Web Serch エンジンの高速化を行っている。この研究では、FPGA によるオフロードにより、スループットが 2 倍に改善し、かつレイテンシーの 29% 短縮が実現している。この開発に用いた言語は Verilog HDL であり、開発にあたり大きなコストが必要であったため、高位合成などによる開発の高速化が課題であるとされている。

以上より、FPGA はアクセラレータとしてアプリケーションの高速化させる事が期待できる。そして、strong scaling による高速化を考えると、FPGA 間通信技術は重要であると考えられる。

## 3. Intel FPGA SDK for OpenCL

### 3.1 OpenCL

OpenCL は C 言語をベースとしたフレームワークである。CPU, GPU, FPGA などの異なる演算器の間においても統一した開発環境を提供する目的で開発された。各ベンダが専用の開発環境を提供することで、異なる演算加

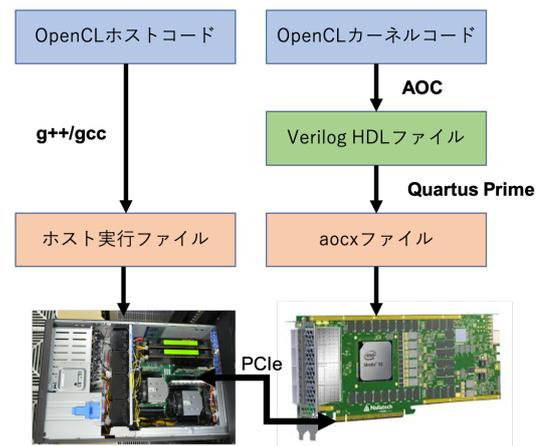


図 1 Intel FPGA SDK for OpenCL 開発フロー

速装置に対しても統一した API を用いて動作を記述することが可能になる。Intel の提供する FPGA の開発環境として、Intel FPGA SDK for OpenCL が提供されている。CIRCUS・SMI は共に、Intel FPGA SDK for OpenCL 開発環境の利用を前提して実装されている。

OpenCL では、ホストコードとカーネルコードの 2 種類のプログラムを別々に作成する必要がある。前者がホスト上、すなわち CPU 上で動作するプログラムであり、後者が演算加速装置上で動作するプログラムである。OpenCL では、ホストプログラムがカーネルプログラムを呼び出し、制御を行うという形式で動作する。

#### 3.1.1 開発の流れ

Intel FPGA SDK for OpenCL は、ホストドライバ・OpenCL コンパイラを含む統合開発環境である。この SDK と既存の C/C++ コンパイラを使用することで、FPGA の開発を行う事ができる。図 1 に Intel FPGA SDK for OpenCL 環境を利用した開発の流れを示す。OpenCL ホストコードは gcc や Intel Compiler などの一般的に広く使用されるコンパイラを用いて SDK の提供するドライバと共にコンパイルする。そして、OpenCL カーネルコードは SDK から提供されるコンパイラ AOC(Altera Offline Compiler) を用いてコンパイルする。OpenCL カーネルコードは、一度 HDL に変更された後に、FPGA 回路情報を含む aocx ファイルへと変更される。aocx ファイルが、ホスト実行ファイル中の OpenCL API により FPGA へ読み込まれることで FPGA の再構成が行われる。

#### 3.1.2 Board Support Package

aocx ファイルの形で与えられる FPGA の回路は、2 つの部分に大別される。1 つ目がユーザーが記述した OpenCL カーネルプログラムから生成された回路であり、2 つ目が BSP (Board Support Package) から生成された回路である。全ての FPGA ボードから統一した OpenCL API を利用する環境を作るためには、FPGA ボード固有の機能/回路と OpenCL API の間を接続する仕組みが必要となる。

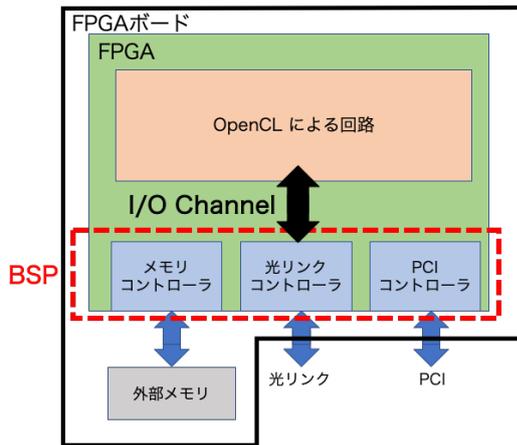


図 2 BSP と IO Channel

この役割を担うのが BSP であり、FPGA チップの違いや FPGA ボードにおける外部ペリフェラルの差異を隠蔽する働きをする。各 FPGA ボードに固有な機能は全て BSP に含まれている。図 2 における赤枠で囲まれた部分が BSP である。BSP には PCIe のコントローラや外部メモリコントローラなどが含まれる。各 FPGA ボードに対応した公式の BSP が各 FPGA ボードベンダーから提供されている。また、BSP を独自に開発して組み入れることも可能であり、ベンダーが提供する公式の BSP には存在しない機能を追加することができる。独自に開発した BSP を利用しているのが CIRCUS であり、Bittware の提供する BSP の利用を前提にしているのが SMI である。

### 3.1.3 Channel 機能

Intel FPGA SDK for OpenCL では、標準の OpenCL API に加えて、FPGA に特化した機能である Channel 拡張機能を備えている。Channel を利用することで、異なるカーネル間における直接のデータ転送が可能になる。OpenCL における一般的なカーネル間データ転送では、グローバルメモリを介して通信が行われる。しかし、FPGA は外部メモリへのアクセスに大きなコストを要するという特徴がある。そこで、Channel を利用することで、外部メモリへのアクセスをすることなくデータ転送を行うことが可能になる。つまり、Channel により、FPGA 内部のカーネル間でパイプラインを構成することが可能になる。Channel の中でも、OpenCL カーネルと BSP の間の通信を実現するものを IO Channel という。IO Channel は BSP とカーネルを結び働きをする(図 2 を参照)。光リンクのコントローラは BSP に含まれるため、OpenCL カーネルから光リンクを利用するためには IO Channel を利用する必要がある。SMI と CIRCUS は共に、IO Channel を用いて光リンクへとデータを送信している。

## 4. CIRCUS

### 4.1 概要

CIRCUS[1] は筑波大学計算科学研究センターが開発した FPGA 間高速通信フレームワークである。CIRCUS は FPGA 間におけるパイプライン通信を実現することを目的として開発が行われている。CIRCUS の提供する CIRCUS Channel を利用することで、ユーザーは FPGA 間の通信を記述することができる。IntelFPGA SDK for OpenCL の提供する Channel は、単一の FPGA 内部におけるパイプライン通信を実現するものであり、CIRCUS Channel は複数の FPGA 間においてパイプライン通信を実現するものである。

CIRCUS は、HDL で記述されたルータモジュールと OpenCL 記述された制御カーネルの 2 つの基本コンポーネントから構成される。前者は BSP、後者は OpenCL カーネルプログラムとして提供される。制御カーネルは、CIRCUS Channel と BSP のインターフェースとなるため、ユーザーが利用する CIRCUS Channel に対応した内容になる必要がある。そのため、CIRCUS は制御カーネルを生成するジェネレーターを提供している。ユーザーが記述した Channel 情報を含む XML ファイルをこのジェネレーターに読み込ませることで、適切な制御カーネルが生成される。OpenCL で記述したユーザープログラムを、ルータモジュールおよび制御カーネルと共に、合成することで CIRCUS を利用することができるようになる。

CIRCUS の提供するジェネレーターは制御カーネルの生成と同時に、ホスト側から CIRCUS の制御を行う OpenCL ホストコードを生成する。ここには、初期化時における制御カーネルの起動処理を行う関数が記述されており、CIRCUS の利用にあたり必要となる初設定を行うために必要となる。

ユーザーが記述する OpenCL カーネルプログラム内における CIRCUS の使用例を図 3 に示す。CIRCUS Channel である simple\_out, simple\_in には同じ Channel ID が付与されており、simple\_out へ書き込んだデータが simple\_in へ送信される。CIRCUS Channel への読み書きは通常のチャネルの扱いと同様である。

### 4.2 ルーティング機構

CIRCUS では各 FPGA は固有の NodeID をもつ。さらに、FPGA 内における各 CIRCUS Channel も同様に固有の ID をもち、この 2 つの ID を指定することで任意の配送先を指定することができる。

CIRCUS の全体像を図 4 に示す。ユーザーの定義するカーネルが左端にある。そして、制御カーネルが中間にある。制御カーネルはパケットの送信を行う Send カーネル、

```

1 //送信側
2 __kernel void sender(__global uint16* restrict x, int n) {
3     for (inti= 0; i< n; i++) {
4         uint16 v = x[i];
5         write_channel_intel(simple_out, v);
6     }
7 }

1 //受信側
2 __kernel void receiver(__global uint16* restrict x, int n) {
3     for (inti= 0; i< n; i++) {
4         uint16 v = read_channel_intel(simple_in, v);
5         x[i] = i;
6     }
7 }
    
```

図 3 CIRCUS 利用例

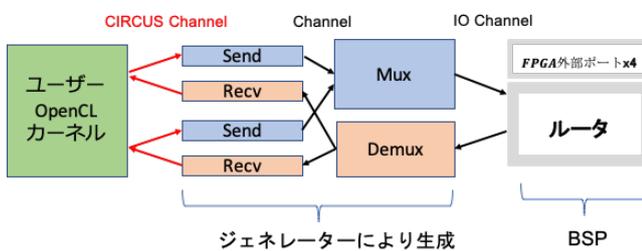


図 4 CIRCUS 全体像

受信を行う Recv カーネル，BSP とのインターフェースである Multiplexer/Demultiplexer からなる。ユーザーカーネルから CIRCUS Channel を介して Send カーネルへデータを送信することにより FPGA 間の通信が行われる。そして反対に，Recv カーネルから CIRCUS Channel を介してデータを受信する。

CIRCUS は同時並列による通信を可能にするため，カーネル側からルータモジュールへの入力は 1 ポートに制限している。カーネル側の Send カーネルから送信される複数の出力を 1 つのポートに集約する役割を担うのが，中段にある Multiplexer である。同様に，ルータモジュールからカーネル側への出力ポートも 1 つである。ルータモジュールからの出力をカーネル側の Recv カーネルへ分配する役割を担うのが Demultiplexer である。このとき，Recv カーネルを識別するために用いられるのが Channel ID である。

CIRCUS のルータモジュールは HDL で記述され，BSP に組み込まれている。ルータモジュール内部にはクロスバーと調停機能が備わっており，宛先が同じパケット同士の衝突を防止しつつ，複数のパケットを同時に処理することができる。この機能の実現のためには入出力ポートを一定数以下にする必要があり，CIRCUS ではルータモジュールの前に Multiplexer/Demultiplexer を挿入することで対応している。

また，現在の CIRCUS のルータモジュールにはフロー制御機能が実装されていない。そのため，通信路内部のバッファが溢れるとパケットが損失してしまう。

## 5. Streaming Message Interface

### 5.1 概要

SMI [2] は FPGA に特化した分散メモリプログラミングモデルである。SMI のソースコードは全てオープンソースとして公開されている。SMI が想定している開発環境は Intel SDK for OpenCL および、その拡張機能 IO Channel が利用可能な FPGA ボードである。

SMI は，インターフェースカーネルと実際の送信を行うカーネルの 2 つから構成されている。ユーザーはインターフェースの呼び出しのみを記述するだけでよく，その機能の実現のための OpenCL カーネルプログラムは SMI の提供するジェネレーターから生成される。図 6 が SMI の実装の概要図である。

SMI インターフェースには P2P と Collective 通信の 2 種類が用意されている。また，CIRCUS と同様に，ジェネレーターはホスト側から SMI の制御を行う OpenCL ホストコードを生成する。このコードを呼び出すことで FPGA 間通信を実現するカーネルを起動させることができる。

### 5.2 SMI インターフェース

SMI では P2P 通信と Collective 通信を実現するインターフェース [6] が用意されている。これらのインターフェースを利用することで，様々な通信を記述することができる。SMI では P2P 通信のインターフェースとして Push と Pop が用意されている。前者はデータの送信を行い，後者はデータの受信を行う。送信側のカーネルから Push を呼び出し，受信側のカーネルから Pop を呼び出すことで，P2P 通信を記述することができる。Push/Pop を利用するためには SMI.Open\_send\_channel / SMI.Open\_receive\_channel 関数を用いて SMI channel を生成する必要がある。生成された SMI channel を Push/Pop の引数にデータとともに与えることで P2P 通信を実現できる。さらに Collective 通信のインターフェースとして，Broadcast, Reduce, Scatter, Gather が用意されている。これらのインターフェースは後述するルーティング機構を利用して実際の通信を行っている。

ユーザーが記述する OpenCL カーネルプログラム内における SMI の使用例を図 5 に示す。送信側は SMI.Open\_send\_channel を呼び出し，SMI Channel を生成する。対応する受信側は SMI.Open\_recv\_channel を呼び出し，SMI Channel を生成する。受信側は SMI.Push を呼び出すことでデータが転送され，受信側は SMI.Pop を呼び出してデータを受信する。

```

1 //送信側
2 _kernel void sender(_global double restrict x, int n, ...)
3 {
4     SMI_Channel chan=SMI_Open_send_channel(...);
5     for (inti= 0; i < n; i++) {
6         double v = x;
7         SMI_Push(&chan,&v);
8     }
9 }

1 //受信側
2 _kernel void receiver(_global double* restrict x, int n,
3     ...)
4 {
5     SMI_Channel chan=SMI_Open_receive_channel(...);
6     for (inti= 0; i < n; i++) {
7         double recv;
8         SMI_Pop(&chan,&recv);
9         x[i] = recv;
10 }

```

図 5 SMI 利用例

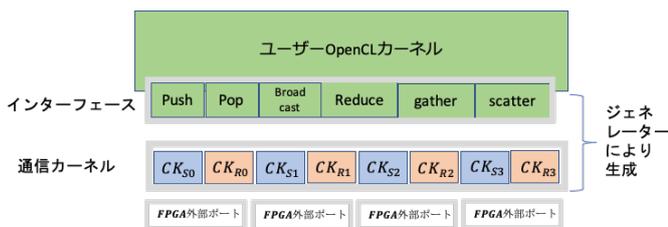


図 6 SMI 全体像

### 5.3 ルーティング機構

SMI は、Bittware が提供している BSP を使用している。この BSP にはフロー制御機能が備わっており、通信路内部のバッファを超過しそうな状態を検知すると通信量を制御してバッファ溢れを回避できる。SMI におけるパケットの送信は、送信用カーネル (Send Communication Kernel,  $CK_S$ ) および受信用カーネル (Receive Communication Kernel,  $CK_R$ ) が行う。 $CK_S$  および  $CK_R$  は光リンク毎にそれぞれ 1 対ずつ実装される。本研究では光リンクを 4 つ持つ FPGA ボードを使用しているため、送受信それぞれ 4 つのカーネルが生成されている。

$CK_S$  は、アプリケーションや別の光リンク  $CK_S$  からパケットを受け取り、担当する光リンクや別の  $CK_S$  へパケットを送信する。一方、 $CK_R$  は、FPGA 光リンクや別の  $CK_R$  からデータを読み込み、アプリケーションや別の光リンク  $CK_R$  へパケットを送信する。

図 7 は、1 つの光リンクを担当する  $CK_S$  および  $CK_R$  まわりの接続関係図である。同じポートを担当する  $CK_S$  および  $CK_R$  は互いにパケットを交換することができる。また、異なる光リンク  $CK_S$  間においてもパケット交換が可能である。

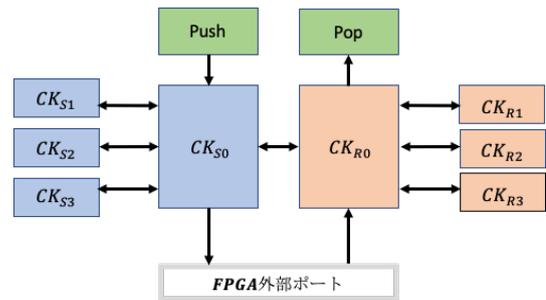


図 7  $CK_S$  まわりの接続関係図

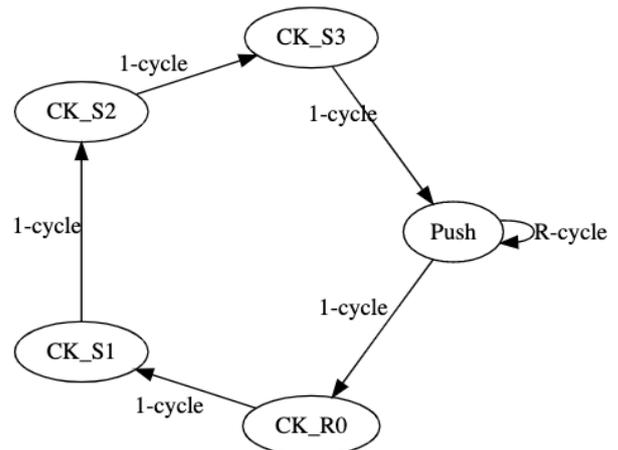


図 8  $CK_{S0}$  読み込み先切り替えの状態遷移図

各  $CK_S$  および  $CK_R$  はパケットを受け取ると、先頭のヘッダ情報と自身のもつルーティングテーブルの情報を照会し、適切な  $CK_S$  または  $CK_R$  へ転送する。これらの複数ある読み込み先の切り替えは、ラウンドロビンで行われる。すなわち、各クロック毎に接続先を選択しパケットの有無を確認する。パケットが存在する場合は次のサイクルも同じ接続先を読み込み、パケットが存在しない場合は読み込み先の切り替えを行う。同じ接続先の連続最大読み込み回数は、最適化パラメータ  $R$  によって設定可能である。 $R=1$  の場合は、各サイクル毎に別の接続先を読み込む。この値が大きいほど、同じ接続先のデータを連続して送信することができる。

使用されている SMI インタフェースが Push カーネル 1 つのみと仮定して、送信カーネル  $CK_{S0}$  の読み込み先の切り替えの様子を状態遷移図で表したものが図 8 である。この例では、Push カーネルがアプリケーションからのデータを受け取り、 $CK_{S0}$  へ送信する。そして、 $CK_{S0}$  は適切な宛先へとデータを送信する。 $CK_{S0}$  は Push カーネルからのデータを連続で  $R$  回送信した後、他の読み込み先に切り替える。再び、Push カーネルからデータが読み込まれるのは最低でも 5 クロック後となる。使用する SMI インタフェースが増えるほど遷移図を構成する要素数が増え、再び読み込むまでに要するオーバーヘッドが増加する。

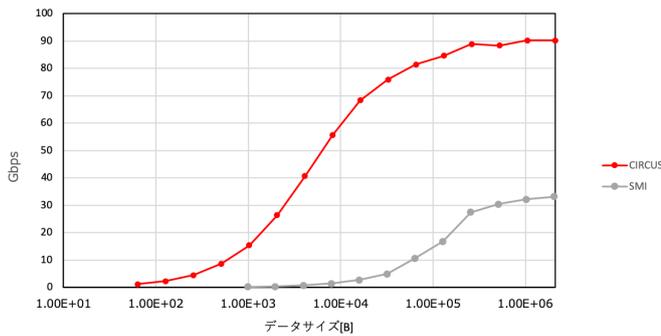


図 9 SMI と CIRCUS のバンド幅性能比較

## 6. SMI と CIRCUS の性能比較

### 6.1 実験環境

本研究では、筑波大学計算科学研究センターで運用中の Cygnus スーパーコンピュータと、同センターで運用している実験クラスター Pre-PACS-X (PPX) を性能評価に用いた。いずれも、Intel Stratix 10 FPGA を搭載した Bittware 520N ボードを複数台利用可能であり、それぞれの FPGA ボードが光リンクにより相互に接続されている。

バンド幅の測定は、両フレームワークとも PPX 上における 2 つの FPGA を用いた。そして、Broadcast 通信の測定について、CIRCUS 側では PPX を利用し、SMI 側で Cygnus を用いた。前者では 2x2 の 2D トーラスに接続された FPGA を用い、後者で 2x2 のメッシュ上に接続された FPGA を用いた。今回は、4 ノードの Broadcast 通信を比較しており、それぞれのノードへの通信ホップ数の組は両者ともに 1,1,2 となるため循環の有無は結果に大きく影響しないと考えられる。同じ環境を使用しない理由は、CIRCUS の利用にあたり BSP の書き換えが必要になるためである。BSP の書き換えには計算ノードの再起動が必要となり、本番環境として供される Cygnus では事前予約が必要となる。そのため、CIRCUS 側の評価は PPX 環境を利用している。

### 6.2 バンド幅

CIRCUS と SMI のそれぞれの枠組みを利用した FPGA 間通信プログラムを用いて、両フレームワークのバンド幅を測定した。CIRCUS には PINGPONG ベンチマークを用い、そして SMI は公開されているバンド幅測定プログラム [5] をそのまま使用した。実験では、各フレームワークの測定を 100 回行い、それらの平均からバンド幅を算出した。このとき得られた測定結果が図 9 である。

#### 6.2.1 FPGA リソース使用量

CIRCUS により PINGPONG ベンチマークのソース量は表 2 となった。一方、SMI バンド幅計測プログラムにおけるリソース量は表 3 である。SMI 側は送信側と受信側が

別々の aocx として合成されているため、CIRCUS との直接の比較は困難である。しかし、周波数が大きく異なっていることが確認できる。これはルーティング機構の実装が HDL と OpenCL という違いに起因するものと考えられる。

表 1 SMI バンド幅計測プログラム FPGA リソース

カーネル	送信側	受信側
周波数	254.2 MHz	247.5 MHz
ALUTs	171,202	202,385
Registers	367,024	420,729
Logic utilization	25%	27%
RAM blocks	5%	5%

#### 6.2.2 考察

今回の実験に使用した FPGA は最大 100Gbps の性能をもつ光リンクを有する。しかし、SMI が利用する標準 BSP における OpenCL カーネルと光リンクとの間のバス幅は 256-bit である。そのため、SMI の理論ピークバンド幅は動作周波数に 256 を乗じた値に律速される。周波数の値は合成時パラメータや使用する SMI インターフェースにより変動し、おおよそ 230-270MHz の間になる。一方で、CIRCUS は専用の BSP を開発することにより、OpenCL カーネルと光リンクとの間のバス幅を 512-bit に拡張し、この問題を緩和している。そのため、CIRCUS は光リンクの上限である 100Gbps に律速される。

しかしながら、SMI バンド幅測定プログラムによる測定値は理論ピークの半分程度となっている。この原因はラウンドロビンによるルーティング機構にあると考えられる。ラウンドロビンの影響を調べるために、同じ接続先への連続最大読み込み回数 R (デフォルト値は 8) の値を変化させて測定した。さらに、読み込み先を固定しラウンドロビンを停止したもの (R=∞) を測定した。この書き換えを行うと特定の経路以外の通信を行うことができなくなるため、通信フレームワークとしての役割は果たせなくなる。しかし、ラウンドロビン動作が SMI のバンド幅性能へ与える影響の程度を見積もるための比較対象として扱うことができる。これらの測定により図 10 が得られた。また、パケット生成による影響を排除するため、いずれの場合においてもデータをバッファリングせずに送信するようにしている。

図 10 より、ラウンドロビン動作を除去したものが最も性能高く、順に R=127, R=8 となっている。ラウンドロビン動作を除去したときの測定値は、理論ピークの 93% となった。R の値を大きくすると、データが存在する限り同じ宛

表 2 CIRCUS PINGPONG ベンチマーク FPGA リソース

周波数	330.0 MHz
ALUTs	153,959
Registers	323,797
Logic utilization	21 %
RAM blocks	6 %

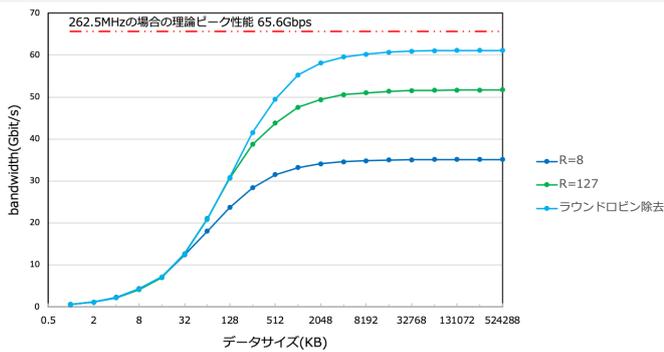


図 10 連続最大読み込み回数 R を変えたときの SMI バンド幅

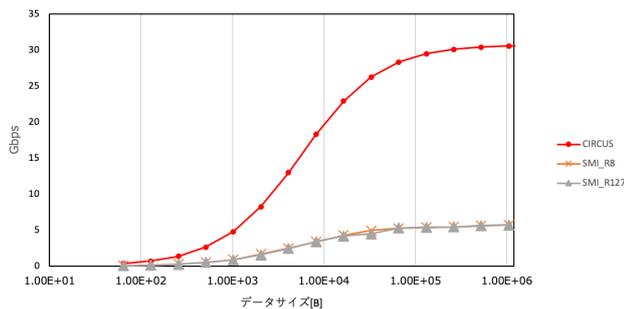


図 11 Broadcast 性能比較

先を優先的に読み込むようになる。すなわち、ラウンドロビンが固定化される。バンド幅測定では送信元と受信元がそれぞれ 1 つに限定されるため、R の値を上げることが、バンド幅性能の向上につながったと考えられる。しかしながら、R の値を上げることにより実際のアプリケーションの全体通信性能が向上するとは限らない。連続最大読み込み回数 R を大きくすると特定の読み込み先のパケットを優先することになるので、通信の公平性が担保されなくなる。カーネル間の通信時間に差が生じることで、同期待ちが発生し、全体の通信性能が悪化する恐れがある。

### 6.3 Broadcast 通信性能

CIRCUS と SMI のそれぞれの枠組みを利用した FPGA 間通信プログラムを用いて、両フレームワークにおける Broadcast 通信の性能を比較した。CIRCUS では新たに Broadcast プログラムを実装し、そして SMI は公開されている Broadcast マイクロベンチマーク [5] を使用した。さらに、SMI では最大読み込み回数 R をデフォルト値の 8 と 127 の 2 つの場合を測定した。実験では、4 台の FPGA を用いて、各フレームワークの測定を 100 回行い、それらの平均から Broadcast の性能を算出した。このとき得られた測定結果が図 11 である。

#### 6.3.1 FPGA リソース使用量

SMI の Broadcast 通信におけるリソース量は表 3 である。CIRCUS の Broadcast 通信におけるリソース量は表 4 である。CIRCUS 側のリソース消費量が大きい原因は、簡

表 3 SMI Broadcast ベンチマーク FPGA リソース

周波数	260 MHz
ALUTs	167,072
Registers	349,700
Logic utilization	23 %
RAM blocks	5 %

表 4 CIRCUS Broadcast ベンチマーク FPGA リソース

周波数	330.0 MHz
ALUTs	212,653
Registers	418,912
Logic utilization	25 %
RAM blocks	6 %

易的な実装を行ったためと考えられる。SMI は Bcast 専用の処理を行う共通のカーネルが実装されているのに対して、CIRCUS では P2P 通信を RECV の数だけ行うという単純な実装になっている。つまり、RECV の数だけ受信するカーネルが用意されており、カーネルが共通化されている SMI 側と比べて消費量が多いと考えられる。

#### 6.3.2 考察

図 11 より Broadcast 通信は、CIRCUS が SMI の約 5 倍近い性能となった。SMI では、通信路上にある送受信カーネル  $CK_S, CK_R$  を経由する毎にラウンドロビンによるルーティングを行う。そのため、実際の配送までにレイテンシを要し、性能差につながったと考えられる。加えて、カーネルの動作周波数も性能差に起因していると考えられる。

SMI において、バンド幅計測の場合と異なり、連続最大読み込み回数 R の値が性能に影響していないことが図 11 からわかる。これは、ルーティングが必要な Broadcast 通信では、読み込み先の固定化が効果を及ぼさないことを意味している。

## 7. まとめ

FPGA 間通信フレームワークである CIRCUS と SMI の特性の比較を行った。CIRCUS は CIRCUS Channel を提供することでユーザーの FPGA 間通信を実現し、一方の SMI は独自のインターフェースを提供することで実現している。このインターフェースにより、SMI は Collective 通信が標準で利用できる。また、SMI は全てが OpenCL で記述されているため移植性に優れている。それぞれの方式は異なるが、両フレームワークともに OpenCL のレベルから FPGA 間通信を利用できる。そして、CIRCUS と SMI の内部のルーティング機構は大きく異なっている。CIRCUS は BSP にルータモジュールを組み込むことによりルーティング機構を実現しており、SMI はラウンドロビンによりルーティング機構を実現している。この実装の差異により、CIRCUS は同時に入力されたパケットを並列に扱えるのに対して、SMI では逐次的に処理されることになる。また、SMI が使用している Bittware BSP のバス幅の

問題により, SMI の理論ピークは CIRCUS より低いものとなる. これらのルーティング機構の実装の違いが CIRCUS と SMI の通信性能差の大きな原因となっているものと考えられる. ただし, SMI と異なり CIRCUS はフロー制御が未実装である. そのため, 現在の CIRCUS は一定以上の通信を行うことができないという制約がある.

### 謝辞

本研究は, 文部科学省「次世代領域研究開発」(高性能汎用計算機高度利用事業費補助金) 次世代演算通信融合型スーパーコンピュータの開発の一環として実施したものである. 本研究成果は筑波大学計算科学研究センターの学際共同利用プログラム (Cygnus) における 2020 年度課題「FPGA-GPU 混載プラットフォームにおける HPC アプリケーションとシステム・ソフトウェアの開発」を利用して得られたものである.

### 参考文献

- [1] Norihisa, F., Ryohei, K., Yoshiki, Y., Tomohiro, U., Kentaro, S. and Taisuke, B.: Performance Evaluation of Pipelined Communication Combined with Computation in OpenCL Programming on FPGA, 2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) (2020 (to be published)).
- [2] Tiziano De Matteis, Johannes de Fine Licht, Jakub Bernek and Torsten Hoefer. Streaming Message Interface: High-Performance Distributed Memory Programming on Reconfigurable Hardware. SC '19: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis November 2019 Article No.: 82.
- [3] Chiharu Tsuruta, Yohei Miki, Takuya Kuhara, Hideharu Amano and Masayuki Umemura. Off-loading LET generation to PEACH2 : A switching hub for high performance GPU clusters . ACM SIGARCH Computer Architecture News 43(4):3-8 April 2016.
- [4] Andrew Putnam, Adrian Caulfield, Eric Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, Eric Peterson, Aaron Smith, Jason Thong, Phillip Yi Xiao, Doug Burger, Jim Larus, Gopi Prashanth Gopal and Simon Pope. A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services. Proceeding of the 41st Annual International Symposium on Computer Architecture (ISCA) — June 2014.
- [5] SMI microbenchmarks  
<<https://github.com/spcl/SMI/tree/master/microbenchmarks>>
- [6] SMI Interface  
<<https://github.com/spcl/SMI/wiki/The-programming-interface>>