# Read/Writeレイテンシの違いを考慮した不揮発性メモリのソフトウェアエミュレータ

小柴 篤史[1,a)]　広渕 崇宏[2]　高野 了成[2]　並木 美太郎[1]

**概要**：STT-MRAM などの不揮発性メモリ (NVM) は，従来の DRAM に替わる次世代の高性能メモリとして注目されている．NVM の特性を活用したメモリ管理手法やアーキテクチャの研究が進められている一方，実際に製品化されている NVM は少ないため，評価にはサイクルレベルのシミュレータを用いることが一般的である．しかし従来のシミュレータは長い実行時間を要するため，大規模なアプリケーションやオペレーティングシステムを含む挙動をシミュレーションによって評価することを困難にしている．そこで筆者らは，より軽量なソフトウェアによる NVM のエミュレーション手法の研究を進めている．本論文では，既存の研究のエミュレーション方式を拡張し，Read/Write レイテンシが異なる特性を持つ NVM をエミュレーションする方式を提案する．提案手法を VM に適用した時のベンチマーク実行時間を実機上で評価し，指定した Write レイテンシに応じてベンチマークの実行時間が増加することを確かめた．

## 1. はじめに

近年，計算機システムの処理速度向上およびメニーコア化に伴い，より大容量のメモリを搭載したシステムの需要が高まっている．DRAM は代表的な計算機の主記憶装置として広く用いられてきたが，更なるスケールアップにおいて電力制約が問題となる．より大容量の DRAM をシステムに搭載すると，データを保持するためのリフレッシュ処理による電力消費量が無視できなくなる．加えて，DRAM そのものの微細化によるスケールアップも限界が近い．そこでメモリの電力制約を解決する新たな記憶装置として，STT-MRAM などの不揮発性メモリ (Non-Volatile Memory, 以降 NVM) が注目されている．NVM は将来的には DRAM と同等の読み書き性能を持つと予測されており [1]，DRAM のように CPU の主記憶装置として利用できる可能性がある．加えて，データを保持するためにリフレッシュ処理を行う必要がないため，DRAM よりもエネルギー効率が高いと考えられている．このように NVM は従来の DRAM に替わる次世代の主記憶装置として期待されている．

NVM は稼働中の電力効率が高い一方で，Read/Write のレイテンシが異なる，書き込み時の消費エネルギーが読み込み時よりも大きいなど，DRAM とは異なる特性を持つ．そこで，これらの特性を考慮して NVM を活用するためのメモリの管理手法や新たなメモリシステムの研究が行われている [2], [3], [4]．NVM を用いた計算機システムの研究において，現状では製品化されている NVM の実機は少ないことから，サイクルレベルでのメモリシミュレータを用いて提案手法の評価を行うことが一般的である [5]．しかし，このようなサイクルレベルでのメモリのシミュレーションは非常に長い実行時間を要し，実機では 1 秒で終わる処理が 8 時間以上かかることもある．そのため，大規模なワークロードやベンチマークを使った実験には膨大な時間を要してしまう．実環境に即した評価を容易にする新たな手法が求められる．

そこで筆者らは，DRAM 環境上で NVM の性能を疑似的に再現 (エミュレーション) する，軽量なソフトウェア技術の研究を進めている．本研究では Volos らが提案している NVM エミュレータ [6] の方式を拡張し，NVM の Read/Write レイテンシの違いを考慮したエミュレーション手法を提案する．Volos らの手法は，プロセスのメモリアクセス頻度を観測しながら，想定する NVM のアクセスレイテンシに基づいてプロセスの実行速度を調整することで，NVM 環境におけるプロセスの挙動を DRAM 環境で再現できる．この手法はほぼ実時間でのエミュレーションを可能にする一方で，NVM の Read と Write のレイテンシは同等であるという前提を置いているため，適用範囲が狭い．多くの NVM では，Write レイテンシが Read レイテンシよりも大幅に遅いという特性を持つ．そのため，キャッシュミス時に発生したライトバック処理が要する時

1　国立大学法人 東京農工大学
2　国立研究開発法人 産業技術総合研究所
a)　koshiba@namikilab.tuat.ac.jp

間が大幅に増加し，実際には大きな性能低下が生じる可能性がある．そこで本論文では，エミュレーション対象のプロセスの動作に起因するキャッシュミスをライトバックを伴うものと伴わないものの 2 種類に判別し，ライトバックを伴うキャッシュミスがより長く CPU をストールさせる場合のプロセスの実行時間の推定方法を提案する．提案手法を Volos らの手法に適用することで，Read/Write レイテンシが異なる特性を持つ NVM に対しても高い精度でのエミュレーションを可能にする．更に本論文では，提案手法の基礎評価として，DRAM を搭載した Intel プロセッサ上で提案するエミュレータのプロトタイプを稼働し，ベンチマークの実行時間を計測する．これにより，異なる Read/Write レイテンシを持つ NVM の性能を DRAM の実機上で再現できることを示す．

## 2. 関連研究と課題

本研究で想定する NVM は，DRAM と同様にバイトアドレッサブルであり，CPU の load/store 命令でアクセスできるものとする．NVM を主記憶装置として用いるシステムでは，DRAM よりも大きい Read/Write レイテンシが課題となると予想される．近年の CPU の多くはライトバック方式のキャッシュ機構を持つため，メモリモジュールへの読み書きは主に load/store 命令がラストレベルキャッシュ (LLC) をミスする時に行われる．CPU がメモリに対して load/store 命令を実行すると，まずキャッシュを参照して指定したメモリ領域のデータがあるか調べ，LLC に該当するデータがない場合 (LLC ミス)，メモリへアクセスする．メモリモジュールに対する読み込みは load/store 命令が LLC ミスを引き起こした時にすぐ実行され，キャッシュに該当するデータが読み込まれるまで CPU はストールする．このとき，LLC に新しいデータを読み込むために古いデータが追い出されることがある．この追い出しの対象になったデータが store 命令等によって変更されていた場合，変更を反映するためにメモリモジュールに対する書き込みが行われる (ライトバック)．NVM を用いたシステムでは，このような LLC ミスによるメモリアクセス時に DRAM よりも多くのペナルティがかかると予想されるため，この読み書き性能の違いを考慮した NVM 向けの資源管理手法を検討する必要がある．

NVM の実製品の入手が困難なことから，NVM を用いたメモリシステムに関する研究では，DRAM マシン上でメモリのシミュレータを用いて評価が行われる．一般的には，サイクルアキュレートなメモリのシミュレータと CPU のシミュレータを組み合わせて，サイクル単位で NVM のアクセスレイテンシを再現する方法がとられている [7], [5]．しかし，このような詳細なシミュレーションはオーバヘッドが大きく，長い実行時間を要するため，大規模なベンチマークの実行には適さない．また，Intel Labs は専用の

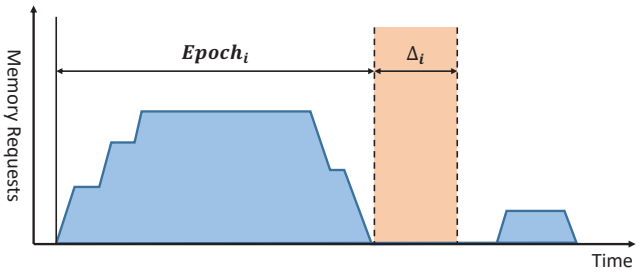


図 1 Quartz の NVM レイテンシのエミュレーション手法 [6]

CPU マイクロコードを用いて NVM のアクセスレイテンシを再現できるハードウェアを開発した [3] が，この手法はハードウェアベンダの協力が必要で，一般的には使えない．そこで，ソフトウェアによる軽量な NVM エミュレータとして，Volos らは Quartz[6] を提案している．Quartz は，主記憶装置が DRAM のマシン上で動作する NVM エミュレータで，想定する NVM の性能に応じて，ワークロードの実行速度を調整する機構である．Quartz はユーザ空間で稼働するアプリケーションとハードウェアカウンタを制御するカーネルモジュールで構成される．Quartz のエミュレーション手法を図 1 に示す．Quartz は一定の時間間隔 $Epoch$ ごとにハードウェアカウンタを用いてエミュレーション対象のプロセスの DRAM アクセス回数を計測する．そして，DRAM ではなく NVM 上で実行されたときに生じていたであろうメモリアクセスによる CPU 遅延時間 $\Delta$ を見積もり，$\Delta$ の分だけ CPU 上で稼働しているプロセスの実行を遅らせる．$Epoch_i$ における遅延時間 $\Delta_i$ は式 (1) で算出される．

$$\Delta_i = LDM_i \times (NVM_{lat} - DRAM_{lat}) \quad (1)$$

ここで，$LDM_i$ はプロセスの動作に起因するメモリモジュールからの読み込み，$NVM_{lat}$，$DRAM_{lat}$ はそれぞれ NVM のアクセスレイテンシ，DRAM のアクセスレイテンシを表す．

Quartz は $\Delta_i$ を Epoch ごとに算出し，POSIX Signal を用いてプロセス実行を中断/再開することで NVM 上でのプロセスの挙動をエミュレーションする．この手法はソフトウェアによる計算オーバヘッドが少ないため軽量だが，NVM に対する書き込みによって生じるレイテンシを考慮していないことが課題となる．Quartz の論文では，メモリに対する書き込みはキャッシュの追い出し時にしか起こらないため，CPU のボトルネックにはならないとしている．しかし，NVM は一般的に Write レイテンシが Read レイテンシよりも大きい特性があることが指摘されており，書き込みが読み込みの数倍遅い場合もある [7], [8]．このような環境でキャッシュのライトバック処理が頻発すると，ライトバックが終わるまでキャッシュに新しいデータを読み込めなくなり，結果として CPU のストール時間が増加すると考えられる．そこで本研究では，Read/Write レイテ

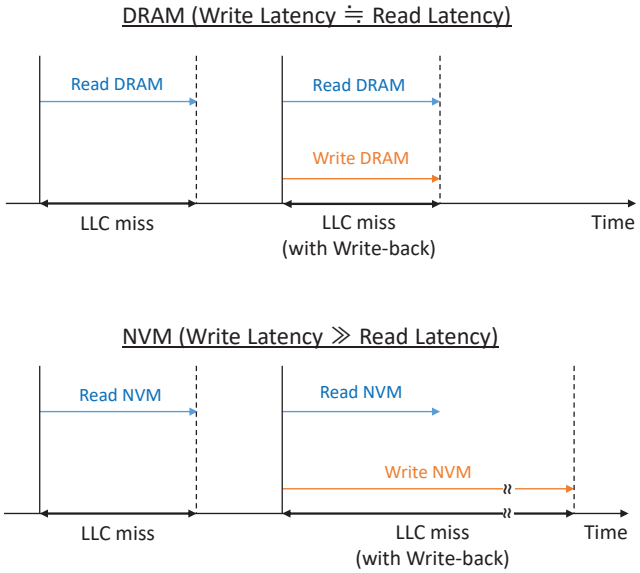


図 2　DRAM および本研究で想定する NVM におけるメモリアクセスレイテンシ

ンシが異なる特性を持つ NVM について，メモリ Write 時の CPU オーバヘッドを考慮したエミュレーション手法を提案する．

## 3. Write レイテンシを考慮した NVM エミュレーション手法

本章では前述の課題を解決するため，NVM の特性においてメモリへの書き込みがメモリからの読み込みよりも遅い環境でのアプリケーションの挙動をエミュレーションする手法を述べる．

### 3.1 キャッシュのライトバックによる CPU オーバヘッド

まず，主記憶装置が NVM のとき，LLC のライトバック (WB) が CPU の性能に与える影響について述べる．CPU による LLC の追い出しが発生しうるケースは，(1)load/store 命令によって LLC ミスが発生した時，(2)clflush などのキャッシュフラッシュを行う命令を実行した時である．これらのケースで変更のあった LLC のラインが追い出される時にのみ，メモリに対する WB が行われる．(2) は明示的にプロセスが clflush を呼ばなければ起きないため，頻度は比較的少ないと思われる (今後検証する予定である)．そのため，本論文では (1) のケースについて考える．

DRAM および本研究で想定する NVM を用いたシステムにおいて，それぞれの環境における LLC ミス時のメモリアクセスレイテンシを図 2 に示す．LLC ミス時に WB が行われない場合はメモリ Read のみ行われるため，メモリアクセスレイテンシは DRAM と NVM 共に Read レイテンシ分だけとなる．一方，LLC ミス時に WB が行われる場合，LLC ミスによるメモリ Read とライトバック処理によるメモリ Write が並列に実行される．このとき，DRAM 環境ではメモリの読み書きに要する時間がほぼ同等とみなせるため，WB はメモリ Read に完全に隠蔽される．すなわち，DRAM 環境における LLC ミスレイテンシは WB の有無に関わらず一定 (DRAM の Read レイテンシと同じ) である．これに対して，NVM 環境ではメモリ Write がメモリ Read よりも大幅に遅いと予想されるため，WB を伴う LLC ミスが起きると，メモリ Write が完了するまでより長い時間がかかる．すなわち NVM 環境では，WB を伴う LLC ミス (メモリ Read) は，WB を伴わない LLC ミスよりも大きなペナルティがかかると予想される．そこで本研究では，プロセスによって要求されたメモリ Read を，WB を伴うものと伴わないものの 2 種類に分け，それぞれのペナルティが異なる場合の遅延時間の算出方法を提案する．これにより，Read/Write レイテンシが異なる NVM について，より正確なエミュレーションを可能にする．

### 3.2 提案手法のエミュレーション方式

本研究では，Quartz の方式を拡張して，Write レイテンシが Read レイテンシよりも遅い NVM のエミュレーション方式を提案する．前述のように，LLC ミスによって発生したメモリ Read が WB を伴う場合，WB を伴わないメモリ Read よりも CPU の待ち時間が大きくなると予想される．そこで本手法では，CPU コアは WB を伴うメモリ Read の実行時に NVM の Write レイテンシだけストール (メモリ Write が完了するまで待つ) し，WB を伴わないメモリ Read の実行時は Read レイテンシだけストールすると想定する．ここで，NVM の Write レイテンシと Read レイテンシをそれぞれ $NVM_{lat}^{Write}$，$NVM_{lat}^{Read}$ とするとき ($NVM_{lat}^{Write} \gg NVM_{lat}^{Read}$)，$Epoch_i$ における遅延時間 $\Delta_i'$ を式 (2) で表す．

$$\begin{aligned}\Delta_i' = {} & LDM_i^{WB} \times (NVM_{lat}^{Write} - DRAM_{lat}) \\ & + LDM_i^{RO} \times (NVM_{lat}^{Read} - DRAM_{lat})\end{aligned} \tag{2}$$

ここで，$LDM_i^{WB}$ は対象プロセスが稼働する CPU コアにおける WB を伴うメモリ Read の回数，$LDM_i^{RO}$ は WB を伴わない (Read Only な) メモリ Read の回数を表す．$LDM_i^{WB}$，$LDM_i^{RO}$ を算出する際，近年のプロセッサのメモリアクセスの高速化機能を考慮する必要がある．近年のプロセッサにはハードウェアプリフェッチャ (PF) やメモリアクセスの並列実行機能が備わっていることが多く，メモリアクセス処理が CPU の実行をさまたげない場合がある．そこで本手法では，式 (3) を用いて実際に CPU の実行のさまたげとなったメモリアクセス回数を求める．

$$\begin{aligned}LDM_i^{WB} &= \frac{LDM\_STALL_i^{WB}}{DRAM_{lat}}, \\ LDM_i^{RO} &= \frac{LDM\_STALL_i^{RO}}{DRAM_{lat}}\end{aligned} \tag{3}$$

ここで，$LDM\_STALL_i^{WB}$，$LDM\_STALL_i^{RO}$ はそれぞ

表 1　Sandy Bridge-E のパフォーマンスイベント

| Non-Architectural Performance Events [9] | |
|---|---|
| $L2_{stalls}$ | CYCLE_ACTIVITY:STALLS_L2_PENDING |
| $LLC_{hit}$ | MEM_LOAD_UOPS_RETIRED:LLC_HIT |
| $LLC_{miss}$<br>$LLC_{miss,cpu_i}$ | MEM_LOAD_UOPS_MISC_RETIRED:LLC_MISS |
| $LLC_{miss,PF_i}$ | OFFCORE_RESPONSE.ALL_PF_CODE_RD.LLC_MISS.DRAM_N &<br>OFFCORE_RESPONSE.ALL_PF_DATA_RD.LLC_MISS.DRAM_N |
| **Uncore Performance Events for CBo [10]** | |
| $WB$ | LLC_VICTIMS |

れ，WB を伴う/伴わないメモリ Read によって CPU コアがストールしたサイクル数を表す．近年のプロセッサの多くは，搭載されているハードウェアカウンタを用いてメモリアクセスによるストール時間を計測できる．このストール時間を DRAM の Read レイテンシ ($DRAM_{lat}$) で割ることで，実際に CPU ストールを引き起こしたメモリアクセス回数を見積もることができる．

## 4. NVM エミュレータの実装

本章では，3 章で述べた NVM のエミュレーション手法を現行のプロセッサに適用できることを示すため，Intel SandyBridge-E Architecture を対象に提案するエミュレータを実装する方法を述べる．提案するエミュレーションの主な手順は，Epoch ごとにエミュレーション対象のプロセスの遅延時間 $\Delta_i'$ を算出し，プロセスの実行時間を $\Delta_i'$ だけ遅延させる．プロセスの実行時間を遅延させる方法は既に Quartz の論文で提示されているため，本論文では式 (2) で示した $\Delta_i'$ をエミュレータが算出する方法を述べる．

本手法では，エミュレーション対象となるプロセスは特定の CPU コアに固定され，常に同じ CPU コアで実行されるものとする．エミュレータはプロセスが稼働している CPU コアについて，Intel CPU のパフォーマンスカウンタ (Performance Monitoring Counters, 以降 PMC) を用いて式 (3) に示した $LDM\_STALL_i^{WB}$，$LDM\_STALL_i^{RO}$ を計測する．これらをもとに $LDM_i^{WB}$，$LDM_i^{RO}$ を求め，式 (2) に代入し $\Delta_i'$ を求める．ここで，$LDM\_STALL_i^{WB}$，$LDM\_STALL_i^{RO}$ は，Intel の資料 [11] を参考に算出する．まず，ライトバックの有無を区別しない場合，メモリ Read による CPU コアのストールサイクル数 $LDM\_STALL_i$ の算出方法は資料 [11] で提供されており，式 (4) で表される．

$$LDM\_STALL_i = L2_{stalls} \times \frac{W \times LLC_{miss}}{LLC_{hit} + W \times LLC_{miss}} \quad (4)$$

ここで，$L2_{stalls}$ は L2 キャッシュミスによる CPU コアのストール時間，$LLC_{hit}$，$LLC_{miss}$ はそれぞれ CPU コアによる LLC ヒット/ミス回数，$W$ は重み係数 (LLC ヒット時の CPU コアのストール時間に対する LLC ミス時のストール時間の割合) を表す．更にここで，式 (4) の $LLC_{miss}$ に注目すると，$LLC_{miss}$ は WB を伴う LLC ミスと，WB を伴わない LLC ミスの 2 種類に分けられる．すなわち，CPU コアによる WB を伴う LLC ミス回数を $LLC_{miss}^{WB}$ とすると，$LDM\_STALL_i^{WB}$，$LDM\_STALL_i^{RO}$ は，それぞれ式 (5)，式 (6) で表される．

$$LDM\_STALL_i^{WB} = L2_{stalls} \times \frac{W \times LLC_{miss}^{WB}}{LLC_{hit} + W \times LLC_{miss}} \quad (5)$$

$$LDM\_STALL_i^{RO} = L2_{stalls} \times \frac{W \times (LLC_{miss} - LLC_{miss}^{WB})}{LLC_{hit} + W \times LLC_{miss}} \quad (6)$$

ここで，式 (5)，式 (6) の各変数のうち，$LLC_{miss}^{WB}$ 以外は CPU コアごとに PMC で実測できるが，$LLC_{miss}^{WB}$ は PMC では実測できない．一方で，システム全体で生じたライトバック回数は，LLC のコヒーレンシを制御する Caching Agent (CBo) に搭載されているパフォーマンスカウンタで計測できる [10]．そこで，本研究では式 (7) を用いて，$LLC_{miss}^{WB}$ の値を推定する．

$$LLC_{miss}^{WB} = LLC_{miss} \times \frac{WB}{\sum_{i=0}^{n-1} LLC_{miss,cpu_i} + \sum_{i=0}^{n-1} LLC_{miss,PF_i}} \quad (7)$$

ここで，$LLC_{miss}$ はプロセスが稼働する CPU コアによる LLC ミス回数，$WB$ はシステム全体で発生したライトバックの総数を表す．$n$ はプロセッサが持つ CPU コア数，$\sum_{i=0}^{n-1} LLC_{miss,cpu_i}$ はプロセッサが持つ全ての CPU コアによる LLC ミス回数の和，$\sum_{i=0}^{n-1} LLC_{miss,PF_i}$ は全てのハードウェアプリフェッチャによる LLC ミス回数の和を表す．この $\sum_{i=0}^{n-1} LLC_{miss,cpu_i}$ と $\sum_{i=0}^{n-1} LLC_{miss,PF_i}$ の和はシステム全体で発生した LLC ミスの総数となる．つまり，システム全体で発生した LLC ミス回数のうち，WB を伴うものの割合をもとめ，これとプロセス自身による LLC ミス回数 ($LLC_{miss}$) との積をとることで，あるプロセスにおける WB を伴う LLC ミス回数のおおよその値を推定する．

表 1 に，前述の式で示した変数と，Intel SandyBridge-E

プロセッサの PMC のイベントとの対応を示す [9], [10]．なお，$DRAM_{lat}$，$W$ は，エミュレータを適用するマシンのプロセッサ性能に依存する定数であり，Intel が提供しているツール [12] などを用いて算出できる．SandyBridge-E 以外の Intel プロセッサへの実装と評価は今後の課題とするが，Intel SandyBridge-E 以降のアーキテクチャのパフォーマンスカウンタには同様のイベントが備わっているため，他のプロセッサへの移植は可能だと考えられる．

## 5. 評価と考察

本研究では提案手法の効果を示すため，Intel SandyBridge-E Architecture が採用されている Intel Xeon E-2650 プロセッサを対象に，提案するエミュレータのプロトタイプを実装し，基礎的な評価を行った．本章では評価結果と考察について述べるが，この原稿執筆時点では全ての評価を終えていないため，一部の評価については評価方法のみを示す．評価実験を行った環境を表 2 に示す．$DRAM_{lat}$，$W$ は，Intel Memory Latency Checker [12] を用いて計測した．また，本評価では *Epoch* は 100ms としたが，*Epoch* の長さによってエミュレーションの精度が変わる可能性がある．*Epoch* とエミュレータの精度との関係の検証は今後の課題とする．

表 2　評価環境

| 項目 | 詳細 |
|---|---|
| 評価用プロセッサ | Intel Xeon E5-2650 |
| ホスト OS | Debian 8.5 (Linux 3.18.5) |
| VM | 1 コア, メモリ 8GB |
| ゲスト OS | Debian 6.0.8 (Linux 3.18.5) |
| *Epoch* | 100 ms |
| $DRAM_{lat}$ | 70.8 ns |
| $W$ | 3.5 |

### 5.1 エミュレーションの精度

提案手法によって NVM 環境を正確にエミュレーションできることを示すため，本研究で想定する Read/Write レイテンシの異なる NVM をエミュレーションしたときの精度を検証する．なお，本評価はまだ終えていないため，以下は現在取り組んでいる評価方法および本評価で明らかになると期待できる項目を示す．

本評価では，提案手法がある程度正確に NVM の挙動を再現できることを，サイクルアキュレートなシミュレータを用いて確認する．サイクルアキュレート・シミュレータは時間がかかるが正確な挙動を再現できるため，NVM の実製品を用いる環境とほぼ同じ挙動を示すと考えられる．本評価では，サイクルアキュレート・シミュレータとして Gem5 [13] と NVMain [5] を組み合わせた環境を用いる．Gem5+NVMain によるシミュレーション環境と，提案するエミュレーション機構を実機上で用いる環境の 2 種類の手法について，Read/Write レイテンシの異なる NVM 環境を再現し，それぞれの環境でメモリバウンドなワークロードを実行する．もし，提案手法で再現されたワークロードの実行時間が Gem5+NVMain の環境で再現された実行時間と差がない場合，提案手法がサイクルアキュレートなシミュレータよりも小さい手間で NVM 環境を再現できることが示せる．これにより，提案手法の有用性を示せると考えている．

### 5.2 アプリケーションへの適用

提案手法が実際に計算機上で動作するアプリケーションに対して適用できることを示すため，SPEC CPU2006 ベンチマークに対して提案手法を適用したときの挙動を検証した．本評価では，評価マシン上で QEMU を用いて VM を稼働し，VM 上で動作するベンチマークを対象に提案手法を適用した．ベンチマーク実行中，VM の VCPU スレッドが稼働する物理 CPU コアのメモリアクセス頻度を監視し，設定した NVM の Read/Write レイテンシに応じてエミュレータが VCPU スレッドを遅延させたときのベンチマークの実行時間を計測した．評価用ベンチマークとして，メモリアクセス特性の異なる 3 種類のベンチマーク (CPU バウンドな 456.hmmer, メモリバウンドな 429.mcf および 462.libquantum) を用いた．エミュレーションする NVM の Read レイテンシは DRAM のレイテンシと同じとし，NVM の Write レイテンシは DRAM の 1 倍，2 倍，3 倍，4 倍，5 倍に設定した．本評価では，各ベンチマークのメモリアクセス頻度と実行時間との関係を示すため，ベンチマークの実行時間に加えて，そのときのメモリ Write スループットを計測した．メモリ Write スループットは，Xeon E5 プロセッサのメモリコントローラに内蔵されたパフォーマンスカウンタを用いてベンチマーク実行中に DRAM へ書き込まれたデータのバイト数を計測し，これをエミュレーションされたベンチマークの実行時間で割ることで求めた [10]．

図 3 にエミュレーション時の各ベンチマークの実行時間，図 4 にベンチマーク実行中の平均メモリ Write スループットを示す．これらの結果から，メモリへの書き込みをほとんど行わない hmmer は，NVM の Write レイテンシを上げても実行時間はほぼ変わらない事が分かる．一方で，メモリバウンドな mcf, libquantum は Write レイテンシを上げるにつれて実行時間が増えていき，対称的にメモリ Write スループットが低下していることが分かる．特にメモリアクセスの頻度が多い libquantum は，Write レイテンシの増加に対する実行時間の増加と Write スループットの低下が顕著であることが分かる．このように，提案手法は設定した Write レイテンシに応じてベンチマークの実行時間を増やすことができ，その傾向はベンチマークのメモリアク

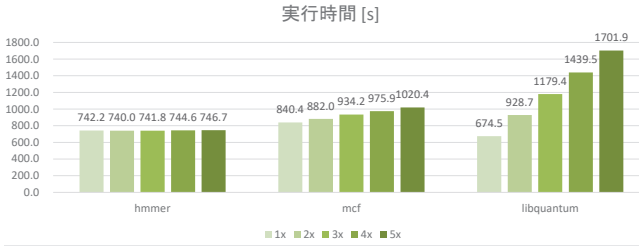


**図 3** エミュレーション時の各ベンチマークの実行時間 (DRAM に対する NVM の Write レイテンシを 1 倍，2 倍，...，5 倍に設定した場合)

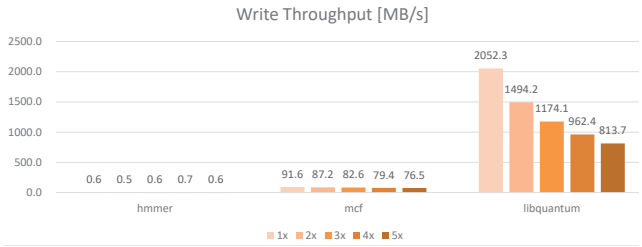


**図 4** エミュレーション時の各ベンチマークのメモリ Write 性能 (図 3 と同様の条件)

セス特性から予想される挙動を示していることが分かる．今後は，より実践的な評価として，STT-MRAM などの実在する NVM の Read/Write レイテンシを再現したときのベンチマークの実行時間を評価する予定である．

## 6. おわりに

本研究では，NVM の Read/Write レイテンシの違いを考慮した，DRAM 環境での NVM エミュレーション手法を提案した．既存の手法では対応できない Read/Write レイテンシが異なる NVM を実機上でエミュレーションする方法を示し，実際に Intel プロセッサに提案手法を実装し，SPEC CPU2006 を用いて評価を行った．初期的な評価実験を通して，提案機構がエミュレーションする NVM の Write レイテンシを大きな値に設定すると，実機上で動作するワークロードの実行時間が増えることを確認した．また，ワークロードのメモリアクセス特性から予想できる挙動を示すことを確認した．今後の課題としては，本論文で未完了である提案手法のエミュレーション精度の評価，および LLC ミスによるライトバック処理以外のメモリ Write(ライトスルー、clflush) によるレイテンシへの対応があげられる．

**謝辞** 本研究は JSPS 科研費 16K00115 の助成を受けたものである．

### 参考文献

[1] ITRS: International technology roadmap for semiconductors 2013 edition, The International Technology Roadmap for Semiconductors (ITRS) (online), available from ⟨http://www.semiconductors.org/clientuploads/Research_Technology/ITRS/2013/2013PIDS.pdf⟩ (accessed 2016-10-25).

[2] Giardino, M., Doshi, K. and Ferri, B.: Soft2LM: Application Guided Heterogeneous Memory Management, *2016 IEEE International Conference on Networking, Architecture and Storage (NAS)*, pp. 1–10 (online), DOI: 10.1109/NAS.2016.7549421 (2016).

[3] 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*, EuroSys '14, New York, NY, USA, ACM, pp. 15:1–15:15 (online), DOI: 10.1145/2592798.2592814 (2014).

[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*, SOSP '09, New York, NY, USA, ACM, pp. 133–146 (online), DOI: 10.1145/1629575.1629589 (2009).

[5] Poremba, M. and Xie, Y.: NVMain: An Architectural-Level Main Memory Simulator for Emerging Non-volatile Memories, *2012 IEEE Computer Society Annual Symposium on VLSI*, pp. 392–397 (online), DOI: 10.1109/ISVLSI.2012.82 (2012).

[6] 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 (online), DOI: 10.1145/2814576.2814806 (2015).

[7] Dong, X., Xu, C., Xie, Y. and Jouppi, N. P.: NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 31, No. 7, pp. 994–1007 (online), DOI: 10.1109/TCAD.2012.2185930 (2012).

[8] Qureshi, M. K., Srinivasan, V. and Rivers, J. A.: Scalable High Performance Main Memory System Using Phase-change Memory Technology, *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, New York, NY, USA, ACM, pp. 24–33 (online), DOI: 10.1145/1555754.1555760 (2009).

[9] Intel: Intel 64 and IA-32 Architectures Software Developer's Manual., Intel Corporation (online), available from ⟨http://www.intel.in/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-manual-325462.pdf⟩ (accessed 2016-10-13).

[10] Intel: Intel Xeon Processor E5-2600 Product Family: Uncore Guide., Intel Corporation (online), available from ⟨http://www.intel.com/content/dam/www/public/us/en/documents/design-guides/xeon-e5-2600-uncore-guide.pdf⟩ (accessed 2016-10-13).

[11] Intel: Intel 64 and ia-32 architectures optimization reference manual., Intel Corporation (online), available from ⟨http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf⟩ (accessed 2016-10-13).

[12] Viswanathan, V.: Intel Memory Latency Checker.,

Intel Corporation (online), available from ⟨https://software.intel.com/en-us/articles/intelr-memory-latency-checker⟩ (accessed 2016-10-13).

[13] 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, pp. 1–7 (online), DOI: 10.1145/2024716.2024718 (2011).