Vol.2020-HPC-173 No.24
2020/3/18

スーパーコンピュータ Cygnus 上における FPGA 間パイプライン通信の性能評価

藤田 典久[1]　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]

概要：再構成可能なハードウェアの一つに Field Programmable Gate Array (FPGA) がある．我々は，FPGA が持つ協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．CIRUCS 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ーコンピュータ Cygnus を運用しており，本稿では Cygnus 上で CIRCUS の通信性能の評価を行い報告する．

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 100Gbps × 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) には及ばない．よって，GPU ではうまく扱えず，効率が低くなってしまう演算を FPGA にオフロードする必要がある．我々は，低オーバー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，そのような処理の一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 100Gbps × 4 の非常に強力な外部通信ポートを持つ．これらのインターフェイスは 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ー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 と全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．そこで，我々は，Communication Integrated Reconfigurable CompUting System (CIRCUS) システムを提唱する．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構築する．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ー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ーラップを記述しなければならなかった．例えば，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 しなければならず，また，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

[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] 理化学研究所 計算科学研究センター

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ーラップできる．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することである．本研究では，性能評価に Intel Stratix10 FPGA を搭載した Nallatech 520N FPGA ボードを用いる．そのボードは 4 つの QSFP28 外部ポートをもち，それぞれが最大 100Gbps の通信をサポートする．並列システムを構築するために，FPGA 間を Mesh/Torus といった静的なネットワークトポロジで接続し，Intel SerialLite III (SL3) intellectual property (IP) をデータリンク層のプロトコルとして用い，外部スイッチは用いない．SL3 は peer-to-peer のプロトコルであり直接接続されている FPGA 間でしか通信できないため，我々は OpenCL 環境にルーターを追加し，非隣接 FPGA 間でも通信できるようにする．本稿では，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．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．

本稿の貢献は以下の通りである．

- 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．本稿では，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-like な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．

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．[2] で，著者らは 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．HPC 研究会においても，[3] や [4] で FPGA と OpenCL を用いた研究報告がなされているが，どちらでも OpenCL の最適化の困難さ，すなわち，CPU や GPU と異なる記述スタイルが必要であると述べられている．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．[5] 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 [6] 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 40Gbps の FPGA 間通信ができる．また，光スイッチを用いているため，要求に応じて，FPGA 間の接続関係を自由に変更できる．Tiziano らは Streaming Message Interface (SMI) [7] を Noctua 上で開発し評価を行った．SMI は，本稿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併せて利用し，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盛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言えない．

3. Intel FPGA SDK for OpenCL

3.1 SDK の概要

Intel 社は Intel FPGA SDK for OpenCL という高位合成開発環境を自社の FPGA 向けにリリースしている．その SDK を使うことで，OpenCL を用いて FPGA のプログラミングができる．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，その SDK を使うだけで FPGA システムを構築できる．

図 1 に OpenCL SDK を使う際の FPGA 回路の概要を示す．FPGA 回路は 2 つの部分から成る．1 つは，Board

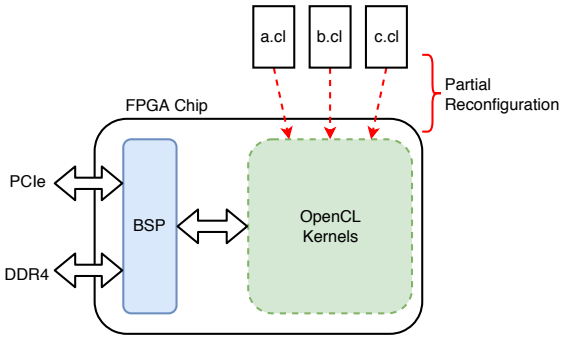

図 1: Intel FPGA SDK for OpenCL を利用する FPGA の内部構造.

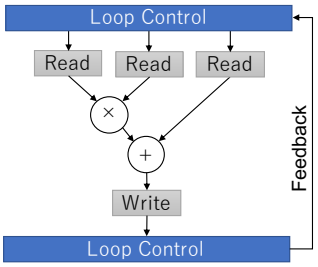

図 2: OpenCL コンパイラによって作成されるパイプラインの例.

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 に固有の要素である．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．例えば，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，例えば，PCIe コントローラやメモリコントローラが含まれる．

3.2 Pipeline 構造

Intel FPGA SDK for OpenCL を用いてプログラムを作成する際，コンパイラは OpenCL コードに基づくパイプラインを FPGA 内に作成する．コンパイラが FPGA 内にプロセッサを作り，その上で OpenCL コードをソフトウェアとして実行しているわけではない．

図 3 のコードから生成されたパイプライン構造を**図 2** に示す．コンパイラのハードウェア生成ルールは，OpenCL コードのループ構造をパイプライン構造に変換するものである．パイプライン制御モジュールをループボディから作成したパイプラインの前後に配置する．OpenCL コード中のメモリアクセスは Load Store Unit (LSU) に変換され，パイプラインの中に組み込まれる．

```
for (int i = 0; i < n; i++)
 d[i] = a[i] * b[i] + c[i]
}
```

図 3: OpenCL のループの例.

3.3 Channel 拡張

Intel FPGA SDK for OpenCL は FPGA 固有の拡張を OpenCL に加えている．“Channel” は拡張の一つであり，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．

通常の環境では，Global メモリを使ってデータ交換を行うことが一般的である．Channel を使うと，コンパイラは通信のためのデータパスを FPGA 内に作り，DDR4 のような off-chip メモリへのアクセスを必要としない．したがって，外部メモリへアクセスするためのモジュール (LSU) を必要としないため，Channel は従来手法と比べて高い性能と少ない FPGA リソース消費量を達成できる．

“I/O Channel” は SDK による拡張の一つであり，Channel に似た機構である．っこれは，OpenCL カーネルと BSP の間を接続する Channel であり，外部ペリフェラルを制御するために用いられる．

3.4 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時は BSP に制御回路や制御 intellectual property (IP) を追加実装しなければならない．

我々は [8] や [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．40Gb 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．本稿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，本稿ではどの様にして 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. CIRCUS

4.1 システムの概要

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．

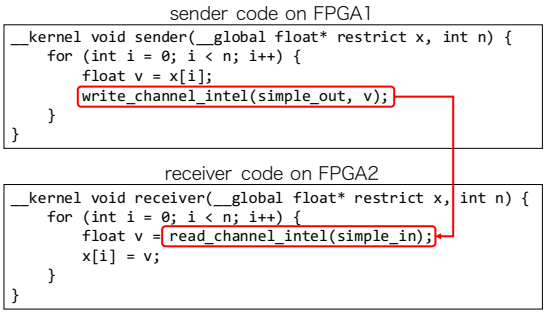

図 4: CIRCUS を用いた通信コード例.

表 1: CoE と CIRCUS の比較.

	CoE [10], [11]	CoE 2.0[12]	CIRCUS
FPGA	Arria10	Stratix10	Stratix10
通信速度	40Gbps	100Gbps	100Gbps
Protocol	Ethernet	SL3	SL3
Topology	Tree	Peer-to-Peer	Peer-to-Peer
間接通信	Switch	N/A	Router

CIRCUS システムは，SerialLite III (SL3) 通信 IP，ルーターモジュール，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．

これ以降，CIRUCS 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．他の目的で使われている Channel と区別するためである．**図 4** は CIRUCS を使う簡単なコード例である．図中にある 2 つのカーネルは別々の FPGA 上で動作しており，CIRCUS システムが送信側の “simple_out” channel と受信側の “simple_in” channel を接続する．

4.2 これまでの研究との違い

CIRCUS システムは，我々がこれまで開発してきた CoE システム [10], [11] および CoE 2.0 システム [12] の後継システムである．本説では，本稿で提案する CIRCUS が，これまでの研究と何が違うのかについて述べる．**表 15** はそれぞれの通信機構の比較を示した表である．

通信と演算を一体化したパイプラインを作成することは，FPGA にとって必要不可欠な要素である．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，Ethernet プロトコルは Streaming 通信には適さないと考えている．また，CoE 開発時に用いていた FPGA は外部ポートが 2 つしかなく，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．本稿でも用いるボードもそうであるが，4 外部ポートを持つ FPGA ボードが登場しており，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．したがって，CoE では Ethernet プロトコルを用いていたが，CIRCUS では Intel から提供されている SL3 プロトコルを用いることを決定した．

CIRCUS システムと CoE 2.0 システム [12] の違いは，ルーターモジュールを BSP に組み込んだことである．SL3 は Peer-to-Peer プロトコルなため，ルーターモジュールを BSP に加えることで，非隣接 FPGA 間通信を可能とする．また，上述したように Ethernet の使用を取りやめたため，名前を変更したものである．詳細は性能評価の章で述べるが，本稿ではスーパーコンピュータ Cygnus を用いて性能評価を行い，最大で 8 FPGA を用いて性能評価を行った．また，CoE を含めた他の OpenCL 通信システムとの性能評価を行う．

4.3 コード生成

アプリケーションに CIRCUS を適用する時，通信に対する要求，例えば 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．したがって，我々はコードジェネレータを開発し，CIRCUS に関するコードを自動生成させる．コードジェネレータは通信に関する構造を定義した XML ファイルを読み込み，そして OpenCL カーネルコードとホスト用のランタイムコードを生成する．

図 5 は CIRCUS ジェネレータを使う際の開発フローを示す．図に含まれているファイルの意味は以下の通りである．

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 5 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．

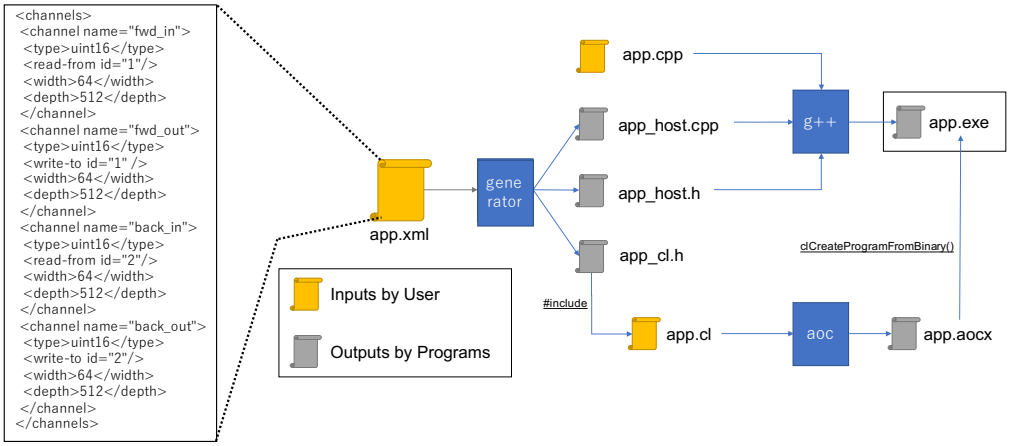

図 5: CIRCUS システムの開発フロー.

4.4 通信と演算を融合したパイプライン

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ー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一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ーラップし通信隠蔽を行う [13].

例えば，オーバーラップを Message Passing Interface (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，FPGA と OpenCL を用いる場合，3.2 節にあるように，演算はパイプラインに変換され FPGA に実装される．また，CIRCUS は Channel に基づく通信を提供するものであり，通信に関する処理もパイプラインが構築される．したがって，両者を組み合わせると，通信と演算の両方を融合したパイプラインを構築できる．言い換えると，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ーラップされているといえる．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.

4.5 ルーター

SerialLite III プロトコルは直接通信用のものであるため，隣接 FPGA 間でしか通信ができない．この問題を解決するために，我々はルーター機構を BSP に組み込む．**図 6** にルーターの構造を示す．青矢印はそれぞれ 384MHz で駆動される 256bits 幅の全二重のバスを示す．($0.384 \times 256 = 98.3$Gbps)

図 7 にルーターの内部ダイアグラムを示す．ルーターは 5 入力ポート，5 出力ポート，クロスバー，ルーティング

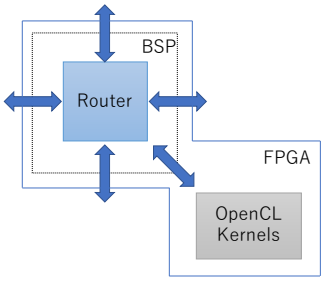

図 6: ルーターとカーネルの接続関係．青矢印は 256bit 幅の全二重バスを表す.

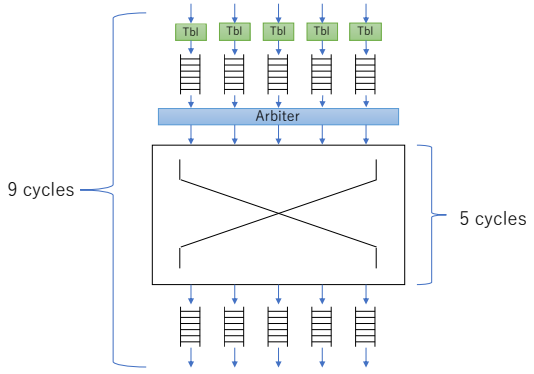

図 7: ルーターのダイアグラム．“Tbl” はルーティングテーブルを参照するモジュールを表す.

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” はルーターの入出力を調停し，パケットが衝突しないように調整を行う．ルーターの動作周波数は通信層の物理層 (SL3) の動作周波数にリンクしており，バスと同じく 384MHz 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 サイクル (~24ns) で，クロスバー本体のレイテンシは 5 サイクル (~13ns) であった.

CIRCUS パケットは 6 ビット幅の宛先フィールドをヘッダーの中に持っている．パケットがルーターに到着すると，

表 2: 評価環境 (Cygnus)

CPU	Intel Xeon Gold 6126 × 2
CPU Memory	DDR4 192GB (96GB / 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 2400MHz 32GB (8GB × 4)
Comm. Port	QSFP28 × 4

ヘッダの中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る出すかを決定する．ルーターの宛先テーブルは 64 エントリを持ち，最大で 64 FPGA のネットワークを構築できる．この数は実験環境の最大 FPGA 数に合わせて設定したものであるが，必要に応じてパラメータを変更することでサイズを調整できる．

4.6 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. 性能評価

5.1 評価環境

本稿では，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を性能評価に用いる．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，32GB DDR4 メモリ，最大 100Gbps をサポートする QSFP28 ポートを 4 つ持つ．

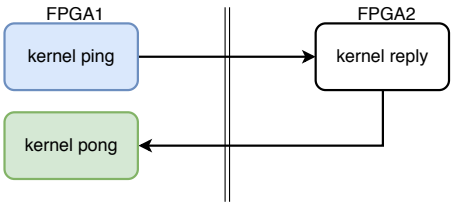

図 8: Pingpong ベンチマークのデータの流れ．

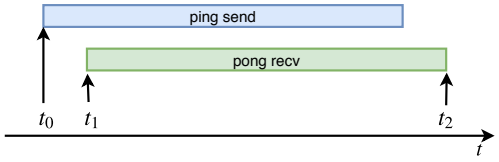

図 9: Pingpong ベンチマークのタイムライン．

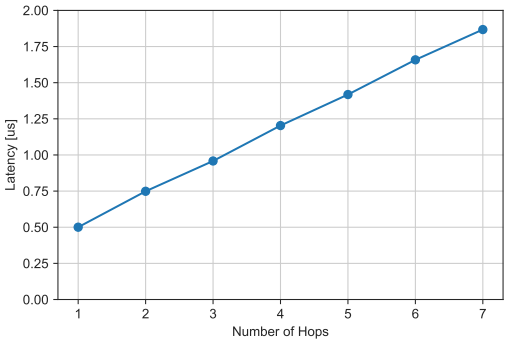

図 10: Pingpong ベンチマークの結果（レイテンシ）．

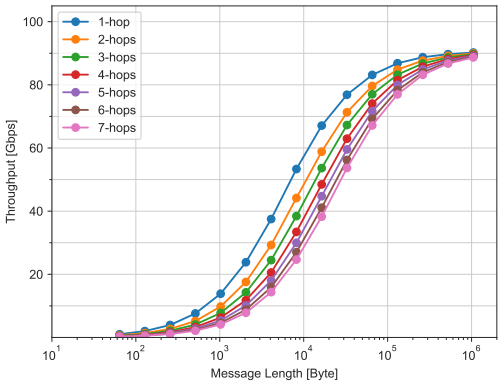

図 11: Pingpong ベンチマークの結果（スループット）．

Cygnus システムには，トータルで 64 枚の FPGA がある (32 Albideo 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．

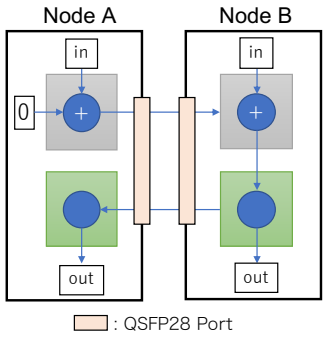

図 12: Allreduce-like ベンチマークのデータの流れ.

5.2 Pingpong Benchmark

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 8** 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 オーバーヘッドも含む．

Pingpong における通信時間計測は**図 9** 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]．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容易に時間測定が行える．OpenCL カーネルの動作周波数はコード毎に可変であるが，Stratix10 デバイス使用時に，一般的に 250MHz 前後の周波数が得られる．Pingpong ベンチマークの結果を**図 10** と**図 11** に示す．メッセージ長 16Byte〜1MB，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．例えば，“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 (512bits width) の CIRCUS Channel を使い，動作周波数は 295.8MHz である．最小レイテンシと最大レイテンシは，それぞれ 0.5μs and 1.87μs である．また，1 ホップあたりの増加レイテンシは約 0.25μs であった．通信スループットは 1-hop 時に 90.2Gbps，7-hops 時に 88.7Gps が得られた．高 hop 時は，通信レイテンシが増加するため，それに応じたスループット低下が見られる．ただし，これらの通信性能は OpenC カーネルの周波数に依存するため，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．

5.3 パイプライン通信と演算

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ーラップされ，通信時間

```
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 13: Allreduce-like ベンチマークのコードの一部.

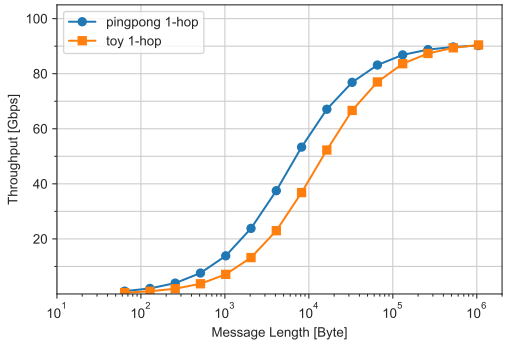

図 14: Allreduce-like ベンチマークの結果.

を完璧に隠蔽できる．このオーバー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 12** 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 で実行した時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．一つは送信・加算カーネル (図 12 の灰色の箱) であり，もう一つは受信カーネル (図 12 の緑色の箱) である．Figure 13 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) ことに注意して頂きたい．

図 14 にベンチマークの測定結果を示す．最大スルー

表 3: リソース使用量: BSP Modules

	ALMs	Registers	M20Ks	DSPs
Design Total	21.5%	19.0%	5.6%	0%
CIRCUS BSP	3.3%	2.9%	1.6%	0%
BSP Others	12.2%	0.9%	2.6%	0%
Kernels	5.5%	6.9%	1.3%	0%
Others	0.5%	0.2%	0.1%	0%

表 4: リソース使用量: Kernels and Channels

	ALMs	Registers	M20Ks	DSPs
CIRCUS Out (x2)	1.0%	1.2%	0%	0%
CIRCUS In (x2)	0.9%	1.2%	0%	0%
CIRCUS Mux.	0.4%	0.5%	0%	0%
CIRCUS Demux.	0.2%	0.3%	0%	0%
Memory Network	1.2%	1.7%	1.0%	0%
Pingpong	1.2%	1.6%	0.3%	0%
Others	0.6%	0.5%	0%	0%

プットは 90.2Gbps が得られた．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．

5.4 リソース使用率

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, Regissters, M20Ks, DSPs はそれぞれ，Adaptive Logic Modules (ALMs), ALM 中の Flip-Flops, 20Kbits 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 3 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 (SL3 とルーター)，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 4 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 1.6%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．本稿の実装では，毎クロック毎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 5: プロトコルから発生する通信オーバーヘッド．

Factor	Throughput of payload	Efficiency
Physical Speed	103.125Gbps	
67b/64b	98.484Gbps	×0.955
Meta Frame	98.287Gbps	×0.998
SL3 Burst	96.813Gbps	×0.985
CIRCUS Header	90.762Gbps	×0.938

6. 考察

Pingpong ベンチマークの結果では，90.2Gbps の性能を達成した．100Gbps 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，この性能は設計通りのものが得られている．**表 5** に利用しているプロトコルから発生するオーバーヘッドを示す．まず，本稿で用いる実装は，物理層に 103.125Gbps ($= 4 \times 25.78125$) の速度を用いている．この速度は 100Gbit Ethernet と同じものである．“67b/64b”, “Meta Frame”, “SL3 Burst” の項目は SerialLite III プロトコルに由来するオーバーヘッドを示す．これらの値は SerialLite III のドキュメント [14] 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，その中で 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ーヘッドは “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．次世代の FPGA，例えば Intel Stratix10 E-Tile Transceiver[15] では，Pulse Amplitude Modulation-4 (PAM-4) を採用することで通信帯域を倍にできる．したがって，我々は将来的に PAM-4 を使うことでこの問題を緩和できると考えている．

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 250ns とわかった．ルーターは Verilog HDL で実装しているため，ルーターにかかるレイテンシ

は詳細に求めることができる．ルーターのレイテンシは23.427ns (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 225ns とわかる．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．Cray Gemini Interconnection Network では，ノード間のレイテンシが 105ns[16] 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．

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し，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．

リソース消費量の解析では，CIRCUS は 3.3%のリソースを BSP に追加することがわかった．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毎に異なり，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挙げられる．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 コードのみ扱えば良く，CIRCUS を用いた通信を利用できる．

7. 既存の実装との比較

本章では，CIRCUS の通信性能を既存の実装と比較する．比較対象は，我々の既存研究である CoE[10], [11] と SMI[7] である．**表 6** に，それぞれの通信実装の概要を示す．ここで，“SMI-HTile” は，SMI の実装を我々の実験環境で実行して計測したものである．Bittware 520N H-Tile FPGA board (up to 100Gbps) と Intel FPGA SDK for OpenCL 19.1.0 を用いている点が異なる．“SMI” は，520N L-Tile FPGA board (up to 40Gbps) とバージョン 18.1.1 の SDK を用いて測定されている．H-Tile ボード用の BSP は古いコンパイラバージョンをサポートしないため，異なるバージョンのコンパイラを用いて評価を実施した．Stratix10 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．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．520N L-Tile ボードは一部の通信ポートでしか 100Gbps を扱えず残りは 40Gbps に制限されるが，520N H-Tile ボードはすべての通信ポートで 100Gbps を扱えるという違いがある．なお，“SMI-HTile” の評価には Cygnus を用いてないことに注意していただきたい．ただし，Cygnus と同じ FPGA ボードを利用して評価を行っており，Cygnus 上の性能も同程度になることが期待される．“SMI” と “SMI-HTile” はボードベンダーから提供される標準 BSP を利用している．したがって，通信には Bittware 社が開発した Proprietary なプロトコルが用いられており，フロー制御と誤り訂正が含まれている．図 15 に 4 つの実装の性能比較の結果を示す．また，**表 7** 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

表 6: 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．

	CIRCUS	CoE [10], [11]	SMI [7]	SMI-HTile
Board	520N H-Tile	A10PL4	520N L-Tile	520N H-Tile
FPGA	S10	A10	S10	S10
Protocol	SL3	Ethernet	Bitt	Bitt
Peak BW.	100Gbps	40Gbps	40Gbps	100Gbps
Switch	No	Yes	No	No

表 7: 各実装の最小レイテンシ．

CIRCUS	CoE [10], [11]	SMI [7]	SMI-HTile
500ns	950ns	801ns	517ns

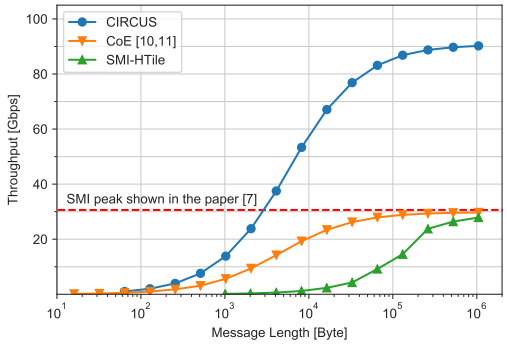

図 15: 実装間の性能比較．

ているが，スイッチ 1 台を経由した通信を “1-hop” として用いた．また，SMI の実装は Github[17] で公開されているものを用い，性能評価には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．

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．

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 15 に記述した．SMI のヘッダ (4 バイト) を含んで 35Gbps 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 30.625Gbps がピーク性能だと定義した．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．この性能の違いはルーター部分の設計方針の違いから来ていると考えている．CIRCUS のルーターは Verilog HDL で記述しているが，SMI のルーターは OpenCL で記述されている．SMI のアプローチは柔軟性にメリットがある．OpenCL コードの開発は Verilog HDL の開発よりも容易であるため，変更を行いやすい．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り，我々はルーターを Verilog HDL で開発しても保守において大きな問題ではないと考えている．一般的に，ルーターの実装を変更する頻度は高くないためである．

SMI は 100Gbps 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，[7] 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 40Gbps 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．Bittware の BSP は 256bit 幅の I/O Channel で BSP と OpenCL カーネル間を接続しているが，SMI の性能が律速されている理由の一つにこの設計がある．SMI ベンチマークの動作周波数は 266.67MHz 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，100Gbps の通信路を満たすことができない．CIRCUS では，この問題を緩和するために 2 倍の 512bit 幅の I/O Channel を用いて BSP と OpenCL カーネル間を接続している．

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る．CIRCUS の実装はそれらの機能を持たない．SerialLite III は Cyclic Redundancy Check (CRC) を用いているため，フレーム中の誤りを検出することはできるが，訂正することはできない．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．

8. まとめと今後の課題

本稿で我々は 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ーラップできる．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．

本稿では，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 0.5μs，最大スループット 90.2Gbps 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 250ns であった．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 90.2Gbps 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 でルーター機能を実装し，100Gbps 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，ルーターの機能強化が今後の課題として挙げられる．集団通信はルーターの機能として組み込むことが可能である．例えば，Tofu Interconnect D は低オーバー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 [18]．頻繁に用いられる集団通信をルーター側に実装できれば，OpenCL カーネルに由来する性能・リソース両面のオーバー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 向け最適化も行なっており [19], [20]，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」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．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．本稿で用いた性能データ測定の一部に協力して頂いた．

参考文献

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

[2] 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)*, Vol. 00, pp. 189–196 (online), available from ⟨doi.ieeecomputersociety.org/10.1109/FCCM.2018.00037⟩ (2018).

[3] 大島聡史，塙 敏博，片桐孝洋，中島研吾：FPGA を用いた疎行列数値計算の性能評価，情報処理学会研究報告，2016-HPC-153 (2016).

[4] 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，情報処理学会研究報告，2016-HPC-155 (2016).

[5] 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, *Proceeding of the 41st Annual International Symposium on Computer Architecuture (ISCA)*, IEEE Press, pp. 13–24 (online), available from ⟨https://www.microsoft.com/en-us/research/publication/a-reconfigurable-fabric-for-accelerating-large-scale-datacenter-services/⟩ (2014).

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

[7] De Matteis, T., de Fine Licht, J., Beránek, J. and Hoefler, T.: Streaming Message Interface: High-performance Distributed Memory Programming on Reconfigurable Hardware, *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, New York, NY, USA, ACM, pp. 82:1–82:33 (online), available from ⟨https://doi.org/10.1145/3295500.3356201⟩ (2019).

[8] 大畠佑真，小林諒平，藤田典久，山口佳樹，朴 泰祐：OpenCL と Verilog HDL の混合記述による FPGA 間 Ethernet 接続, 情報処理学会研究報告, 2017-HPC-160 (2017).

[9] Kobayashi, R., Oobata, Y., Fujita, N., Yamaguchi, Y. and Boku, T.: OpenCL-ready High Speed FPGA Network for Reconfigurable High Performance Computing, *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, HPC Asia 2018, New York, NY, USA, ACM, pp. 192–201 (online), available from ⟨http://doi.acm.org/10.1145/3149457.3149479⟩ (2018).

[10] 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), available from ⟨https://doi.org/10.1109/IPDPSW.2019.00089⟩ (2019).

[11] 藤田典久，小林諒平，山口佳樹，朴 泰祐：OpenCL による FPGA 上の演算と通信を融合した並列処理システムの実装及び性能評価, 情報処理学会研究報告, 2018-HPC-167 (2018).

[12] 藤田典久，小林諒平，山口佳樹，朴 泰祐：再構成可能なハードウェアを用いた演算と通信を融合する手法の提案と性能評価，情報処理学会研究報告, 2019-HPC-171 (2019).

[13] 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).

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

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

[16] Alverson, R., Roweth, D. and Kaplan, L.: The Gemini System Interconnect, *2010 18th IEEE Symposium on High Performance Interconnects*, pp. 83–87 (online), available from ⟨https://doi.org/10.1109/HOTI.2010.23⟩ (2010).

[17] Scalable Parallel Computing Laboratory, ETHZ: SMI, https://github.com/spcl/SMI.

[18] 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), available from ⟨https://doi.org/10.1109/CLUSTER.2018.00090⟩ (2018).

[19] 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, *Proceedings of the 9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*, HEART 2018, New York, NY, USA, ACM, pp. 6:1–6:7 (online), available from ⟨http://doi.acm.org/10.1145/3241793.3241799⟩ (2018).

[20] 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：並列 FPGA システムにおける OpenCL を用いた宇宙輻射輸送コードの演算加速，情報処理学会研究報告, 2018-HPC-165 (2018).