Vol.2019-HPC-171 No.2
2019/9/20

FPGAによる次世代メモリのエミュレーション機構の試作

広渕 崇宏[1,a)]　高野 了成[1]

概要：DRAMとは異なる新たなメモリデバイスが計算機のメインメモリの一部に搭載されつつある。このようなメインメモリに対する新たなシステムソフトウェアの研究が注目されている。メインメモリの性能特性を擬似的に再現する（エミュレーションする）機構が求められるものの、既存技術ではシステムソフトウェアの研究に適したものは存在しない。本稿では、FPGAを用いた次世代メモリのエミュレーション機構を提案する。メインメモリの遅延や帯域幅を様々な値に変更しながら、システムソフトウェアをエミュレータ上で実行できる。そのような大規模なソフトウェアに対しても十分高速に動作する。異なる種類のメモリデバイスがメインメモリ内で併存するハイブリッド型メモリシステムの再現も可能にする。一般的に安価に入手可能なFPGA SoCを用いることで容易に導入可能である。初期的なプロトタイプを実装し予備的な評価を行った。エミュレータを介したメモリアクセスの最小遅延は370 ns程度であった。現実的な実行時間で大規模なソフトウェアをエミュレータ上で動作できると期待される。提案エミュレータでメモリの読み込み遅延および書き込み遅延を変更すると、CPUから観測できるメモリの遅延および帯域幅が正しく変化することを確認した。

1. はじめに

今日の計算機において、メインメモリとして用いられるDRAMは技術的な限界を抱えている。キャパシタを記憶素子とするDRAMは、その値を維持するために常にリフレッシュ電力を必要とする。その消費電力は計算機全体の数割を占めており、さらなる大容量化は難しい。また、製造プロセスが微細化するにつれてリーク電力の問題が深刻化しつつあり、高密度化は徐々に困難になりつつある。

そこで、DRAMとは異なる新たな動作原理に基づくメモリデバイスに注目が集まっている。例えば、2019年4月に発売されたIntelのOptane Data Center Persistent Memory (DCPM) は、抵抗変化する記憶素子を利用したメモリモジュールであり、メモリモジュール一枚あたりDRAMよりも一桁大きい記憶容量を提供する。従来のDRAMモジュール同様にDIMMインタフェースを介して計算機のメモリバスに接続し、そのメモリ空間をCPUの物理メモリアドレス空間に直接マップできる。ソフトウェアはCPUのload/store命令などを利用してOptane DCPMをDRAM同様に利用できる。

しかし、新たな動作原理に基づくメモリデバイスは、DRAMとは異なる性能特性を有する。例えば、我々の評価実験 [1] では、Optane DCPMのランダムアクセスの読み込み遅延は374 nsであり、DRAMよりも4倍程度遅かった。また、ライトバックを伴うランダムアクセスの遅延は391 nsであり、DRAMよりも4.1倍程度遅かった。読み込み帯域幅は、6枚のメモリモジュールがインタリーブされた状態において37 GB/sであり、DRAMの3分の1程度であった。また、ライトバックを伴うメモリアクセスの帯域幅は2.9 GB/sであり、DRAMの一割未満であった。計測方法や計測環境が若干異なるものの、Optane DCPMがDRAMとは異なる性能特性を有することが他からも報告されている [2], [3]。

DRAM同様にアクセスできるにもかかわらずDRAMとは異なる性能特性を有するメモリデバイスをシステムソフトウェアからどのように使いこなすのか、今後研究が盛んになることが期待される。どのような性能特性を有するメモリデバイスに対して、どのようなメモリ管理手法が有効であるのか、明らかにする必要になる。実際に入手できるメモリデバイスのみならず、将来登場するであろうメモリデバイスの性能特性を想定して、評価実験を行えることが望ましい。評価実験においては、メモリデバイスの性能特性を変えながら、ソフトウェアの有効性を定量的に検証できることも望ましい。

しかしながら、メモリデバイスの性能特性を擬似的に再現する既存機構は、システムソフトウェアの研究に用いるためには不十分である。比較的大規模なソフトウェアであるシステムソフトウェアを、エミュレータ上で評価実験に

[1] 国立研究開発法人 産業技術総合研究所
[a)] t.hirofuchi@aist.go.jp

支障を来さない現実的な速度で実行できる必要がある。今後主流になるであろうハイブリッド型のメインメモリを想定して、異なる性能特性を有するメモリデバイスが併存するメインメモリを再現可能なものが求められる。評価実験に広く用いることが可能な導入の容易さを備えることも必要である。

そこで、本稿では、FPGA を用いた次世代メモリのエミュレーション機構を提案する。DRAM とは異なる性能特性を有するメインメモリを擬似的に再現可能にする。システムソフトウェアをエミュレータ上で実行可能し、そのような大規模なソフトウェアであっても十分高速に動作する。異なる種類のメモリデバイスがメインメモリ内で併存するハイブリッド型メモリシステムの再現も可能にする。一般的に安価に入手可能な FPGA SoC を用いることで容易に導入可能である。

本稿の構成は以下の通りである。2 節では、既存のシミュレーションおよびエミュレーション手法を概観し、新たなメモリエミュレーション手法の必要性を述べる。3 節で要求事項をまとめ、4 節で提案機構を紹介する。5 節でプロトタイプの初期的な評価実験を述べる。6 節で関連研究にふれ、7 節で本稿をまとめる。

2. 背景

DRAM とは異なるメモリデバイスがメインメモリの一部に搭載されつつある。従来とは異なるメモリ構成および性能特性を有するメインメモリを管理する手法の研究が重要になる。例えば、メインメモリとして搭載された、CPU の load/store 命令でアクセスできる（byte-addressable な）不揮発性メモリを対象としたファイルシステムの提案 [4], [5] や、同様のメモリを NUMA ノードとして管理する手法の提案がある [6]。byte-addressable な不揮発性メモリに対して、突然の電源遮断時にも不整合が生じることがないよう安全にメモリオブジェクトを割り当てて管理するライブラリの提案 [7] やプログラミングインタフェース [8] がある。また、我々は、DRAM と DRAM とは異なる性能特性を持つメモリデバイスが併存するメインメモリを想定して、メモリマッピングを動的に最適化するハイパーバイザ（RAMinate[9], [10]）を提案している。メモリアクセスが頻出するメモリページを判別し、より高速なメモリデバイス（一般には DRAM）に再配置する。ハイパーバイザとして実装することで、ゲストオペレーティングシステムを一切改変する必要がない。

しかしながら、メモリデバイスの性能特性を擬似的に再現する既存機構は、このようなシステムソフトウェアの研究に用いるためには不十分である。

プロセッサシミュレータ（例えば gem5[11]）およびそれと併せて用いられるメモリシミュレータ（例えば NVMain[12]）は、計算機アーキテクチャに焦点を当てた研究においては有効であるものの、システムソフトウェアのような大規模なソフトウェアに焦点を当てた研究には適用が難しい。ハードウェアの細部をシミュレーションするため膨大な実行時間が必要になる。

PIN などの binary instrumentation 手法は、プログラムの CPU 命令列を逐次分析・翻訳しながら実行するため、対象とするプログラムの性能低下が生じる。カーネル空間も含めたオペレーティングシステム全体を対象とすることは一般的には想定されていない。

対象プログラムの性能低下が軽微な手法として、従来の DRAM を搭載した計算機を用いつつ、DRAM よりもアクセス遅延が大きいメモリデバイスをメインメモリに用いた場合を想定して、対象プログラムの実行速度をわざと遅くする機構が提案されている。プロセッサのパフォーマンスカウンタから取得できる情報を用いて、LLC ミス（すなわちメインメモリアクセス）に伴う CPU のストールサイクル数をリアルタイムに計測し、想定するメモリデバイスのアクセス遅延を考慮して、実行プログラムの CPU 時間割り当てを減少させる。例えば、我々の提案する手法 [13] は、CPU のライトバック処理を監視することで、読み書き遅延が異なるメモリデバイスをエミュレーションした場合であっても高い精度で実行プログラムの挙動を再現できる点で、先行研究（Quartz [14]）よりも優位性がある。比較的導入が容易である反面、パフォーマンスカウンタから取得できる情報は限られるため、エミュレーション可能なメモリ構成は限られる。DRAM に加えて Optane DCPM を物理アドレスにマップした場合のように、物理アドレス領域のメモリの性能特性が一様ではない状況を再現することは困難である。また、評価結果を多面的に検証するためには、このエミュレーション手法を用いた評価実験に加えて、異なる手法に基づく評価実験も併せて行うことが望ましい。

3. 要求事項

以上を踏まえて、我々が求めるエミュレーション機構に対する要件を整理すると以下のようになる。

- DRAM とは異なる性能特性を持つ byte-addressable なメインメモリを擬似的に再現し、その読み書き遅延などのパラメタを任意の値に設定できること。
- オペレーティングシステムなどのシステムソフトウェア全体をエミュレータで動作させることが可能であり、そのような大規模かつ複雑なソフトウェアを対象に現実的な速度でエミュレーションを提供できること。
- 異なる種類のメモリデバイスがメインメモリ内で併存するハイブリッド型メモリシステムの再現が可能であること。

一方、メモリデバイスの不揮発性に関しては、当面エミュレーション機構の対象外とし、遅延や帯域幅の再現に

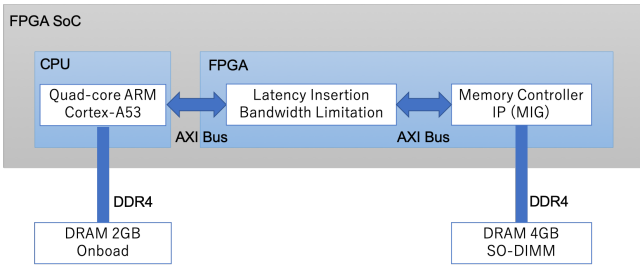

図 1 提案機構の概要

焦点を当てる。

4. 提案機構

そこで、本稿では、FPGA を用いた次世代メモリのエミュレーション機構を提案する。DRAM とは異なる性能特性を有するメインメモリを擬似的に再現可能にする。システムソフトウェアをエミュレータ上で実行可能にし、そのような大規模なソフトウェアを対象としても十分高速に動作する。従来のソフトウェアによるエミュレーション手法で難しかったハイブリッド型メモリシステムの再現も可能にする。一般的に安価に入手可能な FPGA SoC を用いることで容易に導入可能である。

提案機構の概要を**図 1** に示す。FPGA SoC とは、CPU と FPGA がワンチップにパッケージ化されたものであり、CPU ではオペレーティングシステムなどのソフトウェアを動作させつつ、FPGA をプログラムすることで様々な機能をハードウェアで実装することが可能である。CPU は load/store 命令などを用いて CPU 側に接続された DRAM にアクセスできる。提案機構においては、CPU が、FPGA 上でプログラムされたエミュレーションモジュールを介して、FPGA 側に接続された DRAM にもアクセスできるよう設計した。FPGA 側に接続された DRAM も CPU の物理アドレス空間にマップされており、load/store 命令などを用いてアクセスできる。

エミュレーションモジュールにおいては、遅延の挿入や帯域幅の制限などを行って、DRAM とは異なる性能特性を有するメモリの挙動を再現する。ハイブリッド型のメインメモリを想定して、メモリアクセス要求のアドレスに応じて、異なる遅延を挿入することなども可能である [*1]。CPU コアは、CPU 側に接続された DRAM および FPGA 側に接続された DRAM に対して CPU キャッシュを介して接続する。特段の言及がない限り CPU キャッシュが機能するように設定した。

提案機構の仕組み上、想定するメモリデバイスの遅延や帯域幅の値そのものをエミュレータ上で再現することが難しい場合がある。その場合であっても、提案機構を用いれば遅延や帯域幅が変わった場合の影響を相対的に評価することが可能である。

現在、Xilinx 社製の FPGA SoC 開発ボードである Zynq Ultrascale+ ZCU104 を用いて、初期的なプロトタイプを実装中である。これまでにエミュレーションモジュールにおいては遅延の挿入機能のみを実装した。CPU とエミュレーションモジュール間は AXI バスを通じて接続されている。またエミュレーションモジュールとメモリコントローラ（MIG）間も AXI バスを通じて接続されている。エミュレーションモジュールは、両 AXI バス間でデータを透過的に転送するとともに、必要に応じてデータ転送を指定のクロック数だけ遅らせる。AXI バスは 5 つの通信チャネルから構成されており、通信を開始するマスター側とそれを受けるスレーブ側とが接続されている。エミュレーションモジュールから見ると、CPU がマスターに相当し、メモリコントローラがスレーブに相当する。スレーブからの読み込みデータを転送するチャネル（R チャネル）とスレーブに対する書き込みの成否をスレーブから通知するチャネル（B チャネル）に対して、それぞれ遅延の挿入機能を実装した。FIFO キュー内にてチャネル内を通過するメッセージを一定時間遅延させる。一度に複数のメモリアクセス要求が発行された場合に備えて、非同期的な転送に対応する。

5. 予備評価

実装中のプロトタイプを用いて予備的な評価を行った。評価実験は暫定的なものであり、開発の進捗とともに結果が変わる可能性がある。開発ボードに対して ARM 64bit 版 Linux の実行環境を構築した。オペレーティングシステムに対して CPU 側に接続された DRAM をメインメモリとして認識させた。FPGA 側に接続された DRAM はメインメモリとしては認識されていないものの、メモリ空間の物理アドレスにはマップされている。ユーザランドで動作する計測プログラムは、物理アドレスへの直接的なアクセスを提供するデバイスファイル`/dev/mem` を利用して、FPGA 側のメモリを読み書きした。一般的に Linux カーネルはメインメモリとして認識している物理アドレス以外の領域については、CPU キャッシュを無効にする。Linux カーネルを改変し、FPGA 側 DRAM の物理アドレス領域について、CPU キャッシュの有効と無効を切り替えられるようにした。

本実験で利用する CPU は ARM Cortex-A53 の 4 コア構成である。CPU キャッシュは CPU コアごとの L1 キャッシュが 32 KB および CPU コアで共有する L2 キャッシュが 1MB 存在する。

5.1 FPGA 側 DRAM の基本性能（遅延）

FPGA 側 DRAM の基本性能を調査すべく、CPU から FPGA 側に接続された DRAM にアクセスする際の遅延および帯域幅を計測した。この実験ではエミュレータで遅延

[*1] また、本稿では議論しないものの、メインメモリのビット化けを再現することも可能である。

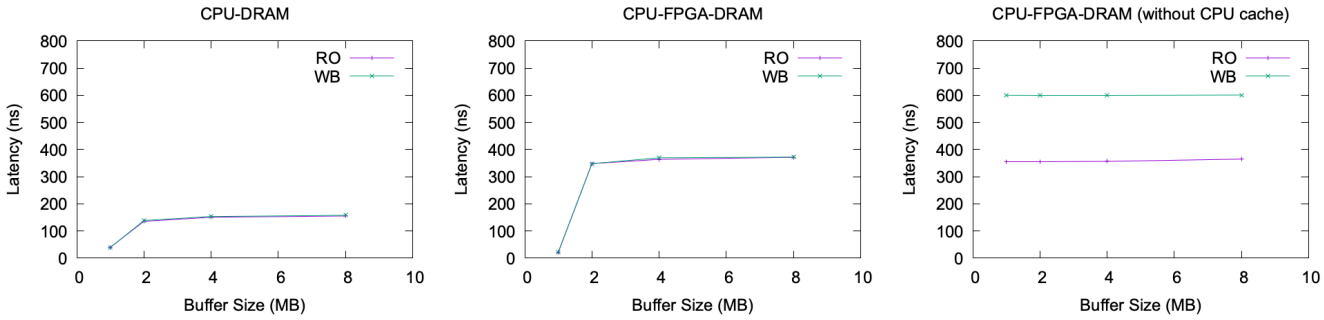

図 2 ランダムアクセスにおけるメモリアクセス遅延。横軸はバッファサイズ。左側より順に CPU 側 DRAM、FPGA 側 DRAM、FPGA 側 DRAM（ただし CPU キャッシュが無効）のグラフ。RO は読み込み動作のみを行うプログラムによるメモリアクセス遅延の計測値、WB は書き込み動作も伴うプログラムによるメモリアクセス遅延の計測値を示す。

は付加しない。計測プログラムは我々が開発したものを用いた [1], [13]。

遅延の計測プログラムは、CPU のプリフェッチや Out-of-order 実行による影響を抑えるため、ランダムなアドレスにジャンプするリンクリストを生成・追跡する。読み込み遅延を計測する動作は次のようになる。

(1) メモリ上に確保したバッファを 64 バイトのキャッシュラインサイズごとに分割する。これをキャッシュラインオブジェクトと呼ぶ。
(2) 各キャッシュラインオブジェクトの先頭 8 バイトに、リンクリストにおける次のキャッシュラインオブジェクトへのポインタを埋め込む。ただし、次のキャッシュラインオブジェクトはランダムな距離だけ離れた場所のキャッシュラインオブジェクトとする。
(3) リンクリストを先頭から末尾まで追跡し、その所要時間より遅延を見積もる。

メインメモリへのライトバックを伴う遅延を計測する場合は、リンクリストを追跡する際に、各キャッシュラインオブジェクトの先頭 8-15 バイト目に任意の値を書き込むことで、CPU のキャッシュ管理においてキャッシュラインを *modified* 状態にする。

図 2 にその実験結果を載せる。確保するバッファのサイズを変更した。CPU 側 DRAM、FPGA 側 DRAM、CPU キャッシュを無効にした場合の FPGA 側 DRAM についてそれぞれ計測した。CPU の LLC が 1 MB であるため、バッファサイズをそれ以上に増やした場合にメモリへのアクセス遅延を計測できる。バッファサイズを 8MB とした時に、CPU 側 DRAM は約 155 ns、FPGA 側 DRAM は約 370 ns であった。FPGA 側 DRAM は FPGA の回路を経由するため遅延の増加はやむえない。メインメモリのエミュレータとして、FPGA 側 DRAM をメインメモリとして見立ててシステムソフトウェアを動作させた場合、CPU 側 DRAM を用いるよりも性能は低下するものの、十分現実的な時間で評価実験を行えると期待する。

なお、必要に応じて CPU キャッシュを無効にできることも併せて確認した。図 2 の右側のグラフが示すように、書き込みを伴うメモリアクセス遅延は増加した。これは、リンクリストのポインタを参照するために生じたメモリからの読み込みと、データのメモリへの書き出しが逐次 FPGA 側 DRAM に対して行われたためである。

5.2 FPGA 側 DRAM の基本性能（帯域幅）

メモリ帯域幅の計測プログラムは、それぞれシーケンシャルアクセスを行う複数のワーカプロセスを起動する。各ワーカプロセスはそれぞれ対象とするメモリ上に 100 MB の領域を確保し、シーケンシャルアクセスを行う。CPU は 4 コアであるため、最大 4 つのワーカプロセスを起動した。実験結果を**図 3** に示す。CPU 側 DRAM の場合、読み込み動作のみの帯域幅は最大約 7 GB/s、ライトバックを伴う帯域幅が約 2.9 GB/s であった。一方、FPGA 側 DRAM の場合、帯域幅はいずれも 0.8 GB/s 程度であった。CPU 側 DRAM よりは帯域幅が低下するものの、システムソフトウェアの動作上は支障がないことが期待される。

一般に、*modified* 状態となったキャッシュラインのメインメモリへのライトバックは、CPU の動作とは非同期に行われるため、CPU からその動作を観測することは難しい。遅延の計測においては、読み込み動作のみのメモリアクセス遅延もライトバックを伴うメモリアクセス遅延もほぼ同じ値であった。しかし、帯域幅の計測においては、CPU 側 DRAM の場合に、メインメモリへの書き込み処理の速度を上回る速度で、CPU のライトバック処理が生じたため、ライトバックを伴うメモリアクセスの帯域が読み込み動作のみの帯域幅よりも低下している。

FPGA 側 DRAM に対して CPU キャッシュを無効にすると、読み込み帯域幅は減少した。これはプリフェッチがなされないことによるものと推測する。

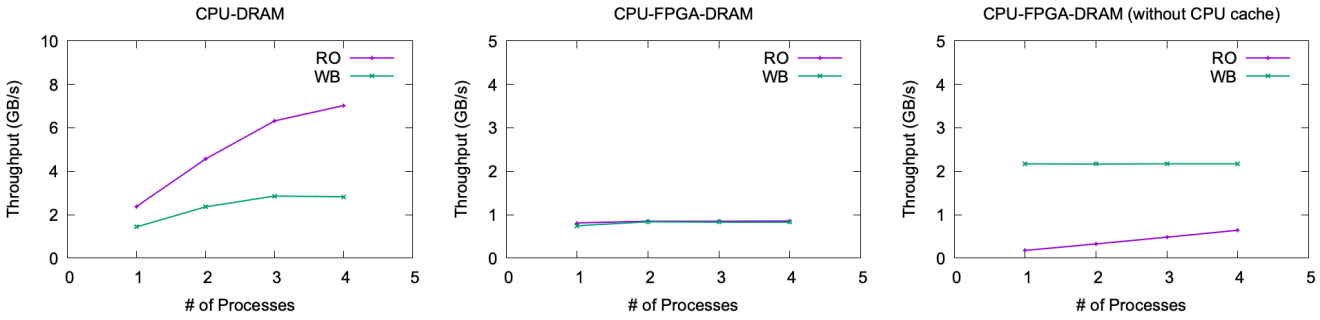

図 3　シーケンシャルアクセスにおけるメモリアクセス帯域幅。横軸はワーカープロセス数。左側より順に CPU 側 DRAM、FPGA 側 DRAM、FPGA 側 DRAM（ただし CPU キャッシュが無効）のグラフ。RO は読み込み動作のみを行うプログラムによるメモリアクセス遅延の計測値、WB は書き込み動作も伴うプログラムによるメモリアクセス遅延の計測値を示す。CPU 側 FPGA のグラフは縦軸の最大値が異なる。

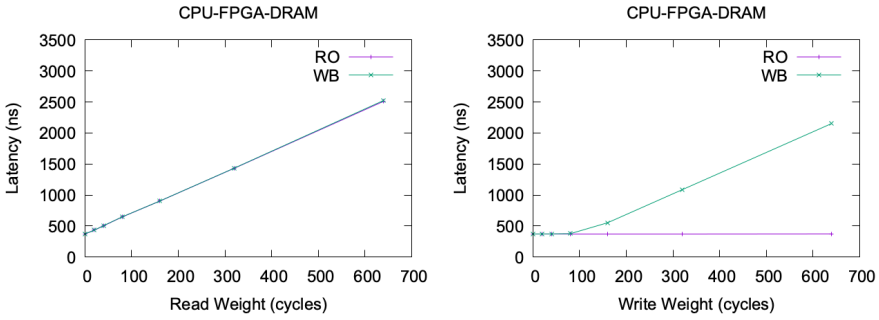

図 4　エミュレータで付加した遅延と計測したメモリアクセス遅延の関係。ランダムアクセス。左側のグラフは読み込み処理に対して遅延を挿入した場合、右側のグラフは書き込み処理に対して遅延を挿入した場合。

5.3　遅延のエミュレーション

同じ計測プログラムを用いて、エミュレータの FIFO キューで挿入した遅延と実際に CPU で計測されるランダムアクセス遅延の関係を調査した。FPGA 側 DRAM の読み込み処理（AXI バスの R チャンル）に対する遅延の挿入および書き込み処理（AXI バスの B チャネル）に対する遅延の挿入をいずれか一方ずつ行った。ランダムアクセスのバッファサイズは 8 MB に設定した。実験結果を**図 4** に示す。

DRAM からの読み込み処理に遅延を挿入した場合、読み込み動作のみのメモリアクセス遅延およびライトバックを伴うメモリアクセス遅延いずれも線形に増加した。計測値から求められる傾きは 3.3 ns/cycle であり、これはこの AXI バスの駆動速度は 300 MHz と合致する。DRAM への書き込み処理に遅延を挿入した場合、読み込み動作のみのメモリアクセス遅延は増加しなかった。ライトバックを伴うメモリアクセス遅延は、挿入した遅延が 80 ns までは増加しなかったものの、その後は線形に増加した。挿入する遅延が増加すると、メインメモリへの書き込み処理が CPU のライトバック動作に間に合わなくなったためである。また、**図 5** に示すように、CPU で観測されるメモリアクセス帯域幅もエミュレータで挿入した遅延に応じて変化することを確認した。

以上、エミュレータで挿入した読み込み遅延ないし書き込み遅延に応じて、CPU で観測されるメモリアクセス遅延が正しく増加することを確認した。それに応じてメモリアクセス帯域幅も減少することを確認した。今後は、エミュレータの FIFO キューを一定時間内に通過するメッセージ数を制限することで、帯域幅を明示的に絞ることも可能にする予定である。

6.　関連研究

FPGA SoC を用いたメインメモリのエミュレータに関する研究を概観する。いずれも FPGA 側に接続された DRAM への遅延を増加させることで DRAM とは異なるメモリデバイスの遅延を再現する。

[15] では、FPGA 側に接続された DRAM の制御タイミング（例えば ACTIVATE 処理をしてから READ/WRITE 処理を行うまでの時間である tRCD など）を意図的に増やすことでメモリアクセス遅延を増加させる。興味深い手法ではあるものの、増加させるとはいえ慎重な扱いが求められる DRAM の制御タイミングを意図的に調整することは、

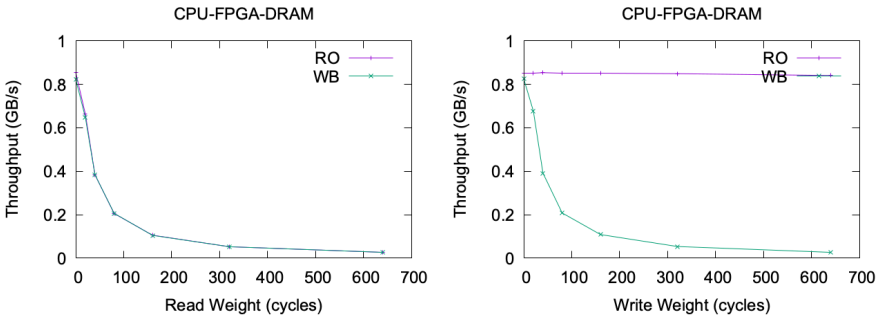

図 5 エミュレータで付加した遅延と計測したメモリアクセス帯域幅の関係。シーケンシャルアクセス。左側のグラフは読み込み処理に対して遅延を挿入した場合、右側のグラフは書き込み処理に対して遅延を挿入した場合。

動作の安定性に支障を来すのではないかと我々は懸念している。また、帯域幅の制限などの発展的なエミュレーションを実装するためには、DRAM の制御タイミングの調整のみでは対応が難しく、我々の手法のようにメモリへのデータを転送する途中にエミュレーションを行うモジュールを実装する方がよいのではと考える。

[16] では、CPU から FPGA 側のメモリデバイスに通じるバスの途中に遅延を挿入する機構を設けている。AXI バスの読み込みアドレスを転送するチャネル（AR チャネル）および書き込みアドレスを転送するチャネル（AW チャネル）において、転送が完了したことをスレーブ側から通知する信号（ARREADY ないし AWREADY のアサート）の伝搬を一定期間遅延させる。アドレス転送用のチャネルに遅延を挿入した理由は明らかではない。一方、我々の手法では、読み込みデータチャネルおよび書き込み成否通知チャネルのメッセージを一定期間 FIFO キュー内にて遅延させている。複数のメモリアクセス要求が同時に発行される場合を想定して、アドレスおよびデータチャネルいずれも非同期的な転送を可能とする。

7. おわりに

FPGA を用いた次世代メインメモリのエミュレーション機構を提案した。DRAM とは異なる性能特性を有するメインメモリを擬似的に再現可能にする。初期的なプロトタイプを実装し予備的な評価を行った。エミュレータを介したメモリアクセスの最小遅延は 370 ns 程度であった。現実的な速度でシステムソフトウェアをエミュレータ上で動作できると期待される。提案エミュレータでメモリの読み込み遅延および書き込み遅延を変更すると、CPU から観測できるメモリの遅延および帯域が正しく変化することを確認した。今後は提案機構の開発を引き続き進めるとともに、エミュレーション機能に関して詳細な評価実験を行う予定である。

謝辞 本研究は科研費 19H01108 の助成を受けた。

参考文献

[1] Hirofuchi, T. and Takano, R.: The Preliminary Evaluation of a Hypervisor-based Virtualization Mechanism for Intel Optane DC Persistent Memory Module, *CoRR*, Vol. abs/1907.12014 (online), available from ⟨http://arxiv.org/abs/1907.12014⟩ (2019).

[2] Izraelevitz, J., Yang, J., Zhang, L., Kim, J., Liu, X., Memaripour, A., Soh, Y. J., Wang, Z., Xu, Y., Dulloor, S. R., Zhao, J. and Swanson, S.: Basic Performance Measurements of the Intel Optane DC Persistent Memory Module, *CoRR*, Vol. abs/1903.05714 (online), available from ⟨http://arxiv.org/abs/1903.05714⟩ (2019).

[3] van Renen, A., Vogel, L., Leis, V., Neumann, T. and Kemper, A.: Persistent Memory I/O Primitives, *CoRR*, Vol. abs/1904.01614 (online), available from ⟨http://arxiv.org/abs/1904.01614⟩ (2019).

[4] Condit, J., Nightingale, E. B., Frost, C., Ipek, E., Lee, B., Burger, D. and Coetzee, D.: Better I/O Through Byte-addressable, Persistent Memory, *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, ACM, pp. 133–146 (2009).

[5] Dulloor, S. R., Kumar, S., Keshavamurthy, A., Lantz, P., Reddy, D., Sankaran, R. and Jackson, J.: System Software for Persistent Memory, *Proceedings of the Ninth European Conference on Computer Systems*, ACM, pp. 15:1–15:15 (2014).

[6] Kannan, S., Gavrilovska, A. and Schwan, K.: pVM: Persistent Virtual Memory for Efficient Capacity Scaling and Object Storage, *Proceedings of the Eleventh European Conference on Computer Systems*, ACM, pp. 13:1–13:16 (2016).

[7] Coburn, J., Caulfield, A. M., Akel, A., Grupp, L. M., Gupta, R. K., Jhala, R. and Swanson, S.: NV-Heaps: Making Persistent Objects Fast and Safe with Next-generation, Non-volatile Memories, *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ACM, pp. 105–118 (2011).

[8] Volos, H., Tack, A. J. and Swift, M. M.: Mnemosyne: Lightweight Persistent Memory, *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ACM, pp. 91–104 (2011).

[9] Hirofuchi, T.: Hypervisor-based Virtualization for Hybrid Main Memory Systems, `https://github.com/takahiro-hirofuchi/raminate`.

[10] Hirofuchi, T. and Takano, R.: RAMinate: Hypervisor-based Virtualization for Hybrid Main Memory Systems,

Proceedings of the Seventh ACM Symposium on Cloud Computing, SoCC '16, New York, NY, USA, ACM, pp. 112–125 (online), DOI: 10.1145/2987550.2987570 (2016).

[11] Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hestness, J., Hower, D. R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M. D. and Wood, D. A.: The Gem5 Simulator, *SIGARCH Computer Architecture News*, Vol. 39, No. 2, pp. 1–7 (2011).

[12] Poremba, M., Zhang, T. and Xie, Y.: NVMain 2.0: A User-Friendly Memory Simulator to Model (Non-)Volatile Memory Systems, *IEEE Computer Architecture Letters*, Vol. 14, No. 2, pp. 140–143 (online), DOI: 10.1109/LCA.2015.2402435 (2015).

[13] Koshiba, A., Hirofuchi, T., Takano, R. and Namiki, M.: A Software-based NVM Emulator Supporting Read/Write Asymmetric Latencies, *To appear in IEICE Transactions on Information and Systems*, Vol. E102-D, No. 12 (online), available from ⟨https://arxiv.org/abs/1908.02135⟩ (2019).

[14] Volos, H., Magalhaes, G., Cherkasova, L. and Li, J.: Quartz: A Lightweight Performance Emulator for Persistent Memory Software, *Proceedings of the 16th Annual Middleware Conference*, Middleware '15, New York, NY, USA, ACM, pp. 37–49 (2015).

[15] 大森　侑，木村啓二：不揮発性メインメモリエミュレータの評価，研究報告組込みシステム（EMB），情報処理学会，pp. 1–8 (2019).

[16] Lee, T., Kim, D., Park, H., Yoo, S. and Lee, S.: FPGA-based prototyping systems for emerging memory technologies, *2014 25nd IEEE International Symposium on Rapid System Prototyping*, pp. 115–120 (online), DOI: 10.1109/RSP.2014.6966901 (2014).