

FPGAにおけるHPCアプリケーション向け HBM2メモリシステムの提案と実装

藤田 典久^{1,2} 小林 諒平^{1,2} 山口 佳樹^{2,1} 朴 泰祐^{1,2}

概要: 高性能計算の分野で Field Programmable Gate Array (FPGA) が新たなアクセラレータとして注目されている。近年、高位合成 (High Level Synthesis: HLS) 開発環境が発展しておきており、C や C++といった言語を用いた開発が可能になりつつある。FPGA は外部メモリ帯域が弱いという課題があり FPGA を HPC で利用する際の障壁となることがあったが、High Bandwidth Memory 2 (HBM2) を搭載した FPGA チップがベンダーからリリースされ始めており、最大で 512GB/s のメモリ帯域を有する。しかしながら、FPGA には、キャッシュやメモリネットワークといったメモリを利用するための機能はなく、HBM2 を FPGA で利用する際の課題の一つである。本稿では、HPC アプリケーションに適する HBM2 メモリシステムの提案と実装を行い性能評価について報告を行う。また、高位合成で記述したカーネルから提案システムが扱えることを示す。

1. はじめに

高性能計算の分野で Field Programmable Gate Array (FPGA) が新たなアクセラレータとして注目されている。これまで、FPGA のプログラミングを行うためには、ハードウェア記述言語 (Hardware Description Language: HDL) を用いることが一般的であった。HDL での記述レベルは、Register Transfer Level (RTL) と呼ばれ、順序回路のレジスタ間での挙動を記述する必要がある。低レベルなプログラミングが求められていた。近年、高位合成 (High Level Synthesis: HLS) と呼ばれる開発環境が発展しておきており、開発の障壁が低下しつつある。HLS はソフトウェアで使われている言語でハードウェアの動作を記述できる物であり、C, C++, OpenCL が良く用いられる。

また、近年の FPGA は強力な外部通信機構を備えており、最大で 100Gbpsx4 の FPGA 間通信ができる。我々はこれまでの研究で、高位合成 (OpenCL) と FPGA 間直接通信を用いて並列計算を実現した [1], [2]。FPGA の通信能力は高く、並列計算を行う際に有益であることは明らかとなっているが、FPGA は外部メモリ帯域が弱いという課題があった。これらの研究で用いていた FPGA ボードでは、外部メモリ帯域は 76.8GB/s の帯域しかない。Graphics Processing Unit (GPU) である NVIDIA A100[3] が 2TB/s のメモリ帯域を持つことと比べると、非常に弱いと言わざ

るを得ない。

メモリ帯域が不足していることが FPGA を HPC で利用する際の障壁となることがあったが、High Bandwidth Memory 2 (HBM2) を搭載した FPGA チップがベンダーからリリースされ始めており、最大で 512GB/s のメモリ帯域を有する。A100 と比較すると 1/4 のメモリ帯域でしかないものの、10 倍やそれ以上の帯域差がある状況よりは改善されている。

本研究の目的は、HPC アプリケーションに適する HBM2 メモリシステムの提案と実装を行うことである。HBM2 搭載 FPGA のメモリ帯域は高いものの、これはあくまでもメモリ IO 性能を示すのみである。FPGA には、CPU や GPU が持つキャッシュやメモリネットワークはなく、ユーザーが実装しなければならない。また、本研究で提案するシステムは高位合成でアプリケーションを記述することを想定している。これは、HPC アプリケーションは複雑なプログラムであり、HDL を用いて FPGA に実装することは非現実的であると考えているからである。

本稿の貢献は以下の通りである。

- HPC および HBM2 向けメモリシステムの提案を行う
- システムのプロトタイプを実装し、実際に計算が行えることを示す
- 高位合成で記述したカーネルから提案システムが扱えることを示す

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

2. 関連研究

FPGA に搭載されている HBM2 を活用する研究としては、従来型のメモリ帯域が必要なアプリケーション [4] に加えて、ニューラルネットワークに適用した研究 [5], [6] が知られている。また、本研究会においても [7] の報告がある。これらの研究では、演算回路が特定の HBM2 メモリ Channel に接続されており、それ以外の channel にアクセスすることはできない。

[8] で Choi らは Xilinx 製 FPGA ボードである Alveo U280 を用いて、Bucket Sort と Merge Sort を実装し、性能評価を行った。Vivado HLS で実装した Sorter 部分と彼らが HBM Connect と呼んでいるメモリネットワークが実装されている。HBM Connect によって Sorter 回路とメモリ間が接続されており、Sorter 回路が全ての HBM2 Channel にアクセスできる。ボード開発環境の制限により、ボードにある HBM2 の半分でしか性能評価ができていないものの、理論ピークの 9 割程度の性能が達成されている。性能の代償として、HBM Connect を構成するために回路リソースを消費しており、その分アプリケーションのために使える回路リソースが減っている。

本稿で提案するシステムは、HBM2 を外部メモリとして用いることを前提としており、HBM2 が持つ数十のメモリチャンネルを前提として設計を行う。また、アプリケーションと HBM2 メモリを直接接続するのではなく、間にメモリネットワークおよび FPGA に内蔵されているメモリをキャッシュとして配置するところに本研究の新規性がある。このようなメモリシステムを実装することで、性能とアプリケーションの自由度のバランスがとれた実装が行えるものと考えている。

3. Intel FPGA における HBM2 の構造

Intel FPGA における HBM2 については、我々の先行研究 [9] で既に報告している。したがって、内容の重複を避けるため、本稿ではその中でも重要な部分について述べるにとどめる。詳細な構造や、メモリ単体としての性能について興味がある方は [9] を参照して頂きたい。

図 1 に Intel Stratix 10 MX FPGA における HBM2 のアーキテクチャを示す。2つの HBM2 ダイが接続され、容量は 8GB (4GBx2) もしくは 16GB (8GBx2) であり、メモリ帯域 (Aggregated) は最大で 512GB/s に達する。HBM2 は、遅いメモリを多数並列に駆動することで高い性能を実現している。図 1 からわかるように、1 チャンネルあたり 16GB/s の帯域しかなく、数チャンネルだけ利用するのであれば、DDR4 のような従来式のメモリを用いた方が帯域が高い^{*1}。したがって、多数のメモリチャンネルを並列に

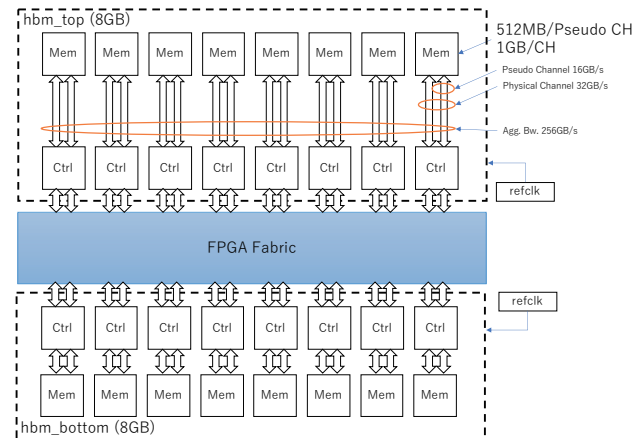


図 1: HBM2 メモリの構造。

駆動することが求められる。

CPU や GPU の場合、高性能なキャッシュやインターコネクトといったメモリシステムが搭載されており、HBM2 の帯域を十分に扱える。しかしながら、FPGA の場合、そのようなメモリシステムはなく、必要に応じてそれらを FPGA 内に実装しなければならない。HPC アプリケーションをターゲットとして考えると、実アプリケーションにおいてはメモリアクセスは複雑であり、ある程度の自由度があるメモリシステムは必須であると考えられる。FPGA で CPU などと同様の回路を構築することはもちろん可能であるが、複雑なシステムは回路リソースを多く必要とする。本来、回路リソースはすべて演算に使いたいところであるため、メモリシステムにすべてのリソースをつぎ込める訳ではない。したがって、我々は性能とコストのバランスをとったメモリシステムを FPGA に実装する必要があると考えている。

4. これまでの研究

我々はこれまでの研究 [9] で、Intel FPGA における HBM2 の性能評価を行った。この研究会報告は、HBM2 コントローラを直接制御し、HBM2 の基本的な性能について評価したものである。メモリコントローラの動作周波数が仕様上の最大値より低くしか動作できていない問題があったものの、動作周波数に応じた想定通りの性能が得られることがわかった。しかしながら、HBM2 が持つ多数のメモリチャンネルを並列に扱うことが、FPGA で HBM2 を利用する際の大きな問題であることを明らかにした。

メモリアクセスパターンが単純なマイクロベンチマークでは、メモリチャンネルの使い分けを手動で行うことは容易であるが、複雑な実アプリケーションではメモリアクセスパターンが複雑になり、制御が困難になる問題がある。前述した研究会報告では、この問題に対して HBM2 と計

という FPGA ボードでは、DDR4-2400 が 4 チャンネルあり 76.8GB/s のメモリ帯域を有する。

*1 例えば、我々が OpenCL 環境としてよく用いる Bittware 520N

算カーネルを直接接続するのではなく、間に Block RAM (BRAM) を挟み、BRAM を一種のキャッシュのように扱う手法が有効なのではないかと述べた。HBM2 は容量はあるものの、アクセスレイテンシが長くランダムアクセスが遅い。一方で、BRAM は FPGA 内蔵のメモリであるため、容量が小さいが、アクセスレイテンシが短くランダムアクセスが高速である。特性が反対な 2 種のメモリを適材適所に使うことで、FPGA 上で HBM2 をうまく扱えるのではと述べた。本稿では、先行研究では今後の課題としていた、HBM2 と BRAM を併用した HPC 向けのメモリサブシステムのプロトタイプ実装を行い、その成果について報告する。

5. 提案するメモリシステム

5.1 設計のコンセプト

Intel FPGA では、HBM2 は最大で 32 個のメモリチャンネルを持つ。アプリケーションと HBM2 のメモリコントローラを直接接続する場合、アプリケーション側で 32 チャンネルを効率よく扱わなければならない。複雑なアプリケーションで、多数のチャンネルを効率よく扱うプログラミングは非常に困難であり、事実上不可能であると我々は考えている。加えて、HBM2 は DRAM であるため、ページを跨ぐランダムアクセスでは高い性能を発揮しづらい。

以上の理由から、BRAM をキャッシュとして実装するのが良い実装方針であると考えている。HBM2 と BRAM 間の転送はバルク転送となり、ある程度の塊でデータ転送を行い、性能低下を避けられる。また、BRAM であれば、高速にランダムアクセス可能であるため、アプリケーションの実装が容易になる。

一般的にキャッシュと言うと、ハードウェアで自動で制御される実装が多い。しかしながら、FPGA でそのような実装を行うと、本来演算で使うべきハードウェアリソースをキャッシュシステムで消費してしまう。したがって、HBM2 と BRAM キャッシュ間のデータ転送はプログラマが手動で指定する方式をとる。

この方式は、プログラミングコストが増加するという問題がある。HPC の領域ではアクセラレータを用いて演算加速することが広く用いられており、CPU とアクセラレータ間のデータ転送を手動で記述することは一般的である。したがって、データ転送を手動で管理するプログラミングモデルの考え方は受け入れられていると考えている。

5.2 回路概要

図 2 に提案するメモリシステムの概要を示す。本システムは次の 6 つの要素から構成される。それぞれのコンポーネントの詳細については、次節以降で順次述べる。

- PCI Express Controller
- (External) Memory Controller

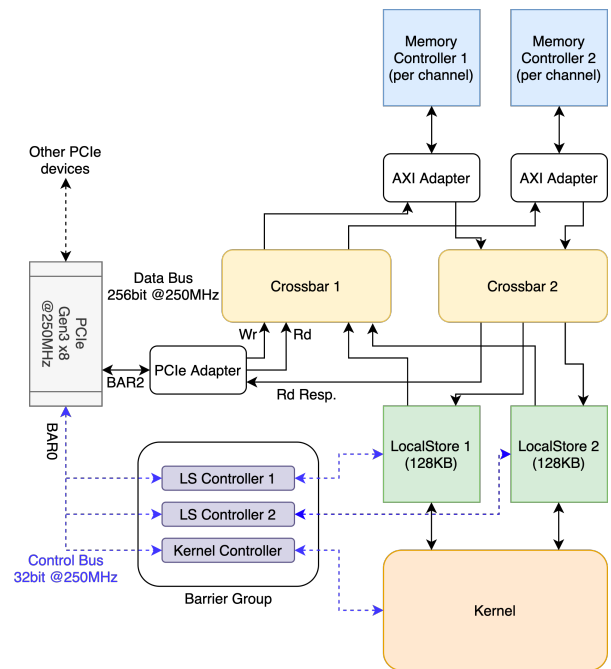


図 2: メモリネットワークの全体図。

- Local Store (LS)
- LS Controller
- Kernel
- Kernel Controller

本システムの内、Kernel 部以外は Chisel[10] で記述されている。Chisel は中間言語である FIRRTL[11] を経由して、最終的には Verilog HDL コードが生成する。Chisel は Scala 上に構築される Domain Specific Language (DSL) であり、Verilog と同等の RTL の抽象度で記述する。したがって、Chisel のコードがどのような Verilog に変換されるかの予測が行いやすく、クロックサイクルレベルで回路動作の記述が可能である。

提案するメモリシステムは、PCI Express (PCIe) バスを通じてホストから制御されることを動作の基本とする。GPU のようなアクセラレータの制御モデルと同等である。一般的なアクセラレータとは異なり、本来、FPGA は自律動作できるデバイスである。したがって、FPGA が主導となって計算や通信を開始できる能力がある。我々は、これまでに OpenCL 環境 [12] で FPGA 間通信 [13] や FPGA-GPU 間 DMA 転送 [14] に関する研究を行った。本稿の範囲では扱わないが、今後はこういった動作も扱えるようにしたいと考えている。

5.3 PCI Express

ホストとの通信用に、Intel 社が提供する Intel L- and H-tile Avalon Streaming and Single Root I/O Virtualization (SR-IOV) IP コア [15] を用いて、PCIe デバイスとしての機能を実装する。Base Address Register (BAR) 0 のアドレ

ス空間を制御用のバスに、BAR2のアドレス空間をHBM2のバスに接続している。PCIe Transaction Layer Packet (TLP)を変換回路で内部パケットに変換してCrossbar経由し、メモリアクセスを行う。

本IPおよび実験に用いるFPGAボードは、最大でGen.3 x16まで対応しているが、現在の実装ではGen.3 x8までの対応にとどめている。x16の運用を行う場合、IPコアから1クロックサイクルあたり最大で2 PCIe TLPが出力されるため、特殊な処理回路が必要となるからである。ホストや他のPCIeデバイスとのデータ転送はPCIeがボトルネックとなるため、PCIeバスの帯域は重要である。したがって、今後、x16の対応^{*2}を予定している。

5.4 Memory Controller

図2で“Memory Controller”と表している部分は、データを格納するメモリのコントローラを接続する。Intel FPGAのHBM2は最大で32メモリチャンネルを持つため、最大で32個のMemory Controllerが実装されることを想定している。しかしながら、本実装はプロトタイプ実装であるため、現時点では2個までの対応となっている。扱うチャンネル数が増えるとクロスバーに接続されるコンポーネント数が増え、クロスバーの複雑度が増す問題があり、この部分がうまく実装できるかどうかは今後の課題の一つとなる。

メモリコントローラ部分は、実装方式をHBM2とBRAMで差し替えられるように実装を行っている。HBM2コントローラのとシミュレーションには時間がかかり、開発の効率が低下してしまうためである。開発中はBRAMを主に用いて開発を行う。BRAMは容量を大きくとることが出来ないが、デバッグ用途であれば十分である。Intel社が提供するHBM2コントローラは、バスにAdvanced eXtensible Interface (AXI)規格を用いる。そのため、内部パケットとAXIを相互に変換する回路を挿入している。BRAMは内蔵メモリなため、固有のアクセスバスを持つが、HBM2側の仕様と合わせるために、AXIバスを経由してアクセスする。BRAMをAXIバスに接続する部分は、Platform Designer[16]を用いて実装している。

5.5 Local Store

Local Storeは、1個のあたり128KBの容量を持つBRAMキャッシュである。FPGAが持つBRAMサイズは、チップの型番によって異なるが、Intel FPGAの場合多くても30MB程度である。また、本稿でターゲットとしているFPGAチップのBRAM容量は約17MBである。LSをHBM2のメモリチャンネルの数と同数(32)実装することを予定しているため、1個あたり128KBという小さいサイズ

に設定している。

仮に、32個の128KB容量LSをFPGAに実装したとすると、およそ25%のBRAMを消費する。BRAMは小容量のバッファとして、FPGAシステム全体で広く用いられることと、アプリケーションカーネル内でも一時領域や配列領域として使う可能性があることを考慮すると、LSあたり128KBという容量は妥当だと考えている。

5.6 Controller

システム内に、各部の制御を司るコントローラを3つ(LS制御に2つ、カーネル制御に1つ)実装している。それぞれのコントローラは独立して動作し、各コンポーネントを制御する。ただし、コントローラ間で協調動作ができないとプログラム実行の進捗を管理できない。そのため、3つのコントローラはバリア同期が取れる仕組みを有する。

このコントローラはRISC-V[17]風プロセッサとして実装している。RV32I命令セットから必要な物だけを実装した実装であることに加えて、RISC-V本来の挙動に準拠していない命令がいくつかあるため「RISC-V風」としている。また、この理由により、C言語などのコンパイラを使うことは出来ないため、アセンブラを記述し、RISC-V対応のGNU asでマシン語を生成している。PCIe BAR0空間の一部がコントローラの命令メモリに接続されており、命令の書き込みはホストから行える。

5.7 Kernel

図2の“Kernel”はアプリケーションカーネルを意味する。この部分は、Intel High-Level Synthesis Compiler (i++) 高位合成処理系を用いて記述することを前提としている。どのようなコードをi++コンパイラでコンパイルすれば、本システム上で実行できるのかについては次章で詳細を述べる。

i++が生成するハードウェアモジュールの制御用インターフェイスは標準的な物が採用されている。したがって、ハードウェア記述言語 (Verilog など) を用いて記述することも可能である。しかしながら、HPCアプリケーションは複雑なため、全体を低レベルな言語で記述することはコスト面から困難であると考えており、高位合成処理系を用いることを前提として本システムは開発を行っている。

5.8 制限事項

本システムは、開発途中であり、様々な制限がある。

- PCIe経由のアクセスは、32バイトアライメントされたアドレスにしか行えない
- PCIe経由のアクセスは、アクセス長は4の倍数でなければならない
- LS→メモリへの書き込み完了を検出できない
- 実機で動作確認できているメモリモジュールはBRAM

^{*2} 例えば、パケット処理回路を現在の2倍の周波数で動作させるといった対応が考えられる。

のみ

5.4節で述べた通り、外部メモリ部分の実装として、BRAMとHBM2が切り替えられる実装となっている。この2つの実装の内、現時点で実機で動作しているのはBRAMのみである。HBM2はRTLシミュレーション(ModelSimを利用)で動作確認をしているが、実機では動作出来ない*3。今後、HBM2メモリの実機動作にむけてデバッグを進めていく。

AXIバスは、書き込みリクエストに対して、完了レスポンスが生成される仕様となっている。しかしながら、現実装では、レスポンスをLS側に転送しておらず、LS側で書き込み完了を検出できない。LSから書き込みQueueへリクエストを送信した時点で、書き込み完了として扱っている。本原稿の実験範囲では、これらの制限が問題となっていないが、複雑なアプリケーションを実装する上では問題となる可能性があり、今後解消できるように実装を進めていきたい。

6. プログラミング環境

6.1 Intel High-Level Synthesis Compiler

本システムでは、前章で述べたとおり、i++を用いてカーネルをプログラミングする。i++はC++コードを入力とし、ハードウェアを生成するコンパイラである。i++はあるC++関数からVerilog HDLのモジュールを生成するコンパイラであり、周辺回路は含まれない。したがって、i++から生成されたモジュールだけを使って実機動作できるというシステムではない。生成されたモジュールを制御する回路や、必要に応じてメモリアクセス回路(コントローラやインターコネクトなど)はユーザーが別途追加しなければならない。

Intel FPGAにおける高位言語の処理系として、Intel FPGA SDK for OpenCL[12]もある。こちらは、i++と違って、FPGAボードにおけるファームウェアに相当するBoard Support Package (BSP)をベースとして、ボード全体のハードウェアを構築するシステムである。したがって、OpenCLコードだけで実際にFPGA上で動作できるハードウェアが構築出来るが、一方で関数のモジュールだけを抜き出すといった処理が出来ない。そのため、本システムでは、関数単位でモジュールとして生成できるi++コンパイラを用いる。なお、両コンパイラでバックエンドは同一のものを利用しているため、生成されるハードウェアの構造や質について違いはないと考えて良い。

*3 FPGAにおけるRTLシミュレーションは実機動作を高精度に再現できるが、100%の再現度ではない。また、コストの問題から簡略化されたモデルでシミュレーションされている場合もあり、シミュレーションと実機動作が異なることがある。

6.2 intadd.cpp

本稿の性能評価には、単純なint配列同士の加算を行うプログラムを用いる。プログラムのソースコード(intadd.cpp)を図3に示す。このコードは、以下の3つの部分から構成される。

(1) パラメータ、型の宣言(5行 - 29行)

(2) uint32_t変数を8個まとめた型(uint8)の宣言(31行 - 44行)

(3) カーネル本体(46行 - 57行)

i++コンパイラはcomponentと修飾された関数(46行)を起点として、ハードウェアモジュールを作成する。component関数がihc::mm_master型の引数を持っていると、Avalon-MM (Memory Mapped)バスとしてハードウェア上で実装される。その際のアクセス特性(レイテンシやバス幅など)をテンプレート引数として与える。図3のコードでは、alias templatesを用いて、read_port, write_portとしてihc::mm_masterを扱えるようにしている。

実際のコードでは、in0, in1, out0の3つの引数がkernel関数にあり(48行 - 50行)、これらがLSに接続されるメモリバスとなる。そして、これらの変数に対して配列のようにアクセスを行うと(54行)、LSに対してメモリアクセスが発行される。

6.3 ホストからの制御方法

ホストOS(Linux)には、独自のドライバーモジュールをカーネルに組み込み、このドライバが低レベルな制御を行う。本システムのドライバは、FPGAが持つPCIeアドレス空間をユーザープロセスにmmap(2)できるように実装しており、mmap(2)で得られた仮想アドレスに対してユーザープロセスからメモリアクセスを行うと、PCIeバスを通じてFPGAと通信が行われる。

現時点では、DMA転送に対応していないため、CPU-FPGA間のデータ転送はすべてMemory Mapped I/O(MMIO)を用いて行う。ここでは、FPGA側にDMAコントローラがないため、DMA転送に非対応としているが、FPGA以外のDMAコントローラを用いれば、DMA転送ができると思われる。ただし、動作検証を行っていないため、正常に動作するかは不明である。

6.4 実行方法

LSの容量には限りがあるため、計算に必要なデータが一度にLSに格納できない場合、問題をLSに収まるサイズに分割し、小分けして計算を行う。intadd.cppは2つの配列を読んで、1つの配列に書き込むプログラムであるが、こういうメモリアクセスパターンの場合、図4にあるように4重バッファリングを行う。各計算フェイズの間は、コントローラ間のバリア同期によって順序制御を行う。

intadd.cppの実装では、LS0, LS1から読み込み、LS0に

```

1  #include "HLS/hls.h"
2  #include <stdint.h>
3  #include <stdio.h>
4
5  unsigned constexpr N_PORTS = 2;
6
7  template <class T, int Aspace>
8  using read_port = ihc::mm_master<
9      T,
10     ihc::latency<4>,
11     ihc::dwidth<256>,
12     ihc::awidth<20>,
13     ihc::aspace<1 + Aspace % N_PORTS>,
14     ihc::maxburst<1>,
15     ihc::align<256>,
16     ihc::readwrite_mode<readonly>,
17     ihc::waitrequest<false>>;
18
19  template <class T, int Aspace>
20  using write_port = ihc::mm_master<
21      T,
22      ihc::latency<2>,
23      ihc::dwidth<256>,
24      ihc::awidth<20>,
25      ihc::aspace<1 + N_PORTS + (Aspace %
26          N_PORTS)>,
27      ihc::maxburst<1>,
28      ihc::align<256>,
29      ihc::readwrite_mode<writeonly>,
30      ihc::waitrequest<false>>;
31
32  struct uint8 {
33      uint32_t s[8];
34  };
35
36  uint8 operator+(
37      uint8 const& lhs, uint8 const& rhs)
38  {
39      uint8 r;
40      #pragma unroll
41      for (int i = 0; i < 8; i++) {
42          r.s[i] = lhs.s[i] + rhs.s[i];
43      }
44      return r;
45  }
46
47  component int kernel(
48      int length,
49      read_port<uint8, 0>& in0,
50      read_port<uint8, 1>& in1,
51      write_port<uint8, 0>& out0)
52  {
53      #pragma ii 1
54      for (int i = 0; i < length; i++) {
55          out0[i] = in0[i] + in1[i];
56      }
57      return 0;
58  }

```

図 3: カーネルコードの例 (intadd.cpp).

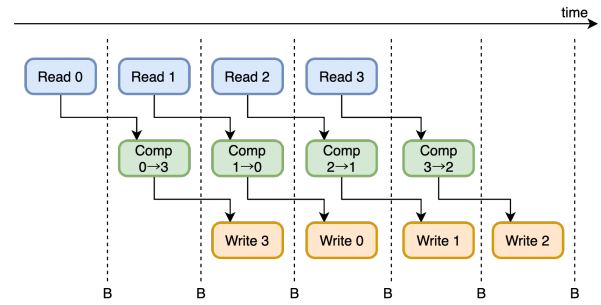


図 4: 計算の流れ.

表 1: 評価環境 (PPX)

CPU	Intel Xeon E5-2690 v4 × 2
CPU Memory	DDR4 2400 MHz 64 GB (8 GB × 8)
Infiniband	Mellanox ConnectX-4 EDR
Host OS	CentOS 7.3
Host Compiler	gcc 4.8.5
FPGA	インテル Stratix 10 MX FPGA 開発キット (1SM21CHU2F53E1VG)
FPGA Memory	HBM2 16GB (8GB × 2)
Quartus Prime	Quartus Prime Pro 20.4.0.72

結果を書き込むため、読み書きの領域も重複してはいけない。したがって、トリプルバッファリングでは不十分であり、4重バッファリングを採用している。現在の実装では、LSあたりの容量は128KBであるから、32768個のuint32_t型を格納できる。したがって、LSを8192個のグループx4に分け、4重バッファリングを行う。

また、5.8節で述べたとおり、現在の実装ではLS→メモリへの書き込み完了を検出できない。今回の計算は、100μs未満で完了することがわかっているため、ホスト側でsleep(3)を用いて1秒待機することで代替している。

7. 性能評価

7.1 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実験クラスター Pre-PACS-X (PPX) を性能評価に用いる。PPXは開発用クラスターであるため、様々な仕様のノードが混在しているが、その中の1ノード(表1)にインテル Stratix 10 MX FPGA 開発キット [18] を搭載し、性能評価を行う。

開発キットは、1SM21CHU2F53E1VGを搭載しており、Speedgrade-1のロジック部と、16GBのHBM2を搭載したMCMとなっている。Intel FPGAのHBM2帯域はチップのSpeedgradeによって決まり、Speedgrade-1の場合、最大で512GB/s(コントローラ動作周波数500MHz)まで対応する。ただし、今回の実験ではシステム全体を250MHz

表 2: 性能測定結果.

メモリ種別	書込量	読込量	転送時間
BRAM	65536 B	131072 B	7356 cycles, 29.4 μ s

表 3: バンド幅および効率.

メモリ種別	帯域	理論ピーク	効率
BRAM	6.687 GB/s	8.0 GB/s	83.6%

でコントローラを駆動している. 過去の研究 [9] でコントローラの 500MHz 駆動を行うには細かいチューニングが必要となることがわかっており, 動作周波数のチューニングを行うよりも, システムの実機動作を示すことを優先したためである. なお, 250MHz は PCIe IP の動作周波数由来であり, これと同じ周波数と設定した.

7.2 評価方法

性能評価には, 前章で示した `intadd.cpp` (図 3) を用いる. ちょうど LS の容量を使い切るように, 長さ 32768 個の配列を用いて計算を行う. したがって, ちょうど図 4 と同じ計算シーケンスとなる. また, 5.8 節で述べたように, 現時点では HBM2 を実機で利用できないため, BRAM モジュールで性能を測定する.

今回のカーネルは, メモリバンド幅律速となる設計である. 演算 (加算) は `int8` 型を用いて $32 \times 8 = 256\text{bit}$ の演算が毎クロックサイクル出来る実装である. そのため, チャンネル 0 のメモリバスの性能測定できるカウンターを実装し, 性能評価を行う. 転送時間は最初にメモリバスへのデータアクセスが発生^{*4}してから, 最後のデータアクセスが発生するまでの間と定義する. チャンネル 0 と 1 の間にメモリアクセスパターンやアクセス量の違いはないため, 0 のみの測定としている.

今回の測定条件では, ホストと FPGA 間の同期に関係するコストは含まれていない. 書き込み完了の検出が未実装なため, 同期に関するオーバーヘッドを測定できないためである. カーネルの動作開始には, MMIO Write を 4 回 (パフォーマンスカウンタのリセットに 1 回, コントローラ x3 の起動に 3 回) 行う必要がある. なお, 図 4 は 6 ステップに計算が分かれているが, 最初に各種コントローラの動作が開始された後はホストからの制御は必要ない. コントローラ間のバリア同期によって自律的に計算が進行する.

7.3 評価結果

測定結果を表 2 と表 3 に示す. 今回の実装では, メモリバスは 250MHz で動作しており, バス幅は 256bit である. したがって, 1 クロックサイクルの実時間は 4ns, メモリバスの理論ピークは 8.0 GB/s となる.

^{*4} AXI バス上で (`wvalid && wready`) || (`rvalid && rready`)

196608 バイト転送するのに 7356 サイクルかった. 1 サイクルあたり 4ns なので, 転送時間は 29.4 μ s となり, バンド幅は 6.687 GB/s となる. また, 8.0GB/s のピークに対して, 6.687 GB/s の帯域が得られており, これは 83.6% の効率となる. 今回の実験ではチャンネル 0 のデータバスしか測定していないが, チャンネル 0 と 1 に対しての負荷は同じであるため, システム全体では 13.374 GB/s の帯域が得られていると考えられる.

8. 考察

性能評価では, BRAM 利用時に 83.6% のメモリ転送効率を得られた. BRAM は SRAM で実装されているので, リフレッシュやメモリアクセスパターンで性能に変化がない. したがって, この効率の値はそのままメモリシステムの効率を表していると考えられる. HBM2 を直接扱う場合, シーケンシャルアクセスで 93% 程度の効率を得られることがわかっており [9], メモリシステムでも同程度の効率を達成することが目標値となると考えている.

転送効率が低下している理由の一つに, 各フェイズでバリア同期を実施している点があると考えられる. この実装方式では, 各フェイズが完了するのを待ってから次フェイズが開始される. すなわち, メモリアクセスやカーネルパイプラインの起動にかかるレイテンシを隠蔽できない. 図 4 を例に挙げると, Read 0 が完了したら Read 1 を開始できるべきであるが, バリア同期があるためそれが出来ない. また, カーネル起動についても同様で, 対応する Read が完了した時点で起動出来た方がよい. `i++` コンパイラするモジュールには, リクエストキューが含まれており, 複数の起動リクエストを投入できる.

また, アプリケーションの実装上の問題も効率を下げる要因の一つであると考えられる. `intadd.cpp` は, 2 つの LS にデータを読み込み, 1 つの LS に結果を書き込み, 1 つの LS からメモリに書き込む動作を行う. メモリとクロスバー間および LS とクロスバー間のバス幅は同じ 256bit である. 読み込み時は LS 側の帯域とメモリ側の帯域が釣り合っており, メモリの帯域を使い切れるが, 書き込み時は半分の帯域しか使えない. したがって, 100% の帯域を得ることは理論上出来ない実装となってしまう. この問題を解決するには, 計算結果を LS0・LS1 の両方に書き込む実装に変更しなければならない.

最後に, 扱うメモリチャンネルを増やす際の動作周波数について検討を行う. 本稿で扱う範囲 (最大で $N=4$) では問題となっていないが, 今後, 多数のメモリチャンネルを実装した際に, クロスバーが実装上のボトルネックとなる可能性は高い. 一般的に, クロスバー接続の複雑度は N^2 であり, 接続するモジュールを多くすると二乗のオーダーでリソース消費が増加すると考えられる. FPGA において 32 ポートクロスバーは現実的ではないと考えており, 8

チャンネルを1つのグループとしてクロスバーを配置することを検討している。この場合、クロスバー間の接続は例えばリングなど複雑度の低いトポロジーを採用する。Intel FPGA では、HBM2 から接続されている配線が4グループに分かれてチップ内に配置されているため、8チャンネル毎にグループ化することは理にかなっている。クロスバー単体で8x8まで動作周波数を目標値(400MHz)以内に維持したままFPGA上に実装できることは確認できている。しかしながら、実際の実装では、クロスバー以外のモジュールも実装され、理想的な配置配線が出来るとは限らないため動作周波数が低下する可能性があり、今後検証しなければならない。

9. まとめと今後の課題

本稿では、HPCとHBM2向けメモリシステムを提案しそのプロトタイプ実装を示した。まだ、HBM2を利用した実機動作ができていないという問題はあるものの、BRAMを代用した時に6.687 GB/sの帯域と83.6%のメモリ効率を得られた。8章で考察したとおり、まだ最適化を行い効率を改善できる要素が残っていると考えられる。しかしながら、メモリシステムの提案および、プロトタイプ実装の初期評価を行うという本稿の目的は達成できたものと考えている。

今後の課題としては、HBM2の実機動作を実現し、HBM2の全体を扱えるようにメモリチャンネルを増やすことがあげられる。8章で述べたとおり、クロスバーの実装がボトルネックとなる可能性が高い。加えて、メモリバスの動作周波数を向上させたいと考えている。メモリコントローラ側の最大動作周波数が500MHzであるため、今後のターゲット周波数は400~500MHzを考えている。また、PCIe Gen.3 x16レーンの対応、DMA転送対応、FPGA間直接通信、Partial Reconfiguration (PR)なども今後対応していきたいと考えている。

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。本研究の一部は、JSPS 科研費 21H04869 の助成を受けたものである。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 社の支援に謝意を表す。

参考文献

[1] 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: Stratix 10 FPGA を用いた ray-tracing 法による輻射輸送計算の高速化, 研究報告ハイパフォーマンスコピューティング (HPC), 2020-HPC-175 (2020).
[2] 柏野隆太, 小林諒平, 藤田典久, 朴 泰祐: 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, 研究報告ハイパフォーマンスコピューティング (HPC), 2020-HPC-175 (2020).
[3] NVIDIA: VIDIA A100 | NVIDIA, <https://www.nvidia.com/ja-jp/data-center/a100/>.
[4] Meyer, M., Kenter, T. and Plessl, C.: Evaluating FPGA Accelerator Performance with a Parameterized OpenCL Adaptation of Selected Benchmarks of the HPCChallenge Benchmark Suite, *2020 IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC)*, pp. 10–18 (online), DOI: 10.1109/H2RC51942.2020.00007 (2020).
[5] Venkataramanaiah, S. K., Suh, H. S., Yin, S., Nurvitadhi, E., Dasu, A., Cao, Y. and Seo, J. S.: FPGA-based Low-Batch Training Accelerator for Modern CNNs Featuring High Bandwidth Memory, *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–8 (2020).
[6] Kuramochi, R. and Nakahara, H.: An FPGA-Based Low-Latency Accelerator for Randomly Wired Neural Networks, *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, pp. 298–303 (online), DOI: 10.1109/FPL50879.2020.00056 (2020).
[7] 埴 敏博, 三木洋平: 宇宙物理アプリケーションのための FPGA 演算オフローディングの検討, 研究報告ハイパフォーマンスコピューティング (HPC), 2020-HPC-172 (2019).
[8] kyu Choi, Y., Chi, Y., Qiao, W., Samardzic, N. and Cong, J.: HBM Connect: High-Performance HLS Interconnect for FPGA HBM, *FPGA '21* (2021).
[9] 藤田典久, 小林諒平, 山口佳樹, 朴 泰祐: HBM2 メモリを持つ FPGA ボードの性能評価, 研究報告ハイパフォーマンスコピューティング (HPC), 2021-HPC-178 (2021).
[10] Bachrach, J., Vo, H., Richards, B., Lee, Y., Waterman, A., Avizienis, R., Wawrzyniek, J. and Asanović, K.: Chisel: Constructing hardware in a Scala embedded language, *DAC Design Automation Conference 2012*, pp. 1212–1221 (online), DOI: 10.1145/2228360.2228584 (2012).
[11] Izraelevitz, A., Koenig, J., Li, P., Lin, R., Wang, A., Magyar, A., Kim, D., Schmidt, C., Markley, C., Lawson, J. and Bachrach, J.: Reusability is FIRRTL ground: Hardware construction languages, compiler frameworks, and transformations, *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 209–216 (online), DOI: 10.1109/ICCAD.2017.8203780 (2017).
[12] Intel: Intel FPGA SDK for OpenCL, <https://www.intel.com/content/www/us/en/software/programmable/sdk-for-opencl/overview.html>.
[13] 藤田典久, 小林諒平, 山口佳樹, 上野知洋, 佐野健太郎, 朴 泰祐: 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, 研究報告ハイパフォーマンスコピューティング (HPC), 2020-HPC-173 (2020).
[14] 小林諒平, 藤田典久, 中道安祐未, 山口佳樹, 朴 泰祐, 吉川耕司, 安部牧人, 梅村雅之: OpenCL 対応 GPU・FPGA デバイス間連携機構による宇宙輻射輸送コードの演算加速, 研究報告ハイパフォーマンスコピューティング (HPC), 2019-HPC-172 (2019).
[15] Intel: Intel L- and H-tile Avalon Streaming and Single Root I/O Virtualization (SR-IOV) IP for PCI Express User Guide, <https://www.intel.co.jp/content/www/jp/ja/programmable/documentation/1b11464284700752.html#1b11466722478760>.

- [16] Platform Designer: <https://www.intel.co.jp/content/www/jp/ja/programmable/products/design-software/fpga-design/quartus-prime/features/qts-platform-designer.html>.
- [17] RISC-V International: <https://riscv.org/>.
- [18] Intel: インテル (R) Stratix(R) 10 MX FPGA 開発キット, https://www.intel.co.jp/content/www/jp/ja/programmable/products/boards_and_kits/dev-kits/altera/kit-s10-mx.html.