情報処理学会論文誌　コンピューティングシステム　Vol.13　No.3　13–28 (Nov. 2020)

OpenCLプログラミングを用いた並列FPGA処理システムの性能評価

藤田 典久[1,2,a)]　小林 諒平[1,2]　山口 佳樹[2,1]　上野 知洋[3]　佐野 健太郎[3]　朴 泰祐[1,2]

受付日 2020年3月31日, 採録日 2020年7月8日

概要：再構成可能なハードウェアの 1 つに Field Programmable Gate Array（FPGA）がある．我々は，FPGA が持つ強力な外部通信機構に注目している．FPGA 開発は低レベルな記述が必要でありコストが高かったが，高位合成（High Level Synthesys，HLS）の技術によって解消されつつある．我々は Communication Integrated Reconfigurable CompUting System（CIRCUS）という FPGA 間通信フレームワークを提唱している．CIRCUS システムを用いることで，通信と演算が一体となったパイプラインを OpenCL で記述できる．筑波大学計算科学研究センターでは 1 ノードあたり 2 FPGA ボードを搭載するスーパコンピュータ Cygnus を運用しており，本論文では Cygnus 上で CIRCUS システムの設計と実装について述べ，また，CIRCUS の通信性能の評価を行う．

キーワード：FPGA，OpenCL，相互結合網

Performance Evaluation of Parallel FPGA System for OpenCL Programming

Norihisa Fujita[1,2,a)]　Ryohei Kobayashi[1,2]　Yoshiki Yamaguchi[2,1]　Tomohiro Ueno[3]
Kentaro Sano[3]　Taisuke Boku[1,2]

Received: March 31, 2020, Accepted: July 8, 2020

Abstract: Field Programmable Gate Array (FPGA) is a kind of reconfigurable hardware. We focus on FPGA's powerful interconnection network capability. We have been proposing Communication Integrated Reconfigurable CompUting System (CIRCUS), which is an inter-FPGA communication framework. CIRCUS system allows us to describe a fused pipeline combining communication and computation in the OpenCL language. Center for Computational Sciences, University of Tsukuba operates Cygnus supercomputer. Each node of Cygnus has 2 FPGAs. In this paper, we describe the design and implementation of CIRCUS system and show the result of its performance evaluation on Cygnus.

Keywords: FPGA, OpenCL, interconnection network

1. はじめに

近年，高性能計算（HPC）の分野で Field Programmable Gate Array（FPGA）が注目されている．従来，FPGA 開発の困難さが FPGA を HPC に適用する際の障壁であったが，高位合成（High Level Synthesys，HLS）の技術によって解消されつつある．最新の FPGA ボードは最大で 100 Gbps×4 と，非常に強力な外部通信ポートを持つ．我々は，この FPGA が持つ高性能な通信機構に注目しており，本研究では高位合成が持つユーザフレンドリーな開発性能と通信機構を組み合わせて用いる．

FPGA の浮動小数点演算，特に倍精度演算における絶対性能は，HPC システムで広く用いられているアクセラレータである Graphics Processing Unit（GPU）には及ば

[1] 筑波大学計算科学研究センター
Center for Computational Sciences, University of Tsukuba, Tsukuba, Ibaraki 305–8577, Japan
[2] 筑波大学システム情報工学研究群
Degree Program in Systems and Information Engineering, University of Tsukuba, Tsukuba, Ibaraki 305–0005, Japan
[3] 理化学研究所計算科学研究センター
RIKEN Center for Computational Science, Kobe, Hyogo 650–0047, Japan
[a)] fujita@ccs.tsukuba.ac.jp

ない．したがって，GPU ではうまく扱えず，演算効率が低くなってしまう問題を FPGA にオフロードする必要がある．低オーバヘッドな直接通信は並列計算，特に強スケーリング計算時に必要不可欠なものと考えており，そのような処理の 1 つとしてアクセラレータ間通信に注目している．すでに述べたように，FPGA は最大で 100 Gbps × 4 の非常に強力な外部通信ポートを持つ．これらのインタフェースは FPGA の内部回路に直接接続されており，これらを用いることで低レイテンシな FPGA 間通信が可能である．NVIDIA GPU は GPUDirect for RDMA（GDR）[1] と呼ばれる技術があり，Network Interface Card（NIC）が GPU のメモリに直接アクセスして通信できる．しかしながら，CPU–GPU 間の同期，PCIe バス，通信ソフトウェアからくるオーバヘッドは依然として残っている．

HPC アプリケーションでは，Message Passing Interface（MPI）が高速通信網を利用するための通信ライブラリとして一般的に用いられている．しかしながら，FPGA の演算アーキテクチャは CPU とまったく異なっているため，我々は MPI の Application Programming Interface（API）を直接 FPGA 向けに移植することは，良い戦略だと考えていない．パイプラインは FPGA における基本的な処理構造であり，OpenCL コンパイラはコード中にあるループからパイプラインを構築する．したがって，演算パイプラインと通信パイプラインを接続し，すべてをパイプラインにして利用することが最適であると考えている．

パイプライン通信を導入する利点は，通信と演算が完全にオーバラップされることである．従来手法では，アプリケーションの中で明示的に通信と演算のオーバラップを記述しなければならなかった．たとえば，演算を止めないために，MPI_Isend や MPI_Irecv といった non-blocking な通信 API を使用しなければならず，また，通信に用いるデータの寿命や，通信とデータの依存関係を考慮にいれてプログラムを記述しなければならない．一方，パイプライン通信では，通信と一体となったパイプラインを自然に得られ，演算と通信をクロックサイクルレベルの粒度でオーバラップできる．我々は，再構成可能なハードウェアが持つこの特性は，GPU といった他のアクセラレータにはないものであり，利点であると考えている．

本研究の目的は，FPGA を用いた並列システムを構築し，FPGA が持つ強力な 100 Gbps 通信を OpenCL から扱えることを示すことである．そして，それを実現するために，我々は Communication Integrated Reconfigurable CompUting System（CIRCUS）システムを提唱している [2]．CIRCUS は OpenCL 高位合成処理系から FPGA 間通信を可能にするものであり，パイプライン通信を基本コンセプトとして通信システムを構築するものである．本論文では，2 つのベンチマークを用いて CIRCUS の性能評価を行う．Pingpong ベンチマークを基礎的な通信性能の評価に，Allreduce-like ベンチマークをパイプライン通信の測定に用いる．通信と演算のパイプライン化のコンセプトと実装の詳細を示し，また，それが最新の FPGA 環境で実現できることを示す．また，既存研究との性能比較を行い，CIRCUS が高い性能を有していることを示す．本研究は次世代の高性能計算に向けた新しい手法を提案するものであり，アプリケーションが持つ大量のバルク並列度に依存せず高性能を達成するものである．

本論文の貢献は以下のとおりである．

- OpenCL との親和性の高い FPGA 間通信システムを提案する．HPC ユーザが計算科学アプリケーションを高速なネットワークで接続された複数の FPGA 上で実行することを可能とする．
- 我々の提案手法は通信と計算が一体となったパイプラインを高位合成言語から作成できる．パイプラインの効果により通信と演算が同時に行えるようになる．本論文では，実装の詳細をソフトウェアとハードウェアの両面から述べる．
- Pingpong ベンチマークを用いて，異なるノードにある FPGA 間通信のレイテンシとスループットを評価する．そして，Allreduce ベンチマークの結果から，通信と演算が一体となったパイプラインを構築でき，それを OpenCL から制御できることを示す．
- 既存の実装との性能評価を行い，CIRCUS の通信性能が他の実装よりも高い性能が得られることを示す．

2. 関連研究

OpenCL を FPGA で用いてアプリケーションやベンチマークの性能評価を行った論文はいくつか報告されている．文献 [3] で，彼らは Maxwell 方程式の解を求めるプログラムを OpenCL を用いて FPGA に実装した．このアプリケーションは非構造格子を用いるため，メモリアクセスが規則的な格子を用いる場合よりも複雑になる．FPGA 向けのメモリアクセス最適化を適用し，マルチスレッド化された CPU 実装とくらべて，約 2 倍の性能を達成した．FPGA の絶対性能は GPU など他のアクセラレータと比べると低く，どのような種類の処理を FPGA にオフロードするのかが重要となる．複数の FPGA をネットワークで接続して利用する研究はいくつか知られている．文献 [4] では，データセンター内に 6 × 8 の 2D トーラスネットワークを持つ FPGA クラスタを構築し，Bing Web 検索エンジンのアクセラレータとして用いている．

Paderborn University の Paderborn Center for Parallel Computing は Noctua supercomputer [5] を運用している．そのうちの 16 ノードは FPGA を有しており，ノードあたり Intel Stratix 10 FPGA を 2 枚持つ．それらの FPGA は光スイッチを経由して接続されており最大で 40 Gbps の FPGA 間通信ができる．また，光スイッチを用いている

ため，アプリケーションの要求に応じて，FPGA 間の接続関係を自由に変更できることがシステムの特徴である．De Matteis らは Streaming Message Interface（SMI）[6] を Noctua 上で開発し評価を行った．SMI は，本論文で我々が提案するシステムに近いシステムであり，OpenCL 環境からストリーミング通信や集団通信できる．SMI は通信機能のほとんどを OpenCL で実装しているが，我々の実装では性能に重要なモジュールは Verilog HDL で記述している．

本研究の独自性は，高位合成が持つ高い生産性と，FPGA が持つ高い通信能力をあわせて利用し，並列計算を行うことである．また，提案手法と既存手法である SMI との性能比較を行う．本章で述べたように，FPGA 上の高位合成に関する研究や，FPGA ネットワークに関する研究はさかんに行われているが，それらを組み合わせた研究は十分なされているとはいえない．

3. CIRCUS

3.1 Intel FPGA SDK for OpenCL

本研究では，Intel が提供している Intel FPGA SDK for OpenCL という高位合成開発環境を用いる．SDK は FPGA の回路を合成するコンパイラだけでなく，ホスト上で動作するランタイムライブラリやカーネルドライバも含んでおり，この SDK を使うだけで FPGA システムを構築できるものである．

図 1 に OpenCL SDK を使う際の FPGA 回路の概要を示す．FPGA 回路は 2 つの部分からなる．1 つは，Board Support Package（BSP）から生成される部分で，もう 1 つはコンパイル対象の OpenCL コードから生成される部分である．

BSP は本 SDK に固有の要素であり，OpenCL 標準の要素ではない．OpenCL コンパイラは，同じ OpenCL コードから複数のボードに対応した回路を生成するために BSP を用いる．OpenCL 環境をサポートとした FPGA ボードは多岐にわたるが，それらのボードはそれぞれ異なっている．たとえば，ボードに搭載されている FPGA チップが異なる場合がある，搭載されているメモリの仕様が異なる場合がある，などの違いがある．したがって，FPGA ボード固有の情報をコンパイラに伝える必要があり，BSP はそれらボード固有の情報を含んでいる．BSP には，外部 IO のためにペリフェラルコントローラ，たとえば，PCIe コントローラやメモリコントローラが含まれる．

Intel FPGA SDK for OpenCL は FPGA 固有の拡張を OpenCL に加えている．“Channel” は拡張の 1 つであり，UNIX における pipe に近いものである．Channel を使うことで同じ FPGA 内の 2 つの OpenCL カーネル間で直接データを交換できる．さらに，Channel には “I/O Channel” と呼ばれる利用形態があり，これは，OpenCL カーネルと BSP の間を接続する Channel で，OpenCL カーネルから外部ペリフェラルを制御するために用いられる．

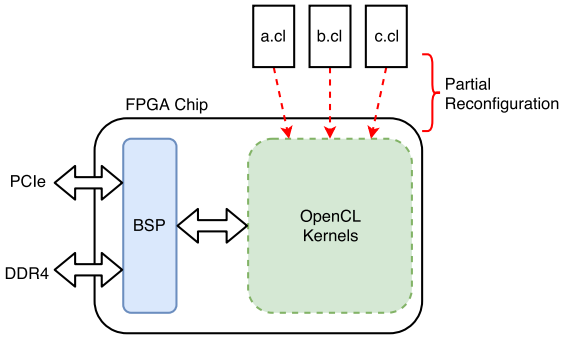


図 1　Intel FPGA SDK for OpenCL を利用する FPGA の内部構造

Fig. 1　Internal structure of an FPGA when we use Intel FPGA SDK for OpenCL.

3.2 パイプライン通信

我々は OpenCL から扱える FPGA 間高速通信システム CIRCUS の開発を行っている．CIRCUS システムの基本的なコンセプトは，パイプライン通信であり，Channel が持つ FPGA 内パイプライン通信を FPGA 間に拡張するものである．

Intel FPGA SDK for OpenCL では，OpenCL コードのループ構造に基づいてパイプラインが構築される．**図 2** のコードをコンパイルした際に構築されるパイプライン構造を**図 3** に示す．この図は，OpenCL コンパイラが生成するレポートに含まれる Graph Viewer を用いて生成した．このコードには，メモリから値を読み出かカーネルと，その値に 1 を加えメモリに書き込む 2 つのカーネルがあり，2 つが Channel で接続されている．ここで，LD，ST，RD，WR はそれぞれメモリからの読み出し，メモリへの書き込み，Channel への書き込み，Channel からの読み出しを表す．それぞれのループは異なるカーネルとして実装されているため，非同期に動作することになるが，Channel によるデータの依存関係があるため，全体として大きな 1 つのパイプラインを構築できる．

このコード例では，通常の Channel を用いているため，このまま異なる FPGA 間での Channel 通信はできない．そこで，CIRCUS は，**図 4** に示すようなパイプライン通信を実現する機構を OpenCL 環境に組み込むことで，異なる FPGA 間での Channel 通信を可能とするものである．CIRCUS が構築する通信パイプラインの詳細は次章で述べるが，OpenCL で書かれた通信用カーネルと Verilog HDL で書かれた通信モジュールを Channel で組み合わせることで本機能を実現する．したがって，OpenCL コンパイラが作成する演算パイプラインと，CIRCUS が提供する通信パイプラインを組み合せると，通信と演算の両方を融合した

```
#pragma OPENCL EXTENSION cl_intel_channels :
    enable
channel int c;

__kernel void sender(__global int* p, int n) {
  for (int i = 0; i < n; i++) {
    int v = p[i];
    write_channel_intel(c, v);
  }
}

__kernel void receiver(__global int* p, int n) {
  for (int i = 0; i < n; i++) {
    int v = read_channel_intel(c);
    v = v + 1;
    p[i] = v;
  }
}
```

図 2　Channel を用いた OpenCL コードの例
Fig. 2　Example OpenCL code using a channel.

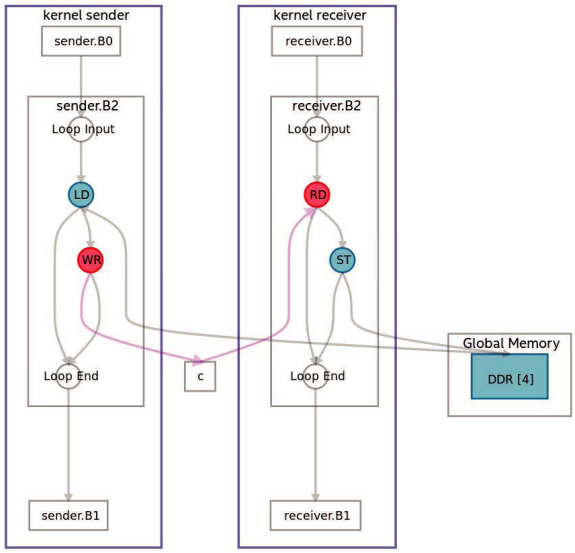


図 3　OpenCL コンパイラが生成するパイプラインの構造
Fig. 3　Pipeline structure generated by the OpenCL compiler.

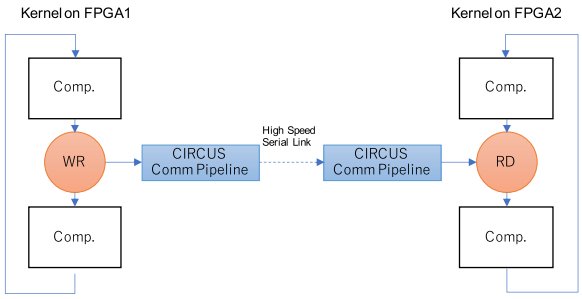


図 4　CIRCUS が構築するパイプラインの構造
Fig. 4　Pipeline structure what CIRCUS makes.

パイプラインを構築できる．本論文では，そのようなハードウェア構造を「演算と通信のパイプライン」と呼ぶ．

高性能計算のアプリケーションでは，通信時間を隠蔽するために，通信と演算をオーバラップする最適化が用いられる．特に，ステンシル計算では一般的な最適化の 1 つとして用いられており，袖領域のデータを必要としない計算と袖領域の通信をオーバラップし通信隠蔽を行う [7]．たとえば，オーバラップを MPI を用いて実装する場合，アプリケーションコードで明示的に記述が必要になる．通信でブロッキングされることを回避するために，MPI_Isend や MPI_Recv などの非同期通信を利用する必要がある．それに加えて，通信と演算の依存関係を考慮して，非同期な通信を行っても正しく計算できるようにしなければならない．

一方で，CIRCUS を用いる場合，演算と通信はパイプラインに変換され FPGA に実装される．いい換えると，クロックサイクルレベルで細粒度に通信と演算がオーバラップされているといえる．これは，それぞれのカーネルで時間方向にパイプライン，1 つの FPGA 内で複数のカーネルが同時に動く空間方向へのパイプライン，それに加えて複数の FPGA をネットワークで接続することによる空間方向への（並列）パイプラインを実現するものである．この特徴は他のアクセラレータにはなく FPGA 特有のものであり，我々は FPGA 上で演算と通信を融合させることで，HPC アプリケーションのさらなる演算加速が可能であると考えている．

表 1　各実装の比較．“A10” は Arria10 FPGA，“S10” は Stratix10 FPGA，“Bitt” は Bittware の Proprietary な通信プロトコルをそれぞれ意味する
Table 1　Comparison among the implementations. “A10”, “S10” and “Bitt” represent Arria10 FPGA, Stratix10 FPGA and the Bittware’s proprietary protocol, respectively.

	CIRCUS	CoE [8]	SMI [6]
Board	520N (H-Tile)	A10PL4	520N (L-Tile)
FPGA	S10	A10	S10
Protocol	SerialLite III	Ethernet	Bitt
Peak BW.	100 Gbps	40 Gbps	40 Gbps
Switch	No	Yes	No

3.3　既存研究との違い

本節では，CIRCUS と既存研究との比較を行う．CoE [8] は我々がこれまで開発してきたシステムであり，CIRCUS の前身にあたる．Streaming Message Interface（SMI）[6] は 2 章で述べた ETHZ で開発されているシステムである．また，**表 1** はそれぞれの通信機構の比較を示した表である．

CoE では，演算と通信のパイプラインというコンセプトは一部しか実現できていなかった．Ethernet プロトコルを用いることに由来する制限があったからである．Ethernet プロトコルはパケットベースのプロトコルであり，store-and-forward のスイッチングが利用されるため，このような用途には適さない．また，通信効率を高めるために長いパケットを作らなければならず，バッファリングによる通信

レイテンシの増加も存在する．非効率であっても，CoE が Ethernet を採用した理由は，Arria 10 FPGA を搭載したボードは，外部通信ポートが 2 ポートしかないものが一般的であり，外部スイッチを用いなければ多数の FPGA を接続できなかったからである．CIRCUS では，peer-to-peer の Streaming プロトコルを用いることで，この問題を解決する．Stratix 10 FPGA を搭載ボードにおいては，4 つの外部ポートを持つものが販売されており，外部スイッチを使わずともスケーラビリティを確保できると判断したため，プロトコルの変更を行う．

SMI は，名前が示すとおり Streaming 通信に基づく実装であり，CIRCUS に近いコンセプトで研究開発されているといえる．SMI は Bittware 社（ボードベンダ）が開発したプロプライエタリな peer-to-peer プロトコルを用いて通信を行う．そのため，演算と通信のパイプラインが実現できているといえる．しかしながら，SMI は 40 Gbps までの通信にとどまっている．本研究では 1 章で述べたように，より高速な 100 Gbps の通信を用いた並列システムを構築する．また，性能比較の項目でで述べるが，SMI は 40 Gbps の環境に最適化されており，高速な 100 Gbps の通信を十分に扱うことができない．

4. 通信の実装と利用方法

4.1 システムの概要

まず，**図 5** に通信システム全体の構造を示す．CIRCUS システムは，ルータモジュールと OpenCL で書かれた制御カーネルから構成される．そして，それらのコンポーネント間は Channel もしくは I/O Channel で接続される．また，ルータモジュールはさらに低レイヤの通信 IP に接続され，高速シリアル通信を通じて他の FPGA と接続される．これ以降，他の目的で使われている Channel と区別するために，図 5 にあるように CIRCUS システムとユーザが記述したアプリケーションカーネルを接続する Channel を “CIRCUS channels” と呼称する．本節では，概要を示す留め，個々の要素の詳細については，後節で詳細を述べる．

図 6 は CIRCUS を使う簡単なコード例である．このコードのは 2 つの CIRCUS Channel（“simple_out” と “simple_in”）が存在するが，これらのカーネルは別々の FPGA で動作しているものとする．カーネルが別々の FPGA 上で動作しいるため，通常の OpenCL コードであれば Chnenel を通じて通信することはできない．CIRCUS は，この 2 つの Channel をネットワークを通じて仮想的に接続する機能を実現するものである．

4.2 コード生成

アプリケーションに CIRCUS を適用する場合，通信に対する要求，たとえば CIRCUS channel の数やデータ型はアプリケーションによって異なることは明らかである．したがって，我々はコードジェネレータを開発し，CIRCUS に関するコードを通信定義ファイル（XML ファイル）から自動生成させる設計とした．定義ファイルにはアプリケーションが必要とする CIRCUS Channel の定義情報が含まれており，それをもとにして，コードジェネレータは，図 5

sender code on FPGA1

```
__kernel void sender(__global float* restrict x, int n) {
    for (int i = 0; i < n; i++) {
        float v = x[i];
        write_channel_intel(simple_out, v);
    }
}
```

receiver code on FPGA2

```
__kernel void receiver(__global float* restrict x, int n) {
    for (int i = 0; i < n; i++) {
        float v = read_channel_intel(simple_in);
        x[i] = v;
    }
}
```

図 6　CIRCUS を用いた通信コード例

Fig. 6　Example code of communication using CIRCUS.

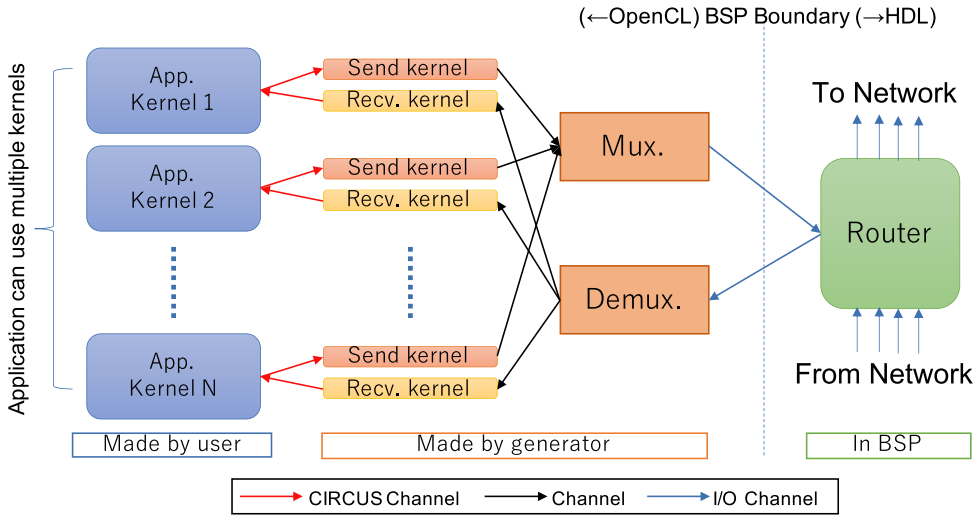


図 5　CIRCUS 通信システムの概要

Fig. 5　Overview of the CIRCUS communication system.

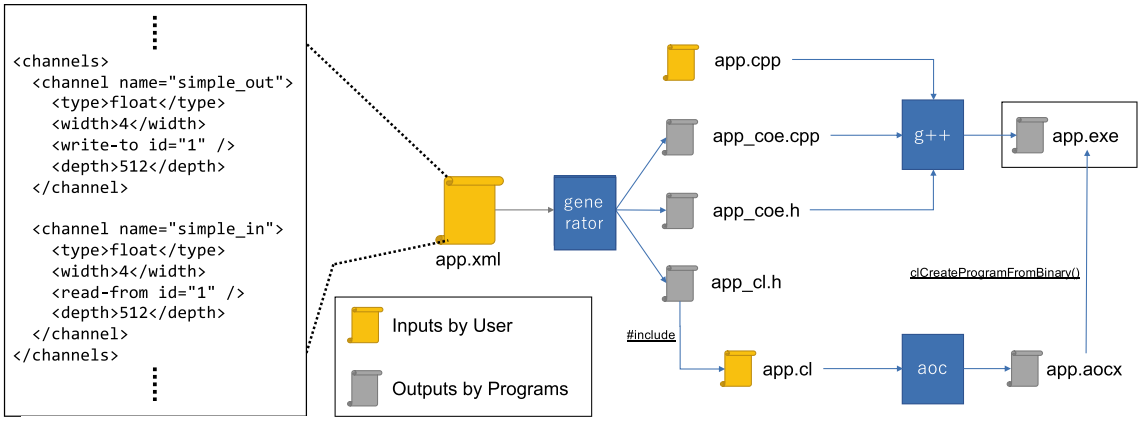


図 7　CIRCUS システムの開発フロー
Fig. 7　Development flow with CIRCUS.

で “By Generator” で示されている範囲のカーネルコードを自動生成する．また，自動生成したカネールを制御するためのホストコードも同時に生成する．

図 7 は CIRCUS ジェネレータを使う際の開発フローを示す．この XML ファイルは図 6 にあるコードを生成するときに用いるものである．図に含まれているファイルの意味は以下のとおりである．

app.xml:
CIRCUS channel の定義ファイル．

app_host.cpp, app_host.h:
CIRCUS ランタイム（host code）．

app_cl.h:
CIRCUS カーネル（device code）．

app.cpp:
CIRCUS を使うアプリケーション（host code）．

app.cl:
CIRCUS を使うアプリケーション（device code）．

“app.xml” はアプリケーション固有の CIRCUS 通信構造を定義するファイルである．図 7 に XML ファイルの一部を示す．“channel” タグで，CIRCUS Channel を定義する．“name” 属性は CIRCUS channel の名称を指定し，“type” タグは CIRCUS Channel の OpenCL における型を指定する．“read-from”/“write-to” で通信先の Channel の ID を指定する．MPI 通信における tag に相当する．“depth” タグは CIRCUS Channel のバッファのサイズを示し，“width” タグは CIRCUS channel の幅をビット単位で指定する．本来，width は type から推論できるパラメータであるが，現在のジェネレータ実装は OpenCL のパーザを持たないため，明示的に与える必要がある．

この構成では，アプリケーションがいくつのカネールを持っていても，また各カーネルが何個 CIRCUS Channel を利用するとしても，OpenCL カーネル–BSP 間の I/O Channel の構造は変化しない．そのため，アプリケーションが変化しても BSP 部は同じものを利用でき，Verilog HDL で記述された部分の変更を必要としない．CIRCUS は，アプリケーションに依存して変化する部分を OpenCL で記述し，性能が重要なルータ部を Verilog HDL で記述して組み合わせることで，開発コストと性能のバランスを保つ．

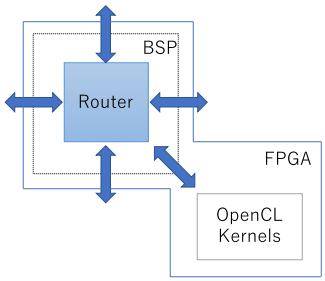


図 8　ルータとカーネルの接続関係．青矢印は 256 bit 幅の全二重バスを表す
Fig. 8　Connection between the router and the kernel. A blue arrow represents a 256 bit-width full-duplex bus.

4.3　パケットルーティング

本節では，どのように CIRCUS パケットが宛先 FPGA に配送されるかについて述べる．CIRCUS では Intel が提供する SerialLite III 通信 IP を用いて FPGA 間を接続する．しかしながら，SerialLite III は 2 つの FPGA 間を接続するための実装であるため，隣接 FPGA 間でしか通信ができない．我々はルータ機構を BSP に組み込むことで，この問題を解決する．

図 8 にルータの構造を示す．青矢印はそれぞれ 384 MHz で駆動される 256 bits 幅の全二重のバスを示す（$0.384 \times 256 = 98.3$ Gbps）．図 9 にルータの内部ダイアグラムを示す．ルータは 5 入力ポート，5 出力ポート，クロスバー，ルーティングテーブルから構築される．図の “Tbl” はルーティングテーブルを参照するモジュールを意味し，“Arbiter” はルータの入出力を調停し，パケットが衝突しないように調整を行う．ルータの動作周波数は通信

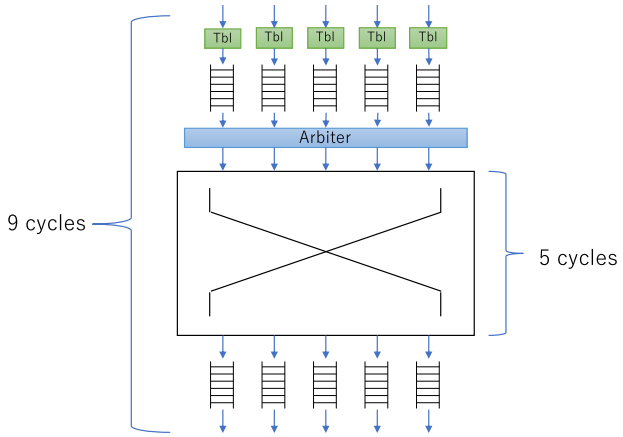


図 9　ルータのダイアグラム．“Tbl” はルーティングテーブルを参照するモジュールを表す

Fig. 9　Diagram of the router. “Tbl” represents a module for routing table lookup.

層の物理層（SerialLite III）の動作周波数にリンクしており，バスと同じく 384 MHz で動作する．ModelSim（RTL Simulator）を用いたレイテンシの測定では，入力から出力までの全体のレイテンシは 9 クロックサイクル（～24 ns）で，クロスバー本体のレイテンシは 5 クロックサイクル（～13 ns）であった．

CIRCUS パケットは，FPGA ノード ID と CIRCUS Channel ID の 2 つの宛先情報を持ち，この 2 つの情報を組み合わせて通信の宛先を決定する．CIRCUS パケットは 6 ビット幅の宛先フィールドをヘッダのなかに持っている．パケットがルータに到着すると，ヘッダのなかの宛先を元にルーティングテーブルを参照し，どのポートにパケットを送り出すかを決定する．また，CIRCUS Channel には，1 つ 1 つに固有の ID があり，パケット内にノード ID だけでなく Channel ID を含めることで，どこに配送するかを指定する．

4.1 節で図 5 に CIRCUS システムの全体図を示したが，ここで，Send Kernel はパケットを生成するカーネル，Recv Kernel はパケットからペイロードデータを取り出すカーネル，Multiplexer は複数の入力をまとめて 1 つのストリームにするカーネル，Demultiplexer は逆に 1 つの入力を分解し複数のストリームに変換するカーネル，Router はルータモジュールである．送信側 FPGA にある Send Kernel で，宛先情報を含むパケットを生成し，Router が FPGA 間のパケット転送を行う．そして，受信側 FPGA にある Demultiplexer が Channel ID をチェックし，対応する Recv Kernel にパケットを配送することで CIRCUS Channel 間通信を実現する．

4.4　通信可能な BSP

OpenCL 環境から，BSP を通じて外部通信機構を制御可能なことは一般的ではなく，外部通信機構を利用するときは BSP に制御回路や制御 IP を追加実装しなければならない．

我々は文献 [9] で，どのように OpenCL BSP に通信機構サポートを追加するかを示した．40 Gb Ethernet コントローラ IP を BSP に追加し，どのようにして OpenCL カーネルから制御するかを示した．本論文で用いる通信が可能な BSP は，これら論文の実装を元にしたものである．そのため，本論文ではどのようにして BSP を改造するかの詳細は触れない．詳細は上述した論文を参照していただきたい．

4.5　制限事項

CIRCUS システムは開発中であるため，通信機能に関していくつかの制限事項がある．

- エラー処理（再送）・フローコントロールが未実装
- Virtual Channel（VC）が未実装

再送などのエラー処理が未実装なため，通信エラー発生時はデータの破損・消失が発生する．フローコントロールが未実装なため，通信経路上でバッファ溢れが発生するとデータが消失する．また，VC を実装していないため，トーラスなど通信路に循環があるとデッドロックが発生しうる．ただし，デッドロックが発生するのは循環がある場合のみであるため，メッシュ状のネットワークで運用する際はデッドロックしない．これらの問題は，CIRCUS を引き続き開発することで解決していきたいと考えている．

5.　評価環境

5.1　Cygnus および PPX

本論文では，筑波大学計算科学研究センターで運用中の Cygnus スーパコンピュータと，同センターが保有する実験クラスタである Pre PACS Version.X（PPX）クラスタを性能評価に用いる．Cygnus は実運用に供されているシステムであるため，OS の再起動を要する FPGA の書き換えに制限がともなう．そのため，多数の FPGA を必要で性能に関する評価は Cygnus 上で実施し，それ以外の補助的な試験には PPX を用いる使い分けを行うこととした．通常，OpenCL を実行する際は部分再構成（Partial Reconfiguration）と呼ばれる技術を用いて，OpenCL カーネルに相当する回路のみを変更でき，PCIe のコントローラーは動作し続ける．ホスト OS から見た際に，正しく動作し続ける PCIe デバイスであり続け，ホスト OS の再起動は必要でない．一方，CIRCUS は BSP を改造しているため，実行する際には FPGA 全体の書き換えが必要であり，その場合，FPGA は PCIe デバイスとしての動作を停止するため，ホスト OS の再起動が必要である．

Cygnus はマルチ・ヘテロジニアスなシステムであり 80 ノードから構成される．そのうち，32 ノードは Albireo ノードと呼ばれ，CPU + GPU + FPGA ノードである．

表 2 にあるように，Albireo は 2 × Intel Xeon CPU，4 × NVIDIA V100 GPU，4 × InfiniBand HDR100 HCA，2 × Bittware（旧 Nallatech）520N FPGA ボードから構成される．520N ボードは Intel Stratix10 FPGA，32 GB DDR4 メモリ，最大 100 Gbps をサポートする QSFP28 ポートを 4 つ持つ．Cygnus システムには，トータルで 64 枚の FPGA がある（32 Albireo nodes × 2 FPGAs/node）．それらの FPGA が 8 × 8 の FPGA 専用 2D トーラスネットワークを構築している．同時に InfiniBand ネットワークも構築されているため，従来の CPU や GPU アプリケーションは InfiniBand ネットワークを独立して利用できる．

PPX は Cygnus を含む次世代 PACS シリーズスーパーコンピュータのためのテストベッドシステムである．**表 3** に本論文で用いる PPX ノードの仕様を示す．CPU の仕様や InfiniBand の仕様が Cygnus とは異なるが，FPGA ボードと FPGA 開発環境は Cygnus と同一のものを用いている．

表 2　評価環境（Cygnus）
Table 2　Evaluation environment (Cygnus).

CPU	Intel Xeon Gold 6126 × 2
CPU Memory	DDR4 192 GB (96 GB / CPU)
Infiniband	Mellanox ConnectX-6 HDR100 ×4
Host OS	CentOS 7.6
Host Compiler	gcc 4.8.5
OpenCL SDK	Intel FPGA SDK for OpenCL 19.1.0.240
FPGA	Bittware 520N (1SG280HN2F43E2VG)
FPGA Memory	DDR4 2400 MHz 32 GB (8 GB × 4)
Comm. Port	QSFP28 × 4
Comm. Cable	Mellanox MFA1A00-E005, MFA1A00-E010

表 3　評価環境（PPX）
Table 3　Evaluation environment (PPX).

CPU	Intel Xeon E5-2690 v4 × 2
CPU Memory	DDR4 64 GB (32 GB / CPU)
Infiniband	Mellanox ConnectX-4 EDR ×1
Host OS	CentOS 7.3
Host Compiler	gcc 4.8.5
OpenCL SDK	Intel FPGA SDK for OpenCL 19.1.0.240
FPGA	Bittware 520N (1SG280HN2F43E2VG)
FPGA Memory	DDR4 2400 MHz 32 GB (8 GB × 4)
Comm. Port	QSFP28 × 4
Comm. Cable	Mellanox MFA1A00-C010

5.2　性能比較環境

本論文では，CIRCUS の通信性能を既存の実装との比較を行う．比較対象は，我々の既存研究である CoE [8] と ETHZ で研究開発されている SMI [6] である．

文献 [6] における SMI の評価環境は Bittware 520N L-Tile Board を用いているため，最大で 40 Gbps までしか通信できない．そこで，オープンソースの SMI 実装 [10] を PPX（520N H-Tile Board）を用いて評価したデータを “SMI-HTile” とする．ここで，L-Tile，H-Tile と区別しているものは，通信トランシーバのダイのバージョンの違いを示す．Intel Stratix 10 FPGA は Multi Chip Module（MCM）構造をしており，FPGA のコアロジックが実装されたダイと，通信トランシーバが実装されたダイが 1 つのパッケージに封入されている．そのため，FPGA ロジックの仕様は同じだが，通信トランシーバダイのバージョンが違うという製品が存在する．520N L-Tile ボードは最大で 40 Gbps の通信に制限されるが，520N H-Tile ボードは最大で 100 Gbps を扱える．

6.　性能評価

6.1　Pingpong Benchmark

6.1.1　概要

CIRCUS システムの基本的な性能を測定するために Pingpong ベンチマークを用いる．我々はベンチマークを **図 10** にあるように FPGA 上に実装した．2 つの異なる FPGA がネットワークを通じてデータを交換し，OpenCL カーネル上で通信に要した時間を測定する．したがって，このベンチマークは単なる通信時間だけでなく，CIRCUS バックエンドで発生した OpenCL オーバヘッドも含む．

Pingpong における通信時間計測は**図 11** に示すように行う．通信時間は次の式で定義するものを使用する：$(t_1 - t_0) \times 0.5 + (t_2 - t_1)$ [s]．スループットは通信メッセージ長を通信時間で除したものであり，レイテンシは片

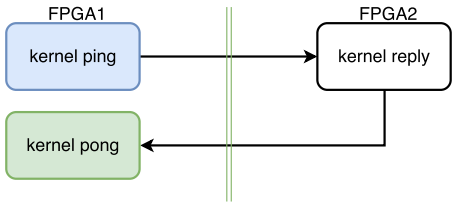


図 10　Pingpong ベンチマークのデータの流れ
Fig. 10　Dataflow of the pingpong benchmark.

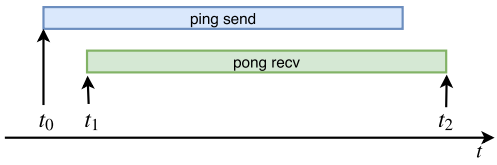


図 11　Pingpong ベンチマークのタイムライン
Fig. 11　Timeline of the pingpong benchmark.

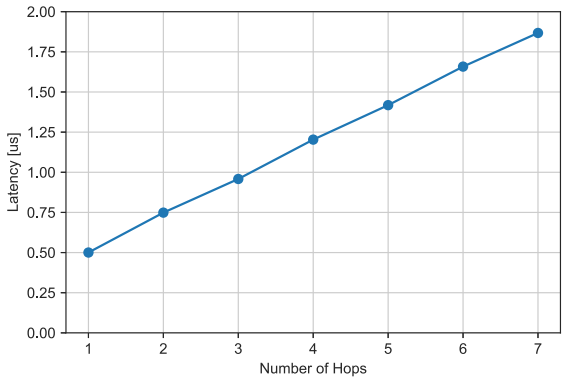


図 12　Pingpong ベンチマークの結果（レイテンシ）
Fig. 12　Result of the pingpong benchmark (Latency).

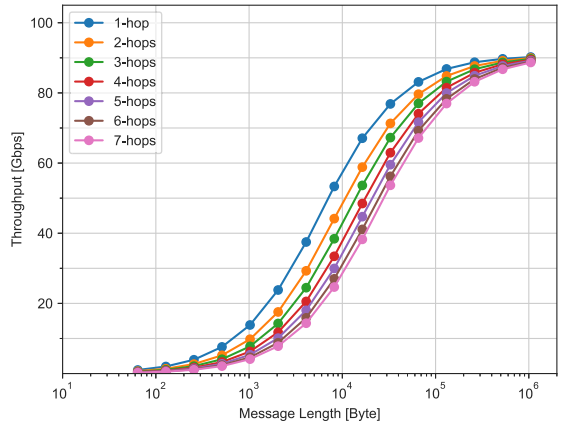


図 13　Pingpong ベンチマークの結果（スループット）
Fig. 13　Result of the pingpong benchmark (Throughput).

道の通信レイテンシを求めるために，最初のデータを送信してから，折り返しの最初のデータが到着するまでの時間を 2 で割ったもの（$(t_1 - t_0) \times 0.5$）である．実験では，メッセージ長を変えて，1 つのメッセージの通信を行い時間を測定する．その測定実験を独立して 10 回行い，全体の通信時間が最も高速だったデータを結果として採用した．これらの時間測定は同じ FPGA が（送信側）で行われるため，クロックサイクルカウンタを用いて通信時間を測定した．OpenCL カーネルの動作周波数はコードごとに可変であるが，Stratix10 デバイス使用時に，一般的に 250 MHz〜300 MHz の周波数が得られるため，十分な時間測定の精度があると考えられる．

6.1.2　測定結果

Pingpong ベンチマークの結果を**図 12** と**図 13** に示す．なお，本実験には Cygnus を用いた．メッセージ長 64 Byte〜1 MB，1 ホップから 7 ホップまでの性能を測定する．ここで，“ホップ” は送信側 FPGA から受信側 FPGA までの距離を指す．たとえば，“1-hop” は隣接する FPGA 間通信，“2-hops” は隣の隣との FPGA 間通信を指す．

このベンチマークは uint16 型（512 bits width）の CIRCUS Channel を使い，動作周波数は 295.8 MHz である．最小レイテンシと最大レイテンシは，それぞれ 0.5 μs and 1.87 μs である．また，1 ホップあたりの増加レイテンシは約 0.25 μs であった．通信スループットは 1-hop 時に 90.2 Gbps，7-hops 時に 88.7 Gps が得られた．hop 数を増やすと，特に小〜中メッセージ長でスループット低下がみられる．これは，通信レイテンシ（$t_1 - t_0$）が増加するため，全体の通信時間に対するレイテンシの割合が大きくなるためである．ただし，これらの通信性能は OpenCL カーネルの周波数に依存する部分があるため，すべてのアプリケーションでこの値が得られることを保証するものではない．

表 4　プロトコルから発生する通信オーバヘッド
Table 4　Overheads from the protocols.

Factor	Throughput of payload	Efficiency
Physical Speed	103.125 Gbps	
67b/64b	98.484 Gbps	×0.955
Meta Frame	98.287 Gbps	×0.998
SerialLite III Burst	96.813 Gbps	×0.985
CIRCUS Header	90.762 Gbps	×0.938

6.1.3　考察

Pingpong ベンチマークの結果では，90.2 Gbps の性能を達成した．100 Gbps の速度を物理層に使っているにもかかわらず 9 割の性能しか得られていないが，この性能は設計どおりのものが得られている．**表 4** に利用しているプロトコルから発生するオーバヘッドを示す．まず，本論文で用いる実装は，物理層に 103.125 Gbps（$= 4 \times 25.78125$）の速度を用いている．この速度は 100 Gbit Ethernet と同じものである．“67b/64b”，“Meta Frame”，“SerialLite III Burst” の項目は SerialLite III プロトコルに由来するオーバヘッドを示す．これらの値は SerialLite III のドキュメント [11] から求めた．“CIRCUS Header” は CIRCUS システムが利用するヘッダに由来するオーバヘッドを示す．CIRCUS のパケットは 64 バイトであり，そのなかで 4 バイトがヘッダ，60 バイトがペイロードである．

最も影響が大きいオーバヘッドは “CIRCUS Header” であるが，これはショートパケットを採用した結果である．我々は CIRCUS の通信網を低レイテンシ通信に最適化を行った．ショートパケットの通信網はバッファリングに要する処理時間やリソース使用量を削減できるため，パイプライン通信に重要な要素であると考えている．我々は将来的に次世代の FPGA を使うことでこの問題を緩和できると考えている．たとえば，Intel Stratix10 E-Tile Transceiver [12] では，Pulse Amplitude Modulation-4（PAM-4）を用いた通信が利用でき，帯域を倍にできる．

Pingpong ベンチマークの結果から，1 ホップにかかる追加のレイテンシは約 250 ns と分かった．ルータは Verilog HDL で実装しているため，ルータにかかるレイテンシは詳細に求めることができる．ルータのレイテンシは 23.427 ns

(9 clock cycles) であった．したがって，光ケーブルを通じて SerialLite III プロトコルを用いて通信するのに要する時間は，およそ 225 ns と分かる．このレイテンシは ASIC で製造されたネットワークと比べると高い．たとえば，Cray Gemini Interconnection Network では，ノード間のレイテンシが 105 ns [13] とされており，これは我々の実装のおよそ半分である．しかしながら，FPGA を用いた再構成可能なネットワークには演算モジュールを通信モジュールに直接接続できるという利点があり，両者を合わせた性能はそれらが分離されたネットワークよりを上回れると考えている．

6.2　性能比較

図 14 に 4 つの実装の性能比較の結果を示す．また，**表 5** にレイテンシの比較を示す．それぞれの実装で “1-hop” に相当する性能を比較した．CoE は Ethernet プロトコルを使用しているため，スイッチ 1 台を経由した通信を “1-hop” とする．また，SMI の性能評価には，Github レポジトリに含まれているベンチマークを Eager プロトコルの設定で実行を行った．ただし，我々は，SMI の論文で示されたデータの値を持たないため，文章として記されていたピーク性能のみを図 14 に記述した．SMI のヘッダ（4 バイト）を含んで 35 Gbps の性能を達成したと記述されていたため，これをペイロードのみの性能に変換し 30.625 Gbps がピーク性能だと定義した．

CoE は Ethernet プロトコルとスイッチを用いているため，CIRCUS と直接比較するのは妥当ではないと考えられるが，CIRCUS は 3 倍高いスループットと 4 倍低いレイテンシを達成した．しかしながら，CIRCUS は直接網であり CoE はスイッチ網であることから，長距離の通信は CIRCUS の方がレイテンシが高くなる．CIRCUS の 3 ホップが CoE の 1 ホップとほぼ同じレイテンシとなる．

CIRCUS と SMI を比較すると，CIRCUS は 3 倍高いスループットと，同程度の通信レイテンシを達成した．また，SMI は 100 Gbps の通信をサポートする H-Tile のボードで実行（SMI H-Tile）しても，文献 [6] で示されている性能と比べて大きな違いが見られなかった．このことより，我々は SMI は 40 Gbps の環境に最適化されており，それよりも高い通信性能があったとしても，それをうまく扱えないと考えている．

我々はこの性能の違いは，BSP の設計の問題と，ルータ部分の設計方針の違いから来ていると考えている．Bittware の BSP は 256 bit 幅の I/O Channel で BSP と OpenCL カーネル間を接続している．SMI ベンチマークの動作周波数は 266.67 MHz であり，したがって実装上のピーク性能は $266.67 \times 256 = 59.7$ Gbps に制限されており，100 Gbps の通信路を満たすことができない．なお，CIRCUS では，この問題を緩和するために 2 倍の 512 bit 幅の I/O Channel を用いて BSP と OpenCL カーネル間を接続している．CIRCUS のルータは Verilog HDL で記述したクロスバーによる接続を用いているが，SMI のルータは OpenCL で記述したラウンドロビン方式で実装されている．そのため，CIRCUS のルータの方が，性能が高いものの，開発コストが高くなっていると考えられる．この問題はパフォーマンスと開発コストのトレードオフの関係にあるが，一般的に，ルータの実装を変更する頻度は高くないため，許容範囲内であると考えている．一方で，SMI 実装の優れている点はフロー制御と誤り訂正を持つことである．実利用を考慮すると，これらの要素は重要であり，我々は引き続き CIRCUS システムの研究開発を進め，フロー制御とエラー処理（再送，もしくは訂正）を実装する予定である．

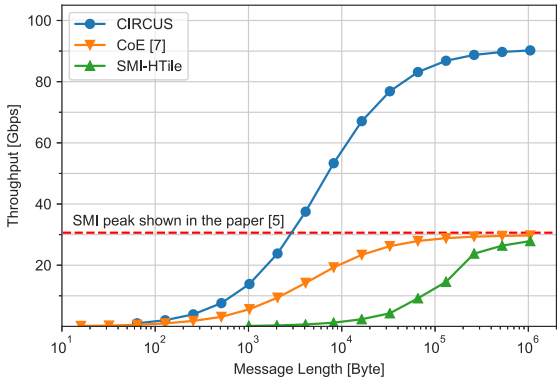


図 14　実装間の性能比較
Fig. 14　Performance comparison among the implementations.

表 5　各実装の最小レイテンシ
Table 5　The minimum latency of each implementation.

CIRCUS	CoE [8]	SMI [6]	SMI-HTile
500 ns	950 ns	801 ns	517 ns

6.3　パイプライン通信と演算

6.3.1　概要

CIRCUS を利用しているコードでは，OpenCL コンパイラは通信と演算を一体化したパイプラインを構築する．したがって，通信と演算はオーバラップされ，通信時間を完璧に隠蔽できる．このオーバラップの効果を測定するために，Allreduce-like なプログラムを作成し性能測定を行う．**図 15** にベンチマーク全体の構成を示す．このベンチマークは MPI_Allreduce 通信を 32-bit 整数（uint type in OpenCL），MPI_SUM で実行したときの動作を模したものである．右側のノードが root ノードのように振る舞う．Node A の入力データを Node B に送り，Node B では「自分の入力データ＋ Node A から来たデータ」の演算を行う，そして，その結果を Node A に送り返す．この構成は，Pingpong benchmark の 1-hop の状態に，整数加算という演算をパイプラインに追加したことに相当する．

それぞれの FPGA は 2 種類の OpenCL カーネルを実装している．1 つは送信・加算カーネル（図 15 の灰色の箱）であり，もう 1 つは受信カーネル（図 15 の緑色の箱）である．**図 16** に送信カーネルのソースコードの一部を示す．“read_channel_intel” と “write_channel_intel” は，それぞれ Channel から読み込む，Channel へ書き込む組み込み API である．“clocktime” 関数は，我々が実装した時間を測定するための関数である．OpenCL コンパイラはループからパイプラインを構築するため，ループ内のすべての行は並列に動作する（CPU とは異なる）ことに注意が必要である．

6.3.2　測定結果

図 17 に Cygnus 上で実行したベンチマークの測定結果を示す．最大スループットは 90.2 Gbps が得られ，この値は，Pingpong ベンチマークと同じものである．ただし，演算分のレイテンシが追加されているため，小サイズから中サイズにかけての性能は Pingpong ベンチマークの結果と比べて悪化している．

最小通信レイテンシは 1,067 ns が得られた．この通信レイテンシは，pinpong ベンチマークにおける 3-hops のケースに近い．図 13 における 3-hops のデータと得られている値が近いことからも，この結果は妥当であると考えられる．

6.3.3　考察

Allreduce-like ベンチマークで得られたスループット性能は Pingpong ベンチマークの性能と同じであった．追加した演算は整数演算という軽量なものであったが，通信レイテンシが増すため，小〜中メッセージでの性能低下が見られた．この結果は，通信と演算が一体となったパイプラインを正しく構築できたことを示すものである．

この結果を裏付けるために，実際の FPGA 上のデータの流れの計測を行う．Intel FPGA 開発環境には，FPGA 実機におけるデータを測定し，性能測定やデバッグなどに利用する Signal Tap [14] という組み込みロジック・アナライザツールがある．ベンチマークを実行した際の OpenCL カーネルと BSP 間にある I/O Channel のデータを測定した結果を**図 18** に示す．このデータは，32 KB のデータを通信した際に，図 15 の “Node B” 上で測定したものである．なお，測定環境は PPX クラスタを用いた．Signal Tap に関する回路を追加してコンパイルしているため，性能評

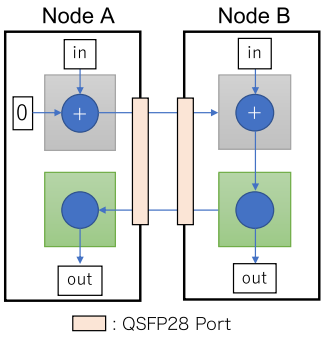


図 15　Allreduce-like ベンチマークのデータの流れ
Fig. 15　Dataflow of the allreduce-like benchmark.

```
for (int i = 0; i < n; i++) {
uint16 val = (uint16)(0);
if (!zero_in) val = read_channel_intel(fwd_in);
val += ...;
if (internal_out) {
  write_channel_intel(internal, clocktime(val, &
      t_tmp));
} else {
  write_channel_intel(fwd_out, clocktime(val, &
      t_tmp));
}
...
}
```

図 16　Allreduce-like ベンチマークのコードの一部
Fig. 16　Part of the Allreduce-like benchmark code.

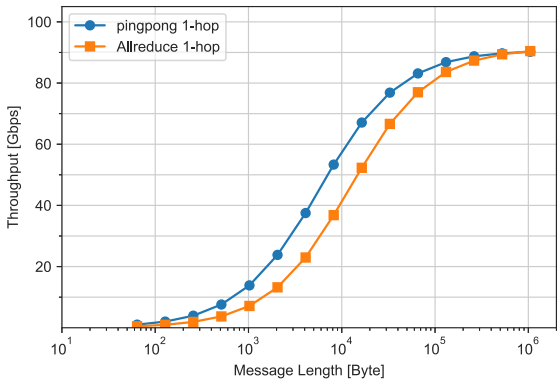


図 17　Allreduce ベンチマークの結果
Fig. 17　Result of the allreduce benchmark.

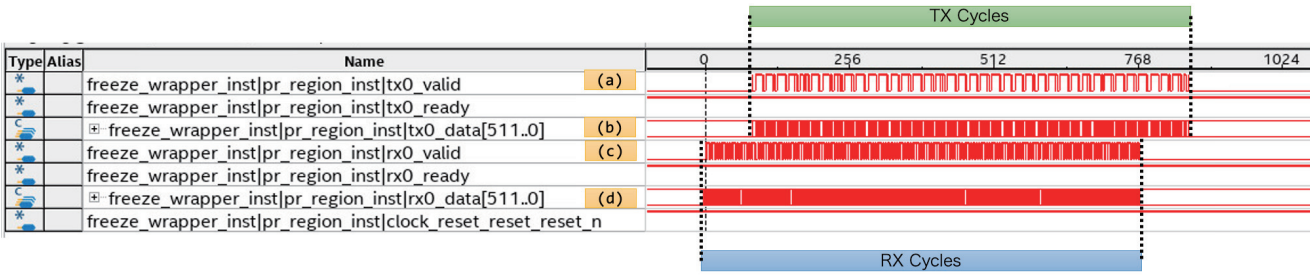


図 18　Signal Tap を用いて観測した I/O Channel の波形
Fig. 18　Waveform of the I/O Channel measured by Signal Tap.

価の章で用いた回路と異なっているが，ハードウェアの挙動としては同一である．ただし，動作周波数が 300 MHz から 265.6 MHz に変化しているため，通信レイテンシは増加していると考えられる．

(a) は送信データの valid 信号，(b) は送信データの中身，(c) は受信データの valid 信号，(d) は受信データの中身を表す．横軸の単位は OpenCL Kernel の Clock Cycle である．0 点は Signal Tap が記録を開始した時間を示し，図の右側にいくほど未来のデータを意味する．また，説明のために，受信している時間（RX Cycles）と送信している時間（TX Cycles）を示す箱を追記している．結果より，受信している時間と送信している時間がオーバラップしていることが分かる．したがって，通信と演算を融合したパイプラインが想定どおりに構築できることを示すものである．

7. リソース使用率

7.1 測定結果

Pingpong ベンチマークのリソース使用量を，モジュールの階層別に 2 つの表に分けて示す．これらの結果は，コンパイル結果出力ディレクトリの “top.fit.place.rpt” ファイルから抽出したものである．ALMs，Registers，M20Ks，DSPs はそれぞれ，Adaptive Logic Modules（ALMs），ALM 中の Flip-Flops，20 Kbits 分散 on-chip メモリ，整数乗算または浮動小数点数の積和演算に用いられている Digital Signal Processors（DSPs）を示す．

表 6 に BSP におけるリソース使用率の詳細を示す．“CIRCUS BSP”，“BSP Others”，“Kernels”，“Others” はそれぞれ，CIRCUS によって BSP に追加されたモジュール（SerialLite III IP とルータ），オリジナルの BSP から引き続き存在するモジュール，OpenCL カーネル由来のモジュール群，FPGA コンパイラによって自動的に追加されているその他のモジュール（主にデバッグなどに使われる）を示す．

表 7 に OpenCL カーネルにおけるリソース使用量の詳細を示す．ただし，Channel の実装に必要なリソースは宛先（読み出し側）のカーネルが実装されているモジュール内に記述されている．“Memory Network” はカーネルが外部メモリにアクセスするための配線を指し，“Pingpong” は Pingpong ベンチマークを実装しているカーネルを指す．

このプログラムには 2 CIRCUS Channel の送受信ペアが実装されている．共有しているリソースがあることと，コンパイラの最適化により，それぞれの CIRCUS Channel がいくつのリソースを使っているかを判断することは難しいが，平均して 1 ペアあたり 1.25%の ALM と 0.8%のレジスタを消費している．“CIRCUS Mux” は複数の送信 Channel の出力を 1 つに纏める（Multiplexer）カーネルである．本論文の実装では，クロックごとに入力にデータがあるかどうかを確認しているため，通信の性能は高くなるが，リソースの消費量は多い．

表 6　リソース使用量：BSP Modules
Table 6　Resource Utilization: BSP Modules.

	ALMs	Registers	M20Ks	DSPs
Design Total	21.5%	9.5%	5.6%	0%
CIRCUS BSP	3.3%	1.5%	1.6%	0%
BSP Others	12.2%	4.5%	2.6%	0%
Kernels	5.5%	3.5%	1.3%	0%
Others	0.5%	0.1%	0.1%	0%

表 7　リソース使用量：Kernels and Channels
Table 7　Resource Utilization: Kernels and Channels.

	ALMs	Registers	M20Ks	DSPs
CIRCUS Out ×2	1.0%	0.6%	0%	0%
CIRCUS In ×2	0.9%	0.6%	0%	0%
CIRCUS Mux.	0.4%	0.3%	0%	0%
CIRCUS Demux.	0.2%	0.1%	0%	0%
Memory Network	1.2%	0.8%	1.0%	0%
Pingpong	1.2%	0.8%	0.3%	0%
Others	0.6%	0.2%	0%	0%

7.1.1 考察

リソース消費量の解析では，CIRCUS は 3.3%のリソースを BSP に追加することが分かった．我々はこの消費量は FPGA 間通信を実現するためのコストとしては小さく，受け入れられるものであると考えている．BSP 内のリソース消費量は，OpenCL カーネルに依存しないため，どのアプリケーションに CIRCUS を適用しても変化しない．一方，CIRCUS channel（送信・受信）ペアは平均して 1.6%のリソースを消費する．加えて，CIRCUS Channel の実装数はアプリケーションごとに異なり，Pingpong ベンチマークよりも多くなることが考えられる．

OpenCL カーネルで消費するリソースは最適化の余地があると考えており，最適化手法の 1 つに Verilog HDL で最適化を行うことがあげられる．CPU におけるインラインアセンブラのように，OpenCL コードの中に Verilog HDL で記述したモジュールを組み込むことができるため，OpenCL よりも詳細な最適化を行える．このような実装をしたとしても，アプリケーションを記述するユーザは OpenCL コードのみ扱えばよく，CIRCUS を用いた通信を利用できる．

7.2 リソース消費量比較

本節では，同じ FPGA ボードを用いている CIRCUS と SMI H-Tile のリソース消費量の比較を行う．ただし，2 つの実装は，実装している機能としては類似性があるが，実装方法はまったく異なる．そのため，厳密なリソース使用量の比較が困難であることに留意していただきたい．本節で行う比較は，おおむね近しいと考えられる機能ブロックどうしで比較を行うものである．

表 8 に，通信機能のコア部分実装に用いられているリ

表 8　通信機能に関するリソース使用量比較
Table 8　Comparison of resource utilization for communication.

	ALMs	Registers	M20Ks	DSPs
CIRCUS	3.9%	1.9%	1.6%	0%
SMI	4.3%	2.0%	1.9%	0%

表 9　Ping に関するリソース使用量比較
Table 9　Comparison of resource utilization for ping.

	ALMs	Registers	M20Ks	DSPs
CIRCUS	1.0%	0.6%	0%	0%
SMI	0.5%	0.3%	0.1%	0%

ソース使用量の比較を示す．CIRCUS 側は，BSP 中のリソース使用量に，Multiplexer と Demultiplexer カーネルのリソース使用量を加えたものである．SMI 側は，BSP 中のリソース使用量に，4 つの Send Communication Kernels（CK_S）と 4 つの Receive Communication Kernels（CK_R）カーネルのリソース使用量を加えたものである．SMI のパケット通信網は，これら 2 種類の通信カーネルで構成され，ルータ機能を構成する．CIRCUS は 3.9%のリソースを使用し，SMI は 4.3%のリソースを使用しており，両者のリソース使用量に大差はないといえる．

次に，Pingpong ベンチマークを実装するためのカーネルにおけるリソース消費量の比較を行う．“Pingpong” の “Ping” に相当する送信側リソース使用量を比較することとし，結果を**表 9** に示す．CIRCUS は Ping を行うカーネルと送信パケット生成のカーネルを合算したもの，SMI は 2 つの送信用カーネルを合算したものである．SMI 側に 2 つのカーネルが存在する理由は，送信用・受信用それぞれ 2 つのカーネルを動作させ，性能を測定する実装となっているためである．表 9 より，CIRCUS は 1.0%のリソース使用量，SMI は 0.5%のリソース使用量であることが分かる．CIRCUS は 2 倍のリソース消費量であるが，これには 2 つの理由が考えられる．1 つは 512 bit 幅のパケットストリームを生成していること，もう 1 つは，データの端数処理の差によるものと考えている．

たとえば，64 bit のデータを含むパケットを作成するケースを考える．SMI のパケットペイロード長は 224 bit であるため，SMI は 64 bit データを 3 個含むパケットを作成する．すなわち，224 bit のペイロード中に $64 \times 3 = 192$ bit のデータを詰め，残りの 32 bit はゼロである．一方，CIRCUS では，できるだけデータでペイロードを埋めるような処理を行う．CIRCUS のパケットペイロード長は 480 bit であるが，これを 7.5 個のデータとして扱う．溢れた 0.5 個分のデータはをバッファリングし，次のパケットのペイロードとして扱うようにしている．

SMI の実装方式の方が必要とするリソースは少ないが，アプリケーションが用いるデータサイズによっては，パケットにデータを詰め込むことができず，実効性能が低下する可能性がある．以上の結果より，CIRCUS のリソース使用量は SMI と比べると多く，特にパケット処理部で 2 倍のリソースを消費している．しかしながら，アプリケーションによっては多数の Channel を利用することが考えられるため，この部分の最適化は重要であり今後の課題である．

8.　まとめと今後の課題

本論文で我々は OpenCL で記述できる高速 FPGA 間通信システムである CIRCUS を提唱し，性能評価を行った．CIRCUS は FPGA 内通信に用いられる Channel を FPGA 間に拡張するものである．Channel を使うことで，CIRCUS は通信と演算を融合したパイプラインを構築できる．そして，通信と演算をクロックサイクルの粒度でオーバラップできる．この特性は FPGA 特有のものであり，通信と演算を融合することで，HPC アプリケーションを FPGA 上で加速できると考えている．

本論文では，2 つのベンチマークを用いて CIRCUS システムの性能評価を行った．Pingpong ベンチマークを用いて基礎通信性能の評価を行い，最小レイテンシ 0.5 μs，最大スループット 90.2 Gbps が得られた．また，1 ホップにかかる追加のレイテンシは約 250 ns であった．この結果より，CIRCUS は FPGA が持つ強力な通信機構を OpenCL から扱えることを示せたと考えている．次に，Allreduce-like なベンチマークを用いて，パイプライン構造の評価を行った．このベンチマークでは，Pingpong ベンチマークに対して演算を加えたパイプラインを構築しているが，Pingpong と同じ 90.2 Gbps のスループットが得られた．この結果は，通信と演算が一体となったパイプラインを正しく構築できたことを示すものである．

CIRCUS の通信機能を既存の実装と比較し，CIRCUS が他の実装よりも高い性能が得られることを示した．これは Verilog HDL でルータ機能を実装し，100 Gbps の通信に最適化したためと考えられる．SMI 実装の優れている点はフロー制御と誤り訂正を持つことであり，実利用を考慮すると，エラー処理は重要であるため，CIRCUS にもフロー制御とエラー処理（再送，もしくは訂正）を実装しなければならないと考えている．

前述した通信エラーに対する処理の追加に加えて，リソース使用量の最適化と，ルータの機能強化が今後の課題としてあげられる．リソース使用量に関しては，一部機能を Verilog HDL で実装し，細かな最適化を施すことで改善できると考えている．また，集団通信はルータの機能として組み込むことが可能である．たとえば，Tofu Interconnect D は低オーバヘッドなハードウェアで構成されたバリア機構を持ち，この機能は Tofu の通信チップに実装されている [15]．頻繁に用いられる集団通信をルータ側に実装で

きれば，OpenCL カーネルに由来する性能・リソース両面のオーバヘッドを削減できる．我々は宇宙物理学のアプリケーションの FPGA 向け最適化も行っており [16], [17]，今後，CIRCUS の通信システムをそのアプリケーションに適用し，複数 FPGA を用いた並列計算を行う予定である．

謝辞　本研究の一部は，「高性能汎用計算機高度利用事業」における課題「次世代演算通信融合型スーパーコンピュータの開発」および，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．最後に，筑波大学情報学群情報科学類柏野隆太氏に感謝を表する．本論文で用いた性能データ測定の一部に協力していただいた．

参考文献

[1] NVIDIA Corporation: GPUDirect for RDMA, available from ⟨https://docs.nvidia.com/cuda/gpudirect-rdma/index.html⟩.

[2] Fujita, N., Kobayashi, R., Yamaguchi, Y., Ueno, T., Sano, K. and Boku, T.: 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)).

[3] Kenter, T., Mahale, G., Alhaddad, S., Grynko, Y., Schmitt, C., Afzal, A., Hannig, F., Forstner, J. and Plessl, C.: OpenCL-Based FPGA Design to Accelerate the Nodal Discontinuous Galerkin Method for Unstructured Meshes, *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines* (*FCCM*), pp.189–196 (online), DOI: 10.1109/FCCM.2018.00037 (2018).

[4] Putnam, A., Caulfield, A., Chung, E., Chiou, D., Constantinides, K., Demme, J., Esmaeilzadeh, H., Fowers, J., Gray, J., Haselman, M., Hauck, S., Heil, S., Hormati, A., Kim, J.-Y., Lanka, S., Peterson, E., Smith, A., Thong, J., Xiao, P.Y., Burger, D., Larus, J., Gopal, G.P. and Pope, S.: A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services, *Proc. 41st Annual International Symposium on Computer Architecuture* (*ISCA*), pp.13–24, IEEE Press (2014) (online), available from ⟨https://www.microsoft.com/en-us/research/publication/a-reconfigurable-fabric-for-accelerating-large-scale-datacenter-services/⟩.

[5] Center for Parallel Computing: PC2 - Noctua (Universität Paderborn), available from ⟨https://pc2.uni-paderborn.de/hpc-services/available-systems/noctua/⟩.

[6] De Matteis, T., de Fine Licht, J., Beránek, J. and Hoefler, T.: Streaming Message Interface: High-performance Distributed Memory Programming on Reconfigurable Hardware, *Proc. International Conference for High Performance Computing, Networking, Storage and Analysis*, *SC '19*, pp.82:1–82:33, ACM (online), DOI: 10.1145/3295500.3356201 (2019).

[7] Idomura, Y., Nakata, M., Yamada, S., Machida, M., Imamura, T., Watanabe, T., Nunami, M., Inoue, H., Tsutsumi, S., Miyoshi, I. and Shida, N.: Communication-overlap techniques for improved strong scaling of gyrokinetic Eulerian code beyond 100k cores on the K-computer, *The International Journal of High Performance Computing Applications*, Vol.28, No.1, pp.73–86 (online), DOI: 10.1177/1094342013490973 (2014).

[8] Fujita, N., Kobayashi, R., Yamaguchi, Y. and Boku, T.: Parallel Processing on FPGA Combining Computation and Communication in OpenCL Programming, *2019 IEEE International Parallel and Distributed Processing Symposium Workshops* (*IPDPSW*), pp.479–488 (online), DOI: 10.1109/IPDPSW.2019.00089 (2019).

[9] Kobayashi, R., Oobata, Y., Fujita, N., Yamaguchi, Y. and Boku, T.: OpenCL-ready High Speed FPGA Network for Reconfigurable High Performance Computing, *Proc. International Conference on High Performance Computing in Asia-Pacific Region*, *HPC Asia 2018*, pp.192–201, ACM (online), DOI: 10.1145/3149457.3149479 (2018).

[10] Scalable Parallel Computing Laboratory, ETHZ: SMI, available from ⟨https://github.com/spcl/SMI⟩.

[11] Intel Corporation: SerialLite III Streaming Intel FPGA IP, available from ⟨https://www.intel.com/content/www/us/en/programmable/products/intellectual-property/ip/interface-protocols/m-alt-seriallite3.html⟩.

[12] Intel Corporation: E-Tile Transceiver PHY User Guide, available from ⟨https://www.intel.com/content/www/us/en/programmable/documentation/kqh1479167866037.html⟩.

[13] Alverson, R., Roweth, D. and Kaplan, L.: The Gemini System Interconnect, *2010 18th IEEE Symposium on High Performance Interconnects*, pp.83–87 (online), DOI: 10.1109/HOTI.2010.23 (2010).

[14] Intel Corporation: FPGAs and Programmable Devices / Documentation / Intel Quartus Prime Pro Edition User Guide: Debug Tools, available from ⟨https://www.intel.com/content/www/us/en/programmable/documentation/nfc1513989909783.html⟩.

[15] Ajima, Y., Kawashima, T., Okamoto, T., Shida, N., Hirai, K., Shimizu, T., Hiramoto, S., Ikeda, Y., Yoshikawa, T., Uchida, K. and Inoue, T.: The Tofu Interconnect D, *2018 IEEE International Conference on Cluster Computing* (*CLUSTER*), pp.646–654 (online), DOI: 10.1109/CLUSTER.2018.00090 (2018).

[16] Fujita, N., Kobayashi, R., Yamaguchi, Y., Oobata, Y., Boku, T., Abe, M., Yoshikawa, K. and Umemura, M.: Accelerating Space Radiative Transfer on FPGA Using OpenCL, *Proc. 9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*, *HEART 2018*, pp.6:1–6:7, ACM (online), DOI: 10.1145/3241793.3241799 (2018).

[17] 藤田典久，小林諒平，山口佳樹，朴　泰祐，吉川耕司，安部牧人，梅村雅之：宇宙輻射輸送コードにおける OpenCL による FPGA 演算加速最適化，情報処理学会論文誌コンピューティングシステム（ACS），Vol.12, No.3, pp.64–75 (2019).

藤田 典久　（正会員）

2016 年筑波大学システム情報工学研究科博士後期課程修了．博士（工学）．同年同大学計算科学研究センター研究員．2019 年より同センター助教．高性能計算における演算加速装置およびそれらを接続する通信機構に関する研究に従事．

小林 諒平　（正会員）

2011 年上智大学理工学部電気電子工学科卒業．2016 年東京工業大学大学院情報理工学研究科博士課程修了．博士（工学）．同年筑波大学計算科学研究センター助教．高性能計算のための FPGA の利活用技術についての研究に従事．電子情報通信学会，ACM，IEEE 各会員．

山口 佳樹　（正会員）

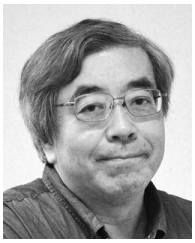
1998 年筑波大学第三学群工学システム学類卒業．2000 年筑波大学大学院修士課程理工学研究科修了．2003 年筑波大学大学院一貫制博士課程工学研究科修了．博士（工学）．同年理化学研究所横浜研究所ゲノム科学総合研究センター研究員，2005 年筑波大学大学院システム情報工学研究科講師，2015 年同准教授，現在に至る．リコンフィギャラブルシステム，特に FPGA を用いた高効率計算に関する研究に従事．Europar 2006 Distinguished Paper，Significant Paper in FPL conferences 等．ACM，IEEE，電子情報通信学会，人工知能学会，自動車技術会，オペレーションズ・マネジメント＆ストラテジー学会各会員．

上野 知洋　（正会員）

2016 年東北大学大学院情報科学研究科で博士（情報科学）取得．博士課程在学中に日本学術振興会特別研究員（DC2）．2016 年東北大学研究員として内閣府の ImPACT プロジェクトに参加．2017 年 12 月より理化学研究所計算科学研究機構（のちに計算科学研究センター）特別研究員．これまで，FPGA を用いたデータ圧縮，画像認識および高性能計算アーキテクチャとネットワークに関する研究に従事．研究分野は，情報理論，データ圧縮，人工知能，高性能コンピューティング，再構成可能コンピューティング，ネットワークシステム等．電子情報通信学会，IEEE 各会員．

佐野 健太郎　（正会員）

2000 年 3 月東北大学大学院情報科学研究科で博士（情報科学）取得．同年 6 月同大学大学院工学研究科助手．2001 年 4 月同大学大学院情報科学研究科助手．2005～2018 年 4 月同助教授（のちに准教授）．2006～2007 年インペリアルカレッジロンドン在外研究員．2017 年 4 月理化学研究所計算科学研究機構（のちに計算科学研究センター）チームリーダ．2019 年 4 月より東北大学大学院情報科学研究科客員教授．高性能計算のための FPGA を用いたカスタム計算機・リコンフィギャラブル計算機システムに関する研究に従事．数値流体力学のためのシストリック計算メモリアレイ，データフロー型ストリーム計算機とそのためのハードウェア高位合成コンパイラ，帯域圧縮ハードウェア，津波シミュレーション FPGA アクセラレータ，密結合 FPGA クラスタ等の研究開発を進めている．電子情報通信学会，IEEE コンピュータソサイエティ，ACM 米国計算機学会各会員．高効率アクセラレータとリコンフィギャラブル技術に関する国際会議 HEART の創設およびステアリング委員，国際会議の組織委員，およびプログラム委員に多数従事．



朴 泰祐　（正会員）

1960 年生．1984 年慶應義塾大学工学部電気工学科卒業．1990 年同大学大学院理工学研究科電気工学専攻後期博士課程修了．工学博士．1988 年慶應義塾大学理工学部物理学科助手．1992 年筑波大学電子･情報工学系講師, 1995 年同助教授，2004 年同大学大学院システム情報工学研究科助教授，2005 年同教授，現在に至る．超並列計算機アーキテクチャ，ハイパフォーマンスコンピューティング，クラスタコンピューティング，GPU コンピューティングに関する研究に従事．筑波大学計算科学研究センターにおいて，超並列計算機 CP-PACS，PACS-CS，HA-PACS 等の研究開発を行う．2002 年および 2003 年度情報処理学会論文賞，2011 年 ACM ゴードンベル賞，2012 年度情報処理学会山下記念研究賞各受賞．2015 年度情報処理学会フェロー認証．IEEECS，ACM 各会員．本会フェロー．