Vol.2021-HPC-180 No.28
2021/7/21

FPGA による次世代メモリのエミュレーション機構

深井 貴明[1,a)]　広渕 崇宏[2,b)]　高野 了成[2,c)]　Akram Ben Ahmed[2,d)]　佐藤 賢斗[1,e)]

概要：次世代のメモリデバイスは、現在一般に利用されるメモリデバイスとは異なる性能特性を持つ。また、今後のメモリデバイスの設計においては、アプリケーションの要求に応えるための性能要件を明らかにする必要がある。このことから、メモリデバイスの性能特性の違いがアプリケーションに与える影響を評価することは重要である。このような評価を行う手法として、ソフトウェアによるアプリケーションへの遅延挿入や CPU シミュレータを用いた方法がある。しかし、前者は精度の面で、後者は実行速度の面で問題がある。そこで我々は FPGA 評価ボード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。先行発表で簡易な試作により基本的な有用性が確認できたことから、その後本格的な開発に着手した。本稿では、提案機構の設計および実装について述べるとともに、その初期評価の結果を報告する。提案機構は、スループット、レイテンシ、およびエラー率の制御が複数のメモリ空間に対して個別に制御できる。また、エミュレーションしたメモリは、物理メモリ空間の一部として、さらには NUMA ノードとして、あるいは NVDIMM としてソフトウェアに見せることができ、用途に合わせてオペレーティングシステムに対する見せ方を切り替えることができる。評価として、まず メモリ性能の制御に関するマイクロベンチマークを行い、メモリ性能の制御が正しく動作することを確認した。また、想定される利用方法の一つとして、レイテンシのみ異なるメモリデバイスのエミュレーションについても評価した。この結果、帯域制限と遅延挿入の組み合わせることで、概ね意図した通りのメモリ性能を再現できることを確認した。

1. はじめに

今日、機械学習や動画像処理など大容量データを計算機で高速に処理する需要が高まっており、それに伴い、メインメモリの性能の重要性も高まっている。このため、次世代メモリデバイスの研究開発が盛んに行われている。例えば、現在普及している DDR4 SDRAM の次世代版となる DDR5 SDRAM や、Intel Opatne DCPM に代表される不揮発性メインメモリなどが存在する。さらにはメモリ素子の信頼性をあえて落とす（すなわちエラー率が上がることを許容する）ことで処理性能や消費電力の改善を図る手法[*1]も研究されている。

これらの次世代のメモリデバイスは、現在利用されているメモリデバイスとは異なる性能特性を持つ。これは、単に読み書き性能がそのまま遅くなるもしくは速くなるのではなく、例えば書き込み性能のみ遅くなるなど性能特性のバランスも異なり得ることを意味する。このような異なる性能特性が実際のアプリケーションの性能にどう影響するかを明らかにすることは重要である。また、明らかになった結果から、アプリケーションの最適化などが必要であれば、その効果を確認することも必要となる。メモリデバイスの設計の観点では、アプリケーションのユーザーが求める性能を実現するために必要なデバイスの性能要件を明らかにすることが重要となる。例えば、読み込み帯域、書き込み帯域、読み込みレイテンシ、書き込みレイテンシがそれぞれどの程度であればアプリケーションが求められる性能で実行できるのかを明らかにし、その要求を満たすようにメモリデバイスを設計することが求められる。このことから、メモリデバイスの性能特性の違いがアプリケーションに与える影響を評価することは重要である。

しかしながら、次世代メモリデバイスは研究開発の段階にあるものであり、実際のハードウェアをアプリケーションの評価に用いることは困難である。このため、実ハードウェアが無い状態でも、これらの様々なメモリ特性がアプリケーションに与える影響を評価する手法が必要となる。

[1] 国立研究開発法人理化学研究所 計算科学研究センター
7-1-26 Minatojima-minami-machi, Chuo-ku, Kobe, Hyogo 650-0047, Japan
[2] 国立研究開発法人 産業技術総合研究所 デジタルアーキテクチャ研究センター
2-4-7 Aomi, Koto-ku, Tokyo 135-0064, Japan
a) takaaki.fukai@riken.jp
b) t.hirofuchi@aist.go.jp
c) takano-ryousei@aist.go.jp
d) akram.benahmed@aist.go.jp
e) kento.sato@riken.jp

*1 このような計算機システムの設計手法を我々は Error Permissive Computing [1] と呼び研究を進めている。

このような評価を行う手法として、ソフトウェアによるアプリケーションへの遅延挿入 [2][3] や CPU シミュレータを用いた方法がある。前者は、アプリケーション実行時に、実際のハードウェアのメモリ性能と再現したいメモリ性能との差分に相当する遅延を挿入することで、メモリ特性を再現する方法である。後者は、シミュレータにおいて、CPU やメモリデバイスの内部挙動をモデル化し、メモリアクセスに要する時間も計算することで様々なメモリ特性を再現する手法である。しかし、前者は精度の面で、後者は実行速度の面で問題がある。

我々は先行発表において FPGA 評価ボードを用いた次世代メモリのエミュレーション機構の試作を報告した [4]。先行発表の時点では、メモリの読み書き操作に対する遅延の挿入機能のみを簡易に実装し、挿入した遅延に応じてアプリケーションが観測するメモリの読み書き遅延やスループットが変化することを確認した。

本稿では、その後行ったメモリエミュレータの再設計および機能拡張について述べるとともに、その初期評価の結果を報告する。報告するエミュレーション機構では、スループット、レイテンシ、およびエラー率の制御が複数のメモリ空間に対して個別に制御できる。また、エミュレーションしたメモリは独立した物理メモリ空間、NUMA ノード、および NVDIMM としてソフトウェアに見せることができ、用途に合わせて切り替えることができる。本稿での評価として、まずメモリ性能の制御に関するマイクロベンチマークを行い、メモリ性能の制御が正しく動作することを確認した。また、想定される利用方法の一つとして、レイテンシのみ異なるメモリデバイスのエミュレーションについても評価した。この結果、帯域制限と遅延挿入の組み合わせることで、概ね意図した通りのメモリ性能を再現できることを確認した。また、後者の実験は未改変のベンチマークツールを用いることで、本エミュレータがアプリケーションの改変なしで利用可能であることも確認した。

本稿の構成は以下の通りである。2 章では次世代メモリエミュレータの構成や機能、および今回評価するエミュレータの拡張内容について述べる。3 章では、今回行った評価の目的、評価の内容、および評価実験の結果と考察について述べる。4 章では関連研究について述べた後、5 章でまとめを述べる。

2. 次世代メモリエミュレータ METICULOUS

提案機構 (METICULOUS) は FPGA を用いたメインメモリエミュレータである。メインメモリの様々な性能特性をハードウェアレベルで再現する。メインメモリに対する読み込み操作および書き込み操作それぞれに対して、任意の遅延の挿入、任意の帯域幅への制限、任意のエラー率のビット化けの投入が可能である。また、複数のメモリ領域

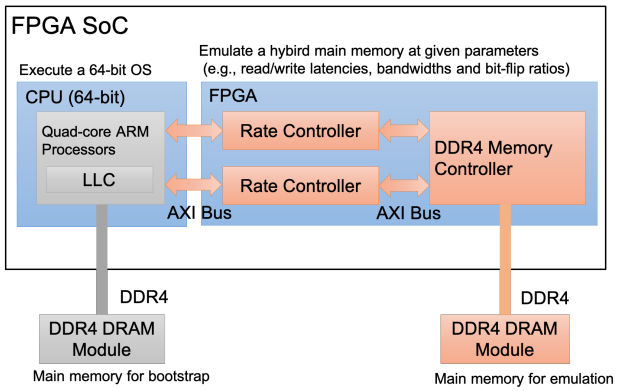

図 1　METICULOUS の概要

それぞれに対して、それぞれ任意の性能特性を再現できる。また、DRAM と Intel Optane DCPMM を搭載する計算機などにおいて見られる、複数の性能特性が異なるメモリデバイスから構成されるメインメモリ（ハイブリッド型メインメモリ）を擬似的に再現できる。

我々は先行発表 [4] において提案機構の予備的な試作を報告している。予備的な試作においては、メモリの読み書き遅延の挿入機能を実装した結果、CPU 上で動作するプログラムが観測するメインメモリの読み書き遅延および帯域が挿入した遅延の大きさに応じて変化することを確認した。その後、再設計を行い、遅延の挿入に加えて帯域幅の制限およびビット化けの投入に対応し、ハイブリッド型メインメモリのエミュレーションを可能とした。

METICULOUS の概要を図 1 に示す。CPU と FPGA がワンチップにパッケージ化された FPGA SoC (System on Chip) を用いる。FPGA SoC の CPU と FPGA 側に接続する DRAM モジュールの間の通信回路において性能調整機構（図中 Rate Controller）を設けることで、遅延の挿入等の操作を可能にする。CPU の任意の物理アドレス空間に FPGA 側の DRAM モジュールを割り当てることが可能である。CPU は、CPU に直接接続された DRAM モジュール同様に、メインメモリの一部として FPGA 側の DRAM モジュールを利用できる。FPGA 側の DRAM モジュールのメモリ領域を任意の容量で分割し、それぞれに対して性能調整機構を接続することが可能である。例えば、相対的に高速なメモリデバイスと低速なメモリデバイスからなるハイブリッド型メインメモリを再現可能である。

2.1 METICULOUS の構成

METICULOUS においては、FPGA 上に実装する主なコンポーネントとして、ハイブリッド型メインメモリのメモリ領域それぞれに対する性能調整機構、CPU 上のプログラムから METICULOUS のパラメタを設定し、また情報を取得するための CSR (Control Status Register)、DRAM モジュールの制御のための DDR4 メモリコントローラが存在する。

性能調整機構は、標準的な通信インタフェース規格である AXI4 (Advanced eXtensivle Interface 4) 向けに設計した。FPGA 上で各コンポーネントを接続する際に用いられる標準的なインタフェースに対応することで、既存の IP 等と容易に組み合わせることが可能である。性能調整機構は、FPGA SoC の CPU に対しては AXI バスのスレーブ、DDR4 メモリコントローラに対しては AXI バスのマスタとして動作する。FPGA SoC の CPU は AXI バスのマスタとして動作し、FPGA 側メモリに対するデータ読み書き要求を送信する。性能調整機構は、読み書き要求を受信すると、性能調整に係る前処理を行った上で、自身が AXI バスのマスタとなって DDR4 メモリコントローラに対して要求を送信する。性能調整機構は、DDR4 コントローラからの読み書き要求の返答を受信すると、遅延の挿入、帯域の制限、エラーの投入等を行い、CPU に対して返答を送信する。

CPU が発行するメモリ読み書き要求に対してハングアップすることなく動作させるため、以下のように AXI インタフェースの仕様に高度に準拠する設計とした。性能調整機構は任意の AXI バスの間に挿入可能であり本稿で述べるエミュレータ以外の用途に対しても適用可能である。

- 非同期な読み書き要求に対応する。例えば、CPU は読み書き要求の完了を待たずに次の読み書き要求を発行する可能性がある。性能を最大化するためには、転送先のメモリコントローラが新たな読み書き要求を受信できるなら、性能調整機構は CPU を待たせることなく次の読み書き要求を受信する必要がある。
- 読み書き要求のバースト転送に対応する。例えば、CPU は一つの読み書き要求で複数回のデータ転送を指示する可能性がある。後述するように、読み書き要求とデータ転送の対応関係を把握する必要がある。
- 読み書き要求の処理順序規則に対応する。異なるトランザクション ID を持つ読み書き要求は順番を並び替えて処理することが可能である。一方、同じトランザクション ID を持つ読み書き要求は順番を維持する必要がある。例えば、ある CPU コアから発行された一連の読み書き要求は同じトランザクション ID を有する可能性がある。この時、先に遅延が大きいメモリ領域に対して発行された読み書き要求は、あとから発行された遅延の小さいメモリ領域に対して発行された読み書き要求よりも、必ず早く完了しなければならない。

AXI バスは 5 つのメモリチャネルから構成される。性能調整機構は、読み込み要求に対して下記のように動作する。

- 性能調整機構は、CPU 側 AXI バスの AR (Read Address) チャネルを通じて読み込み先のアドレスおよび付加情報（バースト回数やトランザクション ID 等）を受信する。CSR を参照して挿入すべき遅延値を求め、トランザクション ID をキーとする順序付きマップに遅延値とバースト回数を記載する。メモリコントローラ側の AR チャネルにアドレスおよび付加情報を転送する。
- 性能調整機構は、メモリコントローラ側 AXI バスの R (Read) チャネルから読み込みデータを受信する。1) CSR を参照して投入すべきビット化けの確率を求め、疑似乱数に基づいて確率的に読み込みデータのビットを反転する。2) 順序付きマップを参照し、読み込みデータに付加されたトランザクション ID からバースト回数を求める。さらに読み込みデータを受信し、同じトランザクション ID を有するバースト回数分の読み込みデータを同じ FIFO キューに入れる。3) 挿入する遅延時間経過後、FIRO キューから読み込みデータを取り出し、CPU 側 AXI バスの R チャネルに読み込みデータを送信する。この際、CSR を参照し帯域の制限値を求め、トークンバケット方式により送信レートを制御する。

書き込み要求についても、同様に AW (Write Address) チャネルおよび W (Write) チャネルを制御することで、遅延の挿入、帯域の制限、エラーの投入を行う。なお、B (Write Status) チャネルは制御しない。

2.2 メインメモリへの統合手法

METICULOUS が提供するメモリ領域をメインメモリに統合する手法は複数存在する。CPU 側の DRAM モジュールのメモリ領域をメインメモリに統合する手法も同様に調整可能であり、任意の組み合わせが可能である。

第一の手法は、METICULOUS が提供するメモリ領域をメインメモリの物理アドレス空間に割り当てるものの、オペレーティングシステムが管理するメインメモリの対象外として設定するものである。METICULOUS に対してメモリの読み書きを行うプログラムは、`malloc()` 等のオペレーティングシステムが提供するメモリ管理 API を用いるのではなく、例えばデバイスファイルの`/dev/mem` を用いて直接物理アドレスの読み書きを行う。後述する評価実験の一部のように、オペレーティングシステムのメモリ管理の影響を排除したい用途で本手法を用いる。

第二の手法は、METICULOUS が提供するメモリ領域を NUMA ドメインとして割り当てるものである。例えば、CPU 側の DRAM モジュールのメモリ領域は最初の NUMA ドメイン、METICULOUS のメモリ領域は 2 番目の NUMA ドメインとなるようにハードウェアを設定する。また、ハイブリッド型メインメモリをエミュレーションする場合には、さらに 3 番目の NUMA ドメインを設ける。オペレーティングシステムのメモリ管理が NUMA に対応していれば、例えば `numactl` 等のコマンドを用いて特定のプロセスのメモリ確保を METICULOUS 上のメモリから行うことが可能である。既存のプログラムを改変せずとも

METICULOUS に対して適用できる。

第三の手法は、METICULOUS が提供するメモリ領域を NVDIMM モジュールが提供するメモリ領域として見せるものである。オペレーティングシステムが備える NVDIMM 向けのデバイスドライバを適用可能にすることで、例えば `ndctl` 等のコマンドを用いて不揮発性メインメモリデバイスと同様に取り扱うことができる。ただし、不揮発性メインメモリ向けのキャッシュライン制御命令が発行できるかは FPGA SoC の CPU 機能に依存する。

2.3 エミュレーションパラメタの制御手法

挿入する遅延等を設定するため、METICULOUS は制御用 API を提供する。CPU 上で実行するプログラムから利用できる。**図 2** に一部抜粋を示す。ハイブリッド型メインメモリを構成する複数のメモリ領域の境界を設定する関数も提供する。また図中には記していないものの、FPGA 側メインメモリの読み書き要求数や転送量等の情報を取得する API も提供する。制御 API を提供するライブラリ内部では、FPGA 上の CSR を読み書きすることでこれらの機能を実現する。

2.4 実装

Xilinx 社製の FPGA SoC 評価ボードである Zynq Ultrascale+ ZCU104 を対象に METICULOUS を実装した。開発フレームワークとして、ハードウェア部分に対しては Vitis 2020.2、ソフトウェア部分に対しては Petalinux 2020.2 を用いた。DDR4 メモリコントローラとしては Xilinx 社製のメモリコントローラ IP（MIG）を用いた。なお、METICULOUS は MIT ライセンスで公開する。照会先等の詳細はホームページを参照のこと[*2]。

3. 評価

本章では、METICULOUS の評価を行う。まず、3.1 章において評価の目的と方法を明らかにした後に、3.2 章においてその評価結果を示す。

3.1 評価目的と方法

本稿では、マイクロベンチマークによる評価 (3.1.1 章) と未改変アプリケーションを想定した制御実験 (3.1.2 章) を表 1 に示す評価環境で行う。

3.1.1 マイクロベンチマークによる評価

まず、METICULOUS が正しくメモリ特性を再現できているか検証するため、自作のマイクロベンチマークによる性能評価を行う。具体的には、METICULOUS の機能である (1) メモリスループット、(2) メモリレイテンシ、(3) エラー挿入の制御を行い、マイクロベンチマークのメモリアクセス性能が意図した挙動を示しているかを検証する。また、これらの測定ではエミュレートされたメモリは NUMA ノードとしては割り当てず、アプリケーションから `/dev/mem` 経由で物理アドレスを指定してマップしアクセスする。

[*2] `https://takahiro-hirofuchi.github.io/meticulous-emulator/`

表 1 評価環境

FPGA board	Zynq UltraScale+ MPSoC ZCU 104
CPU (APU)	Cortex-A53 MPCore (4 コア)
L1 Cache	32 KB (コア毎)
L2 Cache	1 MB (4 コアで共有)
Memory (DRAM)	2GB
Memory (Emulated)	4GB
OS	Ubuntu 20.04 LTS, Linux 5.4.0

3.1.2 未改変アプリケーションを想定した制御実験

近年、メモリウォール問題と言われている通り、特にアプリケーションの性能がメモリの性能に律速されるアプリケーションにおいて、メモリのデータ入出力速度が CPU や GPU 等の演算器が必要とするデータ処理速度に比べ相対的に遅くなっている。このため、これらのアプリケーションでは、メモリ性能の向上が重要となっており、様々な次世代メモリデバイスの研究開発がされている。しかし、開発段階の次世代デバイスの性能を実際のアプリケーションを用いて評価することは困難である。このため、実ハードウェアが無い状態でもこれらの様々なメモリ特性がアプリケーションに与える影響を評価し、次世代デバイスおける実アプリケーションの性能の外挿するための技術開発が必要となっている。

1 つの手法として、ルーフラインモデル [5] によりアプリケーションの性能が演算律速もしくはメモリバンド幅律速であるかを同定することにより、アプリケーションの性能向上のためには、どちらの性能を向上させれば良いかの指針を示すことが可能であるが、演算性能とメモリバンド幅のみが対象であるため、どちらの性能にも律速されない（つまり、ルーフラインモデルのライン上に存在しない）アプリケーションの性能の外挿は難しい。METICULOUS は、メモリレイテンシも制御可能であるため、演算性能とメモリバンド幅のどちらにも律速しないアプリケーションの性能の外挿に活用できると期待される。このため、METICULOUS がこのようなアプリケーション性能の外挿に実用的な機構であるかを検証するために、(1) アプリケーションコードの変更無しに METICULOUS を利用可能か否か、(2) メモリレイテンシーを制御した場合のスループットの変化を評価する。

具体的には、sysbench [6] の memory テストを実行し、スループットを測定する。また、sysbench 実行時にはスループットを 100 MB/sec に制限し、レイテンシーの挿入時間を変えて測定する。この実験において、sysbench はパッケージマネージャーでインストールしたものをそのまま利

```
/* The upper 16 bits of the second argument is read latency,
 * the lower 16 bits is write latency. */
ME_SetLatency(const unsigned int bank, const unsigned int latency_in_100ns);
ME_SetThroughput(const unsigned int bank, const unsigned int throughput_in_10mbps);
ME_SetErrorRate(const unsigned int bank, const unsigned int error_rate_in_percentage);

/* To emulate hybrid main memory, this function sets the boundary of 2 memory regions. */
ME_SetBoundary(const unsigned int bank, const unsigned int boundary);
```

図 2 METICULOUS の制御用 API（一部抜粋）

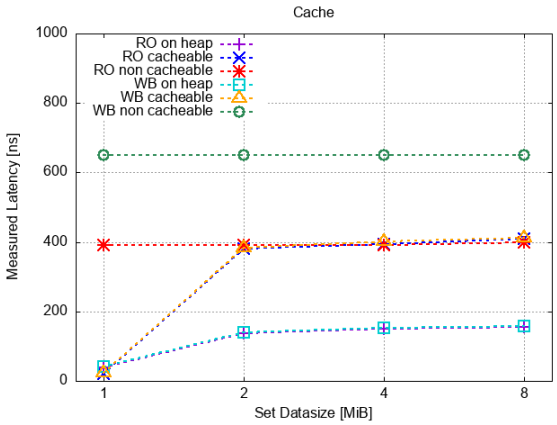

図 3 キャッシュの影響 (各設定での測定値を直線で結んでいるものの、これは視認性のためであり必ずしも直線の通り性能が推移すると予想するものではない)

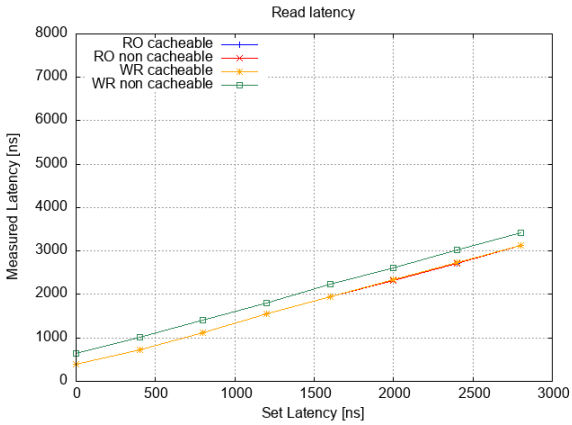

図 4 読み込みレイテンシ

用しており、アプリケーションの改変は行わない。また、sysbench プログラムがエミュレートされたメモリを利用するようにするために、`numactl` コマンドを用いて、エミュレートされたメモリが割り当てられている NUMA ノードのメモリを利用するよう指定する。

3.2 実験結果: マイクロベンチマーク

3.2.1 CPU キャッシュの有効性

METICULOUS が提供するメインメモリ領域において CPU キャッシュが有効に機能する必要がある。図 3 に CPU キャッシュの影響に関する実験結果を示す。この実験では、バッファサイズ 1 MiB, 2 MiB, 4 MiB, 8 MiB のそれぞれについて 1 スレッドでアクセスした際のレイテンシを測定している。CPU キャッシュの有無による影響を調べるために、CPU キャッシュを有効にした場合 (cachable) と無効にした場合 (non cachable) のそれぞれについてレイテンシを測定し比較している。また、比較対象として METICULOUS を介さずにアクセスするメモリ (図中の on heap) についても同様に測定した。アクセスパターンとしては、読み込みのみ行う場合 (図中 RO) と書き込みアクセスと読み込みアクセスを交互に行う場合 (図中 WR) について測定した。全ての測定において、バッファ内のデータにはランダムな順で重複なくアクセスしている。各測定において、バッファサイズ全体へのアクセスを 100 回繰り返し、その平均値を示している。

図 3 の結果から、エミュレートされたメモリ領域へのアクセスにおいても、エミュレートされていないメモリと同様に CPU キャッシュが効いていることが確認できる。なぜなら、L2 キャッシュと同じサイズのバッファ (1MiB) へのアクセスにおいて、キャッシュが有効になっているメモリアクセスについてはエミュレートされたメモリも heap と同等の性能となっているためである。一方、キャッシュを有効にしていないメモリアクセスでは、1MiB のバッファサイズにおいても、heap のアクセスよりも格段に大きなレイテンシとなっている。

また、全てのアクセスパターンにおいて、2MiB 以上のバッファサイズではキャッシュが効いておらず、メモリアクセスの性能差がレイテンシに影響していることが読み取れる。これは、全ての測定において、バッファ内でランダムなメモリアクセスを行っているため、バッファサイズがキャッシュよりも大きくなるとキャッシュヒット率が著しく低下するためである。これは、一般的な CPU キャッシュとメモリアクセスの傾向と同じであり、METICULOUS でもこの関係が正しく再現できていることが確認できたと言える。

3.2.2 レイテンシの制御

図 4, 図 5 および 図 6 に読み込み、書き込みおよび両方でレイテンシ制御を行う実験の結果を示す。これらの測定

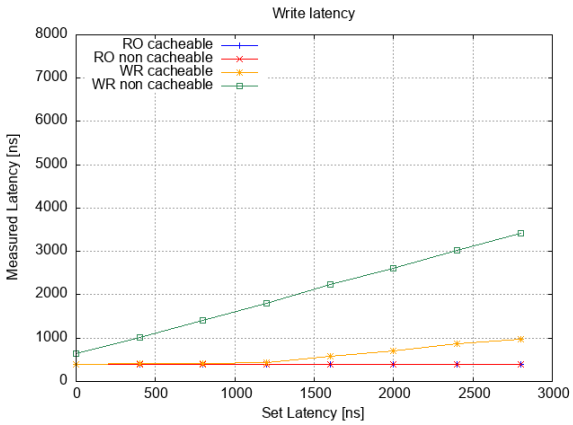

図 5 書き込みレイテンシ

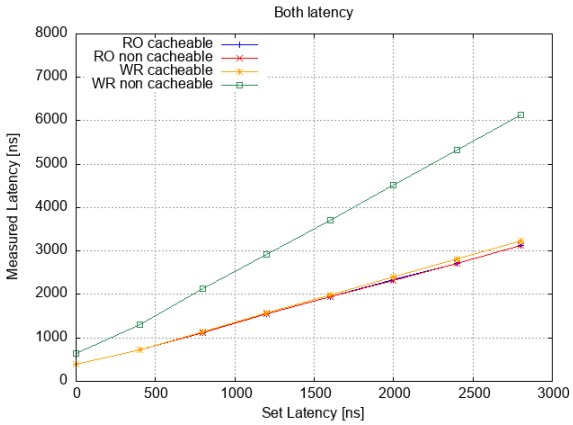

図 6 読み書きレイテンシ

では、読み込み、書き込み、もしくは両方のメモリアクセスに遅延を挿入し、メモリレイテンシを測定した。本実験は 1 スレッドで 4 MiB のバッファに対してランダムにアクセスした場合のレイテンシを測定した。実験において、読み込みのみ行う場合 (図中 RO) と書き込みと読み込みを交互に行う場合 (図中 WR) 、またそれぞれについてキャッシュを有効にした場合 (図中 cachable) と無効にした場合 (図中 non cachable) の計 4 種のワークロードについてレイテンシを測定した。それぞれのグラフにおいて、横軸が挿入する遅延の設定値 (ナノ秒単位)、縦軸がアクセスレイテンシの実測値 (ナノ秒単位) である。グラフには、メモリバッファ全体へのアクセスを 100 回繰り返し、その平均値を示している。

図 4 から、読み込みアクセスに遅延を挿入した場合、全てのアクセスパターンにおいて、挿入した遅延分レイテンシが増加していることが確認できる。図 5 から、書き込みアクセスに遅延を挿入した場合、読み込みアクセスのレイテンシには影響がないことが確認できる。また、書き込みアクセスに関しては、キャッシュが無効の場合、挿入した遅延がそのままアクセスレイテンシーに加算されていることが確認できる。一方、キャッシュが有効になっている場合、1000 ns 程度の遅延まではキャッシュの効果で隠蔽されており、それより大きな遅延を挿入した場合は、挿入した遅延よりは小さいもののレイテンシが増加していることがわかる。

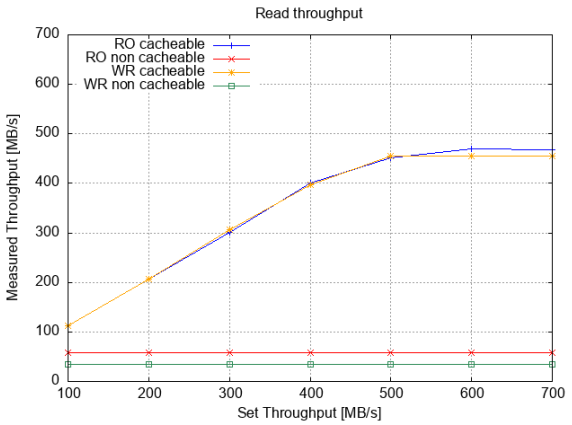

図 7 読み込みスループット

図 6 から、読み込みアクセスと書き込みアクセスの両方に遅延を挿入した場合、読み込みアクセスのレイテンシに関しては読み込みアクセスにのみ遅延を挿入した場合 (図 4) と同様の結果になっており、書き込みアクセスへの遅延挿入が読み込みアクセスへの遅延挿入に影響しないことが確認できる。一方、キャッシュが有効になっている書き込みアクセスについては、ほとんど読み込みレイテンシの遅延と同等となっている。これは、読み込みレイテンシの増加によって、書き込みアクセスへの遅延挿入の影響がほぼ全てキャッシュによって隠蔽されているためと考えられる。一方、キャッシュが無効になっている書き込みアクセスについては、読み込みアクセスと書き込みアクセスそれぞれへの遅延の挿入の合計分、レイテンシが増加している。

これらの結果から、レイテンシの増加を正しく模倣することができていると考えられる。また、キャッシュの有無によるメモリレイテンシ増加の影響の違いも正しく再現できていると考えられる。

3.2.3 スループットの制御

図 7 および 図 8 に読み込みおよび書き込みスループットの制御に関する実験結果を示す。この実験では、4 スレッドでメモリにシーケンシャルアクセスした際のスループットを測定した。メモリバッファはスレッド毎に 1 MiB バッファを用意し、各スレッドが別々のメモリバッファにアクセスするようにした。実験において、読み込みのみ行う場合 (図中 RO) と書き込みと読み込みを交互に行う場合 (図中 WR) 、またそれぞれについてキャッシュを有効にした場合 (図中 cachable) と無効にした場合 (図中 non cachable) の計 4 種のワークロードについてスループットを測定した。実験では、読み込みスループットと書き込みスループットそれぞれを片方ずつ制限しながら上記ワークロードのスループットの測定し、その変化を調べた (それぞ

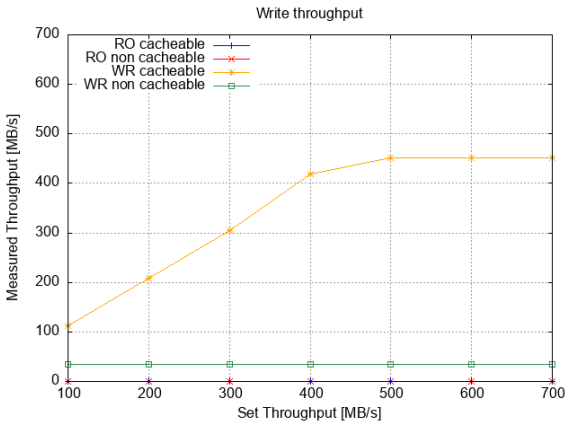

図 8 書き込みスループット

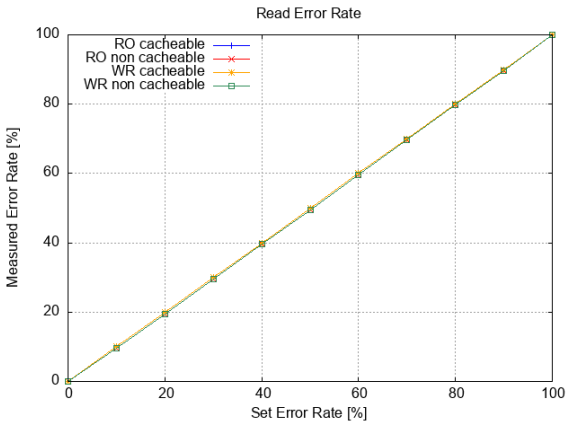

図 9 読み込みエラー

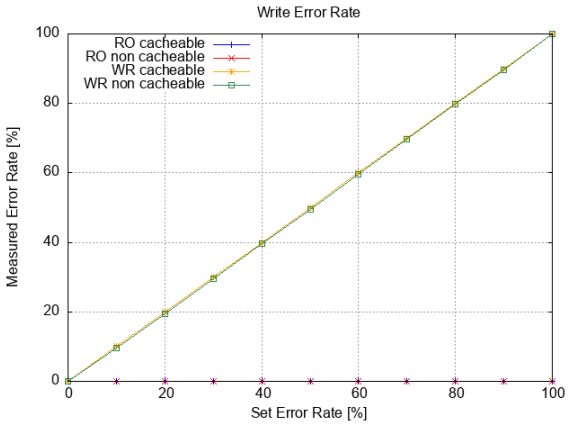

図 10 書き込みエラー

れ図 7, 図 8)。読み込みスループットの実験ではメモリ読み込みのスループット、書き込みスループットの実験ではメモリ書き込みのスループットをそれぞれ測定してグラフに示している。各グラフにおいて、横軸がスループットの制限値、縦軸がスループットの実測値である、メモリバッファ全体へのアクセスを 100 回繰り返した結果の平均値を示している。

図 7 から、読み込みスループットの制限値が 400 MiB 以下かつキャッシュが有効の場合、読み込みおよび読み書きのワークロード共に制限値通りのスループットとなることが確認できた。制限値が 500 MiB 以上のときにはスループットが一定になっているため、エミュレーションされたメモリの性能限界に達したと考えらえる。一方、キャッシュが無効の場合、スループットが 100 MiB に達さず、スループットの制限による影響を確認することができなかった。

図 8 から、書き込みスループットの制限値が 400 MiB 以下かつキャッシュが有効場合、書き込みアクセスのスループットは概ね書き込みスループットの制限値通りになっていることが確認できた。一方、キャッシュが無効の場合は読み込みスループットの実験と同様、実測値が 100 MiB に達さずスループットの制限による影響を確認できなかった。また、読み込みのみのワークロードでは書き込みアクセスが発生しないため、書き込みスループットは 0 となった。

この実験結果から、キャッシュが有効な状態ではメモリアクセスのスループットを METICULOUS によって十分制御できていると言える。

3.2.4 メモリエラーの制御

図 9 および 図 10 に読み込みおよび書き込みアクセスのエラー制御に関する実験結果を示す。この実験では、1 スレッド で 4 MiB のメモリバッファ内をシーケンシャルにアクセスした際のエラー率を測定した結果を示している。図 9 および 図 10 はそれぞれ読み込みエラーおよび書き込みエラーを制御した時のエラー率を示している。アクセスパターンとして、読み込みのみの場合 (図中 RO) とメモリバッファ全体に書き込んだ後にメモリバッファ全体にデータを読み込む場合 (図中 WR) について測定した。また、それぞれのアクセスパターンについてキャッシュを有効にした場合 (図中 cachable) とキャッシュを無効にした場合 (図中 non cachable) について測定した。エラー発生率の測定は、CSR の値を参照して行った。各グラフの横軸がエラー発生率の設定値、縦軸がエラー発生率の実測値である。グラフには、メモリバッファ全体へのアクセスを 100 回繰り返し、その平均のエラー率を示している。

2 つグラフから、キャッシュの有無や読み書きの違いに関わらず、指定したエラー発生率にしたがってエラーが発生していることが確認できる。また、書き込みエラーの実験において、RO のワークロードでは書き込みアクセスが発生しないため常にエラー率が 0 となっている。このため、METICULOUS でメモリのエラー発生率を十分再現できていると言える。

3.3 実験結果: 未改変アプリケーションでのスループットとレイテンシの制御

3.1.2 で述べた通り、METICULOUS は次世代メモリがアプリケーション性能に与える影響の評価に利用することが期待される。その中で、METICULOUS はレイテンシを

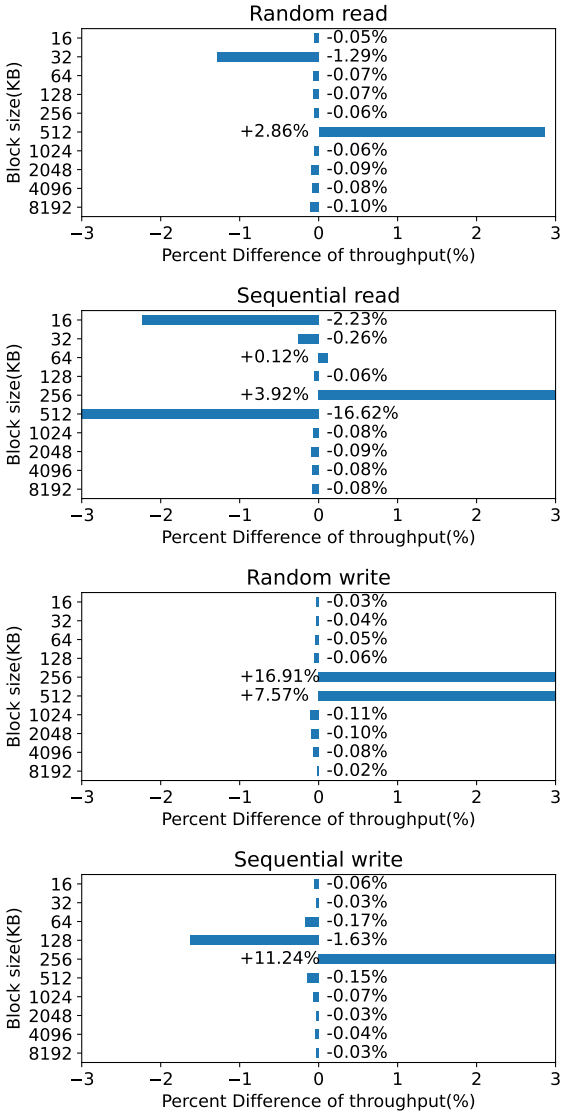

図 11 スループット 100 MiB/sec 制限時における 400ns 遅延挿入の有無によるメモリスループットの比較 (遅延を入れた場合にスループットが下がれば負値)。測定は未改変の sysbench によって実施。

制御可能であるため、メモリレイテンシの違いによるアプリケーション性能への影響をスループットの違いとは独立して評価できることが期待される。このためには、同一スループットかつ異なるレイテンシのメモリを再現する必要がある。また、さまざまなアプリケーションに対して性能への影響を評価するには、アプリケーションへのコードの改変を要求しないことが望ましい。このような観点から、上記用途での実用性を評価するため、本項では同一スループットで異なるレイテンシのメモリを再現できるか否かを未改変の既存ベンチマークソフトを用いて測定した結果を示す。

METICULOUS は遅延を挿入することでメモリレイテンシを制御可能であるものの、単に遅延を入れただけではスループットに影響が出てしまい同一スループットで異なるメモリレイテンシのメモリの再現はできない。そこで、METICULOUS で遅延を挿入すると同時にスループットも制限することで、レイテンシの増加がスループットに直接影響することを避け、同一スループットかつ異なるメモリレイテンシのメモリを再現する。

図 11 はスループットを 100 MB/sec に制限した際に 400 ns の遅延を入れた場合と入れなかった場合のスループットを比較した結果を示している。スループットの計測にはパッケージマネージャーでインストールした未改変の sysbench [6] を用いた。また、sysbench がエミュレートされたメモリを利用するために、エミュレートされたメモリを独立した NUMA ノードとし、コマンド実行時に `numactl --membind` を用いて該当の NUMA ノードのメモリを利用するよう指定している。スループットの計測は読み込みと書き込み、およびシーケンシャルアクセスとランダムアクセスそれぞれについて行っており、4 つのグラフはそれぞれの実験条件に対応している。本実験ではベンチマークソフトは 4 スレッドでそれぞれが独立したメモリバッファに対してアクセスするよう設定した。また、各スレッドに割り当てるメモリバッファのサイズは 16 KiB から 8MiB までの 2 べきの値についてそれぞれ測定した。グラフの縦軸はメモリバッファのサイズ、横軸は遅延を挿入しなかった場合に対する遅延挿入した際のスループットの変化量をパーセントで示している。それぞれの測定において 10 秒間実行を 10 回繰り返し、その平均値をグラフで示している。

評価の結果、バッファサイズ 256 KiB と 512 KiB 以外では 400 ns の遅延を挿入した場合でもスループットを 100 MB/sec に制限すれば同等スループットを実現できていることが確認できた。一方、全てのアクセスパターンにおいてバッファサイズ 256 KiB と 512 KiB において、スループットに 2.86% から 16.91% の差が見られる。これらのバッファサイズは、CPU の 1MiB あるキャッシュのサイズに近しいサイズ (256 KiB × 4 スレッド = 1 MiB) であり、キャッシュヒットとキャッシュミスが混在し偏差が大きくなり性能差が出やすくなったものと考えられる。

この実験で想定している用途は、メモリの性能の変化によるアプリケーション性能への影響を評価することであり、その際主に対象となるアプリケーションはメモリを大量に使用するアプリケーションと考えられる。このため、CPU キャッシュよりも大きな範囲へのメモリアクセスにおいて遅延挿入の有無にかかわらず同一のスループットが実現できているため、想定する用途では多くの場合十分利用できると考えられる。一方、キャッシュサイズと同等程度の範囲にアクセスを繰り返す場合には、現状ではスループットに 2.86% から 16.91% の差があるため、これを考慮する必要があると考えられる。

4. 関連研究

本研究の関連研究として、メモリの性能や動作特性の違いを模倣する研究を示す。

アプリケーションの性能特性を模倣する手法として、アプリケーションの実行に遅延を挿入し実行時間を見積もる手法がある。Volos ら [2] は、アプリケーション実行時のメモリアクセス回数に応じて、アプリケーション実行中に遅延を挿入する手法を提案している。この手法では、アプリケーション実行時にメモリアクセス回数を数え上げ、その回数と模倣対象のメモリ性能に沿って、アプリケーションに遅延を挿入する。Koshiba ら [3] は、読み出しと書き出しアクセスをそれぞれで数え上げることで、読み込み性能と書き込み性能の違いを考慮したメモリの性能を模倣する手法を提案している。また、Tanabe ら [7] は、アプリケーション実行中にメモリアクセス回数を数え上げ、その回数と実際のアプリケーション実行時間から、模倣対象のメモリ上での実行時間を算出する手法を提案している。これらの手法の優れた点は、実計算機を用いてアプリケーションの実行時間は測定するため、ほぼ実時間の実行のみで異なる性能特性のメモリ上での実行時間を見積もれる点である。一方で、メモリアクセス命令間の依存関係が考慮されていないため、アプリケーションのメモリアクセスパターンが異なる際のメモリレイテンシの影響を観測することは難しい。

また、TaskSim[8] や gem5[9] などのシミュレータを用いて、現在のメモリとは異なるメモリ特性を模倣し、そのアプリケーションへの影響を評価する研究もある。Asifuzzaman ら [10] は TaskSim を用いて、STT-MRAM を用いた際の HPC アプリケーションの性能への影響を評価した。また、Venkatagiri ら [11] は、現在のメモリよりも速度や消費電力の面で有利なもののエラー率が高くなるメモリデバイスを用いた際の動作を gem5 を用いて模倣し、アプリケーションの動作にどのように影響するかを評価している。

Omori ら [12] も本研究と同様に FPGA を用いてさまざまな性能特性のメモリを再現し、アプリケーションへの性能の影響を評価している。しかし、我々の手法とはメモリ性能の制御の仕方が異なる。Omori らの手法では、FPGA 内でメモリコントローラ IP のさまざまなタイミングに関するパラメータを調整することで、性能特性を変更している。このため、メモリの細部の性能特性について細かく指定することが可能である。一方で、メモリのタイミングのパラメーターを制御する方法では、スループットとレイテンシを独立して指定することは難しい。我々の手法はターゲットとなるスループットやレイテンシに沿うようメモリアクセス要求のキューの制御や遅延の挿入を行うため、両者をある程度独立して制御することが可能である。これにより、メモリレイテンシの影響のみを評価するといった利用方法が実現できる。

5. まとめ

本稿では、FGPA による次世代メモリのエミュレーション機構 METICULOUS についてその設計および実装について説明し、その評価結果を示した。METICULOUS はスループット、レイテンシおよびエラー率を指定して制御することができる。また、これらの制御はメモリ領域を複数に分割し、それぞれの領域に対して個別に制御することができる。これによって様々な性能特性を持つメモリのハイブリッド構成を再現できる。これは、FPGA で実装された性能調整機構が CPU とメモリコントローラーの間でアクセスを制御することで実現している。METICULOUS で制御するメモリは、物理メモリ領域の一部として、さらには NUMA ノードとして、あるいは NVDIMM 領域としてオペレーティングシステムに見せることができ、用途に応じて適切なインターフェイスを選択できる。

評価では、METICULOUS の制御機構が正しく動作するか否かを評価するためのマイクロベンチマークによる測定と、実際の利用場面を想定して同一スループットで異なるレイテンシのメモリを再現する実験を行った。前者の結果から、METICULOUS の制御機能は正しく動作していることが確認でき、後者の結果から実用できる程度に目的のメモリ性能を再現できたと言える。

エミュレータに関する今後の課題は、より幅広い計算機環境で利用できるよう PCI Express 接続できる FPGA による設計と実装が挙げられる。現在 Xilinx Alveo を用いた試作を行っており今後報告する。また、エミュレータの利用については、エミュレータを用いて様々なベンチマークソフトウェアやアプリケーションの性能分析を行うことが挙げられる。

謝辞 本研究の一部は科研費 19H01108 の助成を受けた。

参考文献

[1] National Institute of Advanced Industrial Science and Technology (AIST): Error Permissive Computing, `https://error-permissive-computing.github.io/`.
[2] 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, Association for Computing Machinery, p. 37–49 (online), DOI: 10.1145/2814576.2814806 (2015).
[3] KOSHIBA, A., HIROFUCHI, T., TAKANO, R. and NAMIKI, M.: A Software-based NVM Emulator Supporting Read/Write Asymmetric Latencies, *IEICE Transactions on Information and Systems*, Vol. E102.D, No. 12, pp. 2377–2388 (online), DOI: 10.1587/transinf.2019PAP0018 (2019).
[4] FPGA による次世代メモリのエミュレーション機構の

試作：研究報告ハイパフォーマンスコンピューティング（HPC），Vol. 2019-HPC-171, No. 2, 情報処理学会，pp. 1–7 (2019).

[5] Williams, S., Waterman, A. and Patterson, D.: Roofline: An Insightful Visual Performance Model for Multicore Architectures, *Commun. ACM*, Vol. 52, No. 4, p. 65–76 (online), DOI: 10.1145/1498765.1498785 (2009).

[6] Alexey Kopytov: sysbench, `https://github.com/akopytov/sysbench`.

[7] Tanabe, N. and Endo, T.: Characterizing Memory-Latency Sensitivity of Sparse Matrix Kernels, *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, Los Alamitos, CA, USA, IEEE Computer Society, pp. 249–254 (online), DOI: 10.1109/PDP2018.2018.00042 (2018).

[8] Rico, A., Cabarcas, F., Villavieja, C., Pavlovic, M., Vega, A., Etsion, Y., Ramirez, A. and Valero, M.: On the Simulation of Large-Scale Architectures Using Multiple Application Abstraction Levels, *ACM Trans. Archit. Code Optim.*, Vol. 8, No. 4 (online), DOI: 10.1145/2086696.2086715 (2012).

[9] 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 Comput. Archit. News*, Vol. 39, No. 2, p. 1–7 (online), DOI: 10.1145/2024716.2024718 (2011).

[10] Asifuzzaman, K., Pavlovic, M., Radulovic, M., Zaragoza, D., Kwon, O., Ryoo, K.-C. and Radojković, P.: Performance Impact of a Slower Main Memory: A Case Study of STT-MRAM in HPC, *Proceedings of the Second International Symposium on Memory Systems*, MEMSYS '16, New York, NY, USA, Association for Computing Machinery, p. 40–49 (online), DOI: 10.1145/2989081.2989082 (2016).

[11] Venkatagiri, R., Ahmed, K., Mahmoud, A., Misailovic, S., Marinov, D., Fletcher, C. W. and Adve, S. V.: gem5-Approxilyzer: An Open-Source Tool for Application-Level Soft Error Analysis, *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, DSN, pp. 214–221 (online), DOI: 10.1109/DSN.2019.00033 (2019).

[12] Omori, Y. and Kimura, K.: Performance Evaluation on NVMM Emulator Employing Fine-Grain Delay Injection, *2019 IEEE Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, NVMSA, pp. 1–6 (online), DOI: 10.1109/NVMSA.2019.8863522 (2019).