DEIM2022 K34-4

不揮発性メモリ性能測定のための マイクロベンチマークの設計と実装

吉岡　弘隆†　　合田　和生††　　喜連川　優††

† 東京大学　大学院情報理工学系研究科
〒 153–8505 東京都目黒区駒場 4–6–1
†† 東京大学　生産技術研究所
〒 153–8505 東京都目黒区駒場 4–6–1
E-mail: †{hyoshiok,kgoda,kitsure}@tkl.iis.u-tokyo.ac.jp

あらまし　近年製品出荷された不揮発性メモリ（Non Volatile Memory 以下 NVM）の動作特性を測定するマイクロベンチマークを開発した．不揮発性メモリはメインメモリに利用されている揮発性メモリ（DRAM）とことなりデータ永続性を持つ．そこで本マイクロベンチマークでは不揮発性メモリの特性に注目し，メモリアクセスのレイテンシやスループットを計測するだけではなく，各種永続性命令のオーバヘッドの計測も可能にした．インテルのプロセッサにおいて，データのロード命令，ストア命令，永続化（フラッシュ命令），同期（フェンス命令）について細かく測定できるように設計した．さらに，データベース演算の実行コストにも注目し，その中でも特に重要な結合演算の実行コストを測定できるようにした．
従来不揮発性メモリの動作特性をあきらかにするためのマイクロベンチマークは標準的なものが存在しないため自作するか，既存のベンチマークを利用するしかなかった．前者は開発コストがかかり，後者は必要とする機能が不足していた．そこで本提案では，汎用的な不揮発性メモリ向けのマイクロベンチマークを開発した．

キーワード　データベース技術，チューニング，インメモリデータ管理，ストレージ管理性能測定，マイクロベンチマーク，不揮発性メモリ，NVM

1 はじめに

不揮発性メモリと呼ばれるバイト単位アクセス可能なメモリが近年出荷されている [19], [20]. その特徴をまとめると，1) 性能（スループット，レイテンシなど）は SSD (NAND Flash), HDD より高い，DRAM より低い，2) DRAM と異なり永続性がある（電源が遮断されても情報は喪失しない），3) 記憶容量は DRAM より大きい，4) バイト単位のアクセスが可能である（DRAM と同様），5) CPU cache coherent である（DRAM と同様），6) DMA （Direct Memory Access）と RDMA （Remote Direct Memory Access）をサポートする（DRAM と同様），7) ユーザ空間でアクセスできる．SSD や HDD のようにアクセスするために system call は必要がない．アクセスするためにカーネルコード，ページキャッシュ，割込などは必要ないなどがある．このような優れた特性を持つため，今後広く利用されることが期待される．

一方で，従来の揮発性主記憶メモリと違い，不揮発であるという特徴はプログラミングモデルの変更を余儀なくする．そのため，その動作特性は単にスループットやレイテンシーの違いだけでなく，アルゴリズムやデータ構造にも影響を与える．

例えば，クラッシュコンシステンシーという概念は従来は主にハードディスクなど永続性を持つデバイスにおいて議論されてきたが，揮発性メモリではクラッシュした時点でそもそも永続性を保証しないので一般的には問題とならなかった．しかし不揮発性メモリに於いては，ストア命令は必ずしも同期的にメディアに書き込むことを保証しない，通常はキャッシュにのみ書き込み，非同期にメディアに書き込む．そのため，プログラムのストアの順序とメディアへの書き込み順は必ずしも等しくならず，書き込み順の一貫性を担保する仕組みが必要になる．

また，メモリレベルでの永続性を担保する命令（FLUSH 命令）や書き込み順の同期を取る命令（FENCE 命令）などをプログラマが意識しないと不揮発性メモリを利用した正しいプログラム（クラッシュした時点での整合性を持つ）は構築できない．

このように不揮発性メモリは揮発性メモリとは異なったプログラミングモデルを持つために従来とは異なるプログラミング上の注意が必要となる．一般的には，上記のような煩雑な処理をアプリケーションプログラマに委ねるのはアプリケーションをアセンブリ言語で記述するようなものなので，動作の詳細を隠蔽するためにベンダーが提供する PMDK (Persistent Memory Developers Kit) などのライブラリを利用することになる．

しかしながらライブラリなどを利用してのプログラミングでは動作特性が隠蔽されるために性能特性がプログラマにとって自明ではなくなる．そこで本研究ではシステム・ソフトウェア（例えば OS やデータベースシステムなど）やアプリケーション・プログラムを構築するプログラマ向け性能測定を目的とし

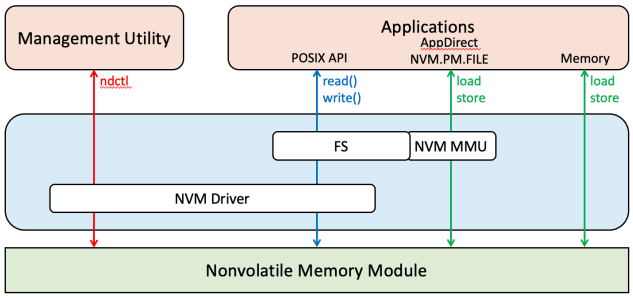

図 1　NVM プログラミングモデル

たマイクロベンチマークを提案する．

まず NVM プログラミングモデルを概説し，次に本研究に先立って実施した，NVM の性能特性評価について触れる．

1.1　NVM プログラミングモデル

アプリケーション・プログラムないしシステム・ソフトウェアからみた NVM のプログラミングモデルはストレージの業界団体である SNIA [23] が NVM Programming Model Specification として定義し公開している [22]．図 1 で示した．

Intel DCPMM は DRAM と同様に DIMM(dual-inline memory module) のパッケージとして提供されていて，システムメモリバスに挿入する．このクラスの DIMM をここでは NVDIMM(Non-Volatile DIMM) とよぶ．

SNIA のプログラミングモデルでは，NVM.PM.FILE モードと呼ばれるモードによってユーザスペース・アプリケーションが直接 NVM をメモリとしてアクセスすることを可能にする．図は NVM.PM.FILE モードの例を示す．PM をサポートするファイル・システムはメモリーマップ・ファイルによって，ユーザスペースから直接メモリへ load/store できる．

メモリマッピングが発生すると，MMU(memory management unit) 経由でメモリにアクセスすることになる．この場合，従来のファイルへの POSIX API 例えば read()/write() によるアクセスとことなり，システムコールが不要になるばかりではなく，OS による IO のオーバヘッドがなくなる

これらの機能は Linux ないし Windows など複数の OS によってすでに実装されている．また NVM.PM.FILE モードも Linux では DAX (Direct Access) モードとして実装されている．Linux の XFS や ext4 ファイルシステムなどは DAX モード (AppDirect モード) をサポートしていて，mmap() のようなシステムコールを利用することによって，ファイルの内容を直接ユーザーメモリ空間にマッピングすることができる．一度メモリ空間にマップされると OS の提供する I/O 機能，すなわち read()/write() システムコールなどを利用しなくても直接 NVM を利用することができる．NVM への読み書きは load/store で行う．この場合，システムコールが必要ないので，コンテキストスイッチ，割り込みなどが発生しない．

1.2　マイクロベンチマーク

NVM の性能特性については，本研究に先立って，著者らは Intel Optane DC Persistent Memory Module (DCPMM) [10] を対象に，その動作特性を明らかにするマイクロベンチマークを作成し，評価実験を行ない，下記のような動作特性を明らかにした [31] ．この実験では一回で 32 byte 転送する VMOVDQA 命令を利用して 64 byte の転送を計測した．

- レイテンシ
  - DRAM (load, fence 組み合わせ) 309ns
  - NVM (load, fence 組み合わせ) 972ns
  - DRAM (store, fence 組み合わせ) 76ns
  - NVM (store, fence 組み合わせ) 78ns
- スループット（順アクセス）
  - DRAM (load, fence 組み合わせ) 19.9GB/sec
  - NVM (load, fence 組み合わせ) 14.3GB/sec
  - DRAM (store, CLFLUSHOPT 組み合わせ) 26.1GB/sec
  - NVM (store, CLFLUSHOPT 組み合わせ) 19.4GB/sec
- スループット（ランダムアクセス）
  - DRAM (load, fence 組み合わせ) 0.584GB/sec
  - NVM (load, fence 組み合わせ) 0.201GB/sec
  - DRAM (store, CLFLUSHOPT 組み合わせ) 3.781GB/sec
  - NVM (store, CLFLUSHOPT 組み合わせ) 1.365GB/sec

load も store も順アクセスに比べてランダムアクセスの場合，極端にスループットが低くなっていることが確認できる．

上記のマイクロベンチマークでは NVM の load/store のレイテンシやスループットは明らかになったが，それが実際のアプリケーションでどのような動作特性を持つのか必ずしも自明ではないので，データベース演算の実行コスト，特に結合演算の実行性能に注目しマイクロベンチマークを作成し計測した [30], [32].

論文の構成：本論文は 2 章で本研究で提案するマイクロベンチマークの設計と実装，続く 3 章でその評価，4 章で関連研究について述べ，5 章でまとめと今後の課題について記す．

本研究の成果は Intel Optane DC Persistent Memory Module (DCPMM) のレイテンシーやスループットの性能測定だけではなく，データベースの典型的な演算（Hash join）コストの性能測定を可能にするマイクロベンチマークを実装したことである．本マイクロベンチマークを利用することによって NVM 向けのシステム・ソフトウェアやアプリケーション・プログラムの性能測定が容易に行えるようになった．

2　設計と実装

前章で示したとおり，NVM の性能特性を明らかにするマイクロベンチマークとデータベース演算の実行コスト，特に結合演算の性能特性計測のマイクロベンチマークを実装した．

2.1　load および store のレイテンシおよびスループット

Intel 64 および IA-32 アーキテクチャ命令セットの load, store，flush，fence 命令について概説する [8]．Intel 64 および

IA-32 命令セットには Intel 8080 プロセッサからの互換性を維持するために様々な命令があり，その組み合わせパターンは必ずしも自明ではない．

load および store は MOV というニーモニックを持つ．また MOV 命令にはキャッシュを経由しない Non Temporal MOV 命令がある．store はレジスタの内容をメモリに書き出すが，通常は CPU 内蔵のキャッシュに書き出すだけである．キャッシュからメモリへ書き出すタイミングはハードウェアないしソフトウェアで制御される．明示的にキャッシュからメモリへ書き出すため flush を利用する [8]．メモリに書き出す前に電源断が起こるとキャッシュの内容は消失する（永続化されない）．

Intel 64 および IA-32 アーキテクチャはアウトオブオーダー実行のため，任意の命令の実行順序は保証されない．それは fence 命令によって実行順序を制御する．

計測方法：Intel 64 および IA-32 アーキテクチャは TSC（time stamp counter）と呼ばれる，単調増加の 64 bit の MSR（model specific register）を持つ．TSC は電源投入時にゼロリセットされ，ハードウェアクロックごとに１づつ増加していく [7], [8], [9]．実行時間計測には TSC を読み取る RDTSCP 命令を利用した．RDTSCP 命令は非特権命令なのでユーザモードから実行でき，オーベヘッドも小さい．

計測項目：レイテンシおよびスループットを順アクセス，ランダムアクセスで計測した．store において各種 store 命令，flush 命令，fence 命令の組み合わせでの計測を行えるようにした．詳細については [31] を参照されたい．

2.2 データベース演算の実行コスト

ここではデータベース演算の中で特に重要な結合演算 (Join) をとりあげる．関係データベースでは結合演算以外にも様々な演算があるが代表的な演算で実行コストが大きいために取り上げた．また結合演算のアルゴリズムも様々あるが，hash 結合を利用した [30], [31]．

ここでは簡単のために，表 1 に示すような R 表と S 表という 2 つの表を結合することを考える．それぞれのサイズは S 表が R 表より大きい想定で R 表の key と S 表の外部キー id を結合する．Hash 結合では，大きく分けて，build フェーズ，次に probe フェーズという２つのフェーズで処理を行う．

build フェーズではまず R 表のキーを読み込み結合用表 (hash 表と呼ぶ) を build する．R 表のタプルを読み込み，そのキー値で hash 値を求め（hash 値，キー値，R 表の値）の三組を hash 表に登録する．hash 値が重複した場合，リンクリストを作成し（キー値，R 表の値）を挿入する．R 表のすべてのタプルの hash 値を求めて hash 表に挿入したら build フェーズは終了となる．次に probe フェーズでは S 表の R 表に対する外部キーで hash キーを求め先程生成した hash 表と結合する．

build フェーズ，probe フェーズそれぞれについてマルチスレッド化し，その実行コストを検証した．

表 1　R 表と S 表

表名	カラム名	データ型（サイズ）	備考
R 表	Key	long (8 byte)	ユニークキー
	value	char (16 byte)	文字列
S 表	Key	long (8 byte)	ユニークキー
	value	int[16] (16*4 byte)	int 型配列
	id	long (8 byte)	R 表 key への参照

表 2　実験環境 [31]

CPU model	Intel Xeon Silver 4215, 2.5GHz
	8 core, 2 socket
No. of nodes	2
Threads per core	2
Cache	L1d 32KiB, L1i 32 KiB,
	L2 1 MiB, L3 11 MiB (shared)
DRAM	32 GiB * 12 (384 GiB)
DCPMM (NVM)	128 GiB *12 (1536 GiB)
OS	CentOS 7.7.1908, linux kernel 3.10
PM library	pmdk 1.8
File System	XFS V5 DAX enabled

3 評　　価

3.1 ハードウェア，ソフトウェア構成

前章で示したモデルに従って Intel DCPMM で評価実験を行った．実験環境を表 2 に示した．Hyper-threading を有効にして実験したので，論理コア数は 32(=8*2*2) である．

3.2 評 価 項 目

3.2.1 レイテンシ及びスループット

ここではレイテンシの計測結果を図 2 に示す．この例では，store 命令の後に fence 命令を実行したもの（図中 fence で示す，以下同様）．Non-Temporal store 命令に fence 命令を実行（nts-f），CLFLUSH 命令に fence 命令（clflush-f），CLWB 命令に fence 命令（clwb-f），CLFLUSHOPT 命令に fence 命令（clfopt-f），MOVQS 命令に fence 命令（movqs-f）をそれぞれ示している．それぞれの flush 命令の動作は次のようになる．CLFLUSH はキャッシュを無効化しオーダー実行，CLWB はキャッシュを無効化せずアウトオブオーダ実行，CLFLUSHOPT はキャッシュを無効化しアウトオブオーダ実行となる．キャッシュが無効化されると同じアドレスにアクセスした場合キャッシュミスが発生する．CLWB のようにキャッシュを無効化しないと，その内容にアクセスした場合，キャッシュヒットする．どの命令を選択するべきかはアプリケーションのユースケースによる．flush のコストが顕著に生じているのが確認できる．詳細については [31] で報告したので，参考にしていただきたい．

3.2.2 データベース演算実行コスト

ハッシュ結合演算について build フェーズ，probe フェーズについてその実行コストを計測した．

build フェーズに於いてスレッド数を変化してみたが，スケールしなかった．ハッシュ表を生成するとき，ハッシュ表への挿

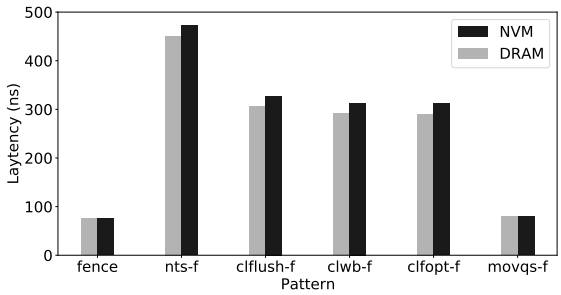

図 2　書込み（ストア fence）レイテンシ [31]

入が発生するが複数のスレッドが並行して行うので適切な同期処理が必要となる．ハッシュ表に対して一つのロックを用意してジャイアントロックで同期制御を行った．同期を取るにあたって，pthread spin lock，pthread mutex lock，および test and set で spin lock (gcc __sync_lock_test_and_set() を実装したものと__sync_add_and_fetch() および pause 命令を利用したもの) の 4 つで比較評価した．

ハッシュ表に対して一つのロックを用意してジャイアントロックで同期制御を行った．図 3 および図 4 にそれぞれの実行時間を示した．図 3 はハッシュ表を DRAM に生成した場合，図 4 はハッシュ表を NVM に生成した場合になる．TAS は gcc __sync_lock_test_and_set()，を利用して test and set をナイーブに実装した．AAF は gcc __sync_add_and_fetch() および pause を利用した．spin loop するときに，最初にロック変数の値をチェックしロックが掛かっているときには pause 命令を発行し，loop する．ロックが掛かっていないときはアトミックに add and fetch をしてロックを取得するというアルゴリズムで spin lock を実装した．pthread spinlock は pthread_spin_lock()/pthread_spin_unlock() を利用，pthread mutexlock は pthread_mutex_lock()/pthread_mutex_unlock() をそれぞれ利用した場合を示す．

どのロック方式をとってもスレッドが増えると経過時間が増え，性能が悪化している．処理を分割し経過時間をへらすのではなく，同期のオーバヘッドのほうが大きくなってしまった．ハッシュ表に対するジャイアントロックは明らかにスケーラビリティがないと言える

probe フェーズに於いてスレッド数を 1 から 64 まで変化させ結合させてみた．シングルスレッドでは 1 秒あたり NVM で 2189K タプル，DRAM で 5899K タプル処理した．横軸にスレッド数をとった結果を図 5 に示す．32 スレッドで，それぞれ 53.23M タプル，118.3M タプル処理した．NVM は DRAM と比較して 44%程度の性能だった．

この実行モデルでは更新はなく読み込み（read）のみである．また hash 表および S 表に対するロックや競合は発生しない．論理コア数までスケールした．

4　関 連 研 究

ストレージのサーベイとして [6] がある．フラッシュメモリ

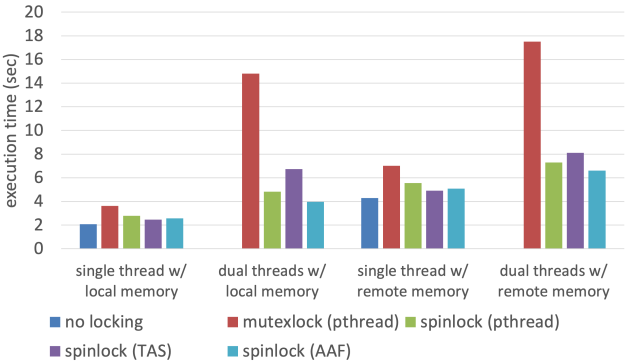

図 3　ハッシュ表のビルドに要した実行時間（ハッシュ表を DRAM に配置）

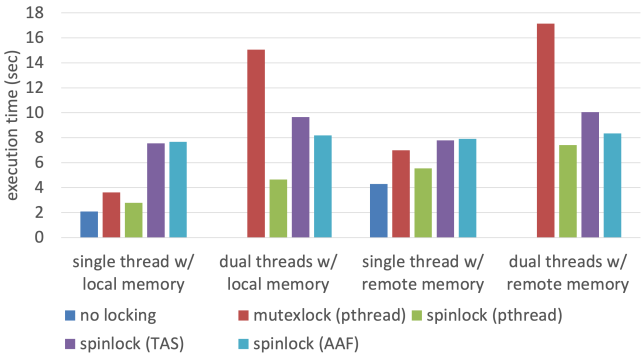

図 4　ハッシュ表のビルドに要した実行時間（ハッシュ表を NVM に配置）

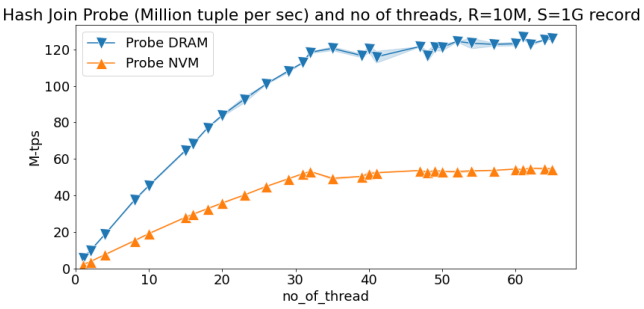

図 5　Hash Join Probe Million Tuple Per Second

の基本性能特性については [25] がある．NVM に関する研究は 2010 年代前半ころから様々な提案がされている．

Intel Opetane DCPMM そのものの性能評価は [13], [24] がある．多くの実装研究は実機がなかったためシミュレーションによって評価をしていた [1]．近年，Intel Optane DCPMM の出荷に伴い，徐々にその性能特性について評価が発表されてきている．

先行研究の提案を実機で検証しシミュレーション結果との差分を報告している [29]．Intel Optane DCPMM のレイテンシとスループットだけではなく，電力消費量も評価している [18]．通常 NVM のほうが DRAM よりエネルギ効率はよいが，キャッシュを経由しない (Non-Temporal)store の場合，エネルギ効率

が悪くなる場合がある．WHISPER とよぶ Persistent Memory（PM）のベンチマークを開発し評価したところ，a)4%が PM で残りは volatile memory への書き込みだった，b) ソフトウェアトランザクションは 5 から 50 のオーダリングポイントを持つが，永続性を必要とするのは最後の一つだった，c)75%のアップデートは 64B キャッシュライン，d)80%の書き込みは同じスレッドの以前の書き込みに依存する，他のスレッドの書き込みに依存するのは少ない，というようなことを示した [16]．Intel の NVM 向けライブラリ（NVML）についての初期の評価は [21] にある．当該ライブラリは現在，pmdk (persistent memory development kit) [11] になった．

既存のデータ構造を NVM に拡張し評価したものとして [14] などがある．B^+Tree 及びその NVM 拡張系の提案として，Persistent B^+-Trees [5],FPTree [17],Bztree [2]，また索引データ構造の比較検討 [12], [15] は従来提案されていたものを DCPMM で検証し，その有効性などを定量的に確認している．ファイルシステム NOVA [28],Hikv(KVS) [27],DBMS の実装 [3],Write-behind logging [4] などがある．

High Performance Computing 分野での評価は [26] があり大規模アプリケーションでの性能を報告している．

データベース演算の実行コスト特に Hash Join を NVM に適用し計測した先行研究は報告されていない．

5 まとめと今後の課題

不揮発性メモリデバイスを対象とする性能特性測定のためのマイクロベンチマークの設計と実装について報告した．先行研究では不揮発性メモリデバイスのレイテンシーならびにスループットが報告されている．また，アプリケーションを不揮発性メモリデバイス向けに変更し性能特性を計測した結果なども報告されている．しかしながらデータベース演算のプリミティブについて不揮発性メモリデバイスを適用した場合の性能特性は報告されていない．本提案の貢献である．今後はデータベース演算のみならず，各種データ及びアルゴリズムでの性能特性評価用に本研究での成果を拡張していく．

6 謝　　辞

本研究の一部は，日本学術振興会科学研究費補助金 20H04191 の助成を受けたものである．

文　　献

[1] Joy Arulraj, Justin Levandoski, Umar Farooq Minhas, and Per-Ake Larson. Bztree: A high-performance latch-free range index for non-volatile memory. *Proceedings of the VLDB Endowment*, Vol. 11, No. 5, pp. 553–565, 2018.

[2] Joy Arulraj, Justin Levandoski, Umar Farooq Minhas, and Per-Ake Larson. Bztree: A high-performance latch-free range index for non-volatile memory. *Proc. VLDB Endow.*, Vol. 11, No. 5, p. 553–565, January 2018.

[3] Joy Arulraj and Andrew Pavlo. How to build a non-volatile memory database management system. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, p. 1753–1758, New York, NY, USA, 2017. Association for Computing Machinery.

[4] Joy Arulraj, Matthew Perron, and Andrew Pavlo. Write-behind logging. *Proceedings of the VLDB Endowment*, Vol. 10, No. 4, pp. 337–348, 2016.

[5] Shimin Chen and Qin Jin. Persistent b^+-trees in non-volatile main memory. *Proc. VLDB Endow.*, Vol. 8, No. 7, p. 786–797, February 2015.

[6] Kazuo Goda and Masaru Kitsuregawa. The history of storage systems. *Proceedings of the IEEE*, Vol. 100, No. Special Centennial Issue, pp. 1433–1440, 2012.

[7] Intel. *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture*, May 2019. Order Number: 253665-070.

[8] Intel. *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 2 (2A, 2B, 2C & 2D): Instruction Set Reference, A-Z*, May 2019. Order Number: 325383-070.

[9] Intel. *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3 (3A, 3B, 3C & 3D): System Programming Guide*, May 2019. Order Number: 325384-070.

[10] Intel. Intel Optane Persistent Memory. https://www.intel.com/content/www/us/en/products/docs/memorystorage/optane-persistent-memory/optane-dc-persistent-memorybrief.html, 2019.

[11] Intel. Persistent Memory Development Kit. https://pmem.io/pmdk/, 2019.

[12] Abdullah Al Raqibul Islam, Anirudh Narayanan, Christopher York, and Dong Dai. A performance study of optane persistent memory: From indexing data structures' perspective. In *36th International Conference on Massive Storage Systems and Technology (MSST 2020)*, pp. 2–2, 2020.

[13] Joseph Izraelevitz, Jian Yang, Lu Zhang, Juno Kim, Xiao Liu, Amirsaman Memaripour, Yun Joon Soh, Zixuan Wang, Yi Xu, et al. Basic performance measurements of the intel optane dc persistent memory module. *arXiv preprint arXiv:1903.05714*, pp. 1–61, 2019.

[14] Se Kwon Lee, Jayashree Mohan, Sanidhya Kashyap, Taesoo Kim, and Vijay Chidambaram. Recipe: converting concurrent DRAM indexes to persistent-memory indexes. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pp. 462–477, 2019.

[15] Lucas Lersch, Xiangpeng Hao, Ismail Oukid, Tianzheng Wang, and Thomas Willhalm. Evaluating persistent memory range indexes. *Proceedings of the VLDB Endowment*, Vol. 13, No. 4, pp. 574–587, 2019.

[16] Sanketh Nalli, Swapnil Haria, Mark D. Hill, Michael M. Swift, Haris Volos, and Kimberly Keeton. An analysis of persistent memory use with whisper. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '17, p. 135–148, New York, NY, USA, 2017. Association for Computing Machinery.

[17] Ismail Oukid, Johan Lasperas, Anisoara Nica, Thomas Willhalm, and Wolfgang Lehner. FPTree: A hybrid scm-dram persistent and concurrent b-tree for storage class memory. In *Proceedings of the 2016 International Conference on Management of Data*, pp. 371–386, 2016.

[18] Ivy B. Peng, Maya B. Gokhale, and Eric W. Green. System evaluation of the intel optane byte-addressable nvm. In *Proceedings of the International Symposium on Memory Systems*, MEMSYS '19, p. 304–315, New York, NY, USA, 2019. Association for Computing Machinery.

[19] Andy Rudoff. Persistent memory programming. *Login: The Usenix Magazine*, Vol. 42, No. 2, pp. 34–40, 2017.

[20] Steve Scargall. *Programming Persistent Memory A Comprehensive Guide for Developers*. Apress, Berkeley, CA, 2020. https://doi.org/10.1007/978-1-4842-4932-1.

[21] H. Shu, H. Chen, H. Liu, Y. Lu, Q. Hu, and J. Shu. Empirical study of transactional management for persistent memory. In *2018 IEEE 7th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, pp. 61–66, 2018.

[22] SNIA. NVM Programming Model. https://www.snia.org/sites/default/files/technical_work/final/NVMProgrammingModel_v1.2.pdf, 2017.

[23] SNIA. SNIA Storage Networking Industry Association. https://www.snia.org, 2017.

[24] Alexander van Renen, Lukas Vogel, Viktor Leis, Thomas Neumann, and Alfons Kemper. Persistent memory i/o primitives. In *Proceedings of the 15th International Workshop on Data Management on New Hardware*, DaMoN'19, pp. 1–7, New York, NY, USA, 2019. Association for Computing Machinery.

[25] Yongkun Wang, Kazuo Goda, Miyuki Nakano, and Masaru Kitsuregawa. Performance evaluation of flash SSDs in a transaction processing system. *IEICE transactions on information and systems*, Vol. 94, No. 3, pp. 602–611, 2011.

[26] Michèle Weiland, Holger Brunst, Tiago Quintino, Nick Johnson, Olivier Iffrig, Simon Smart, Christian Herold, Antonino Bonanni, Adrian Jackson, and Mark Parsons. An early evaluation of Intel's optane DC persistent memory module and its impact on high-performance scientific applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–19, 2019.

[27] Fei Xia, Dejun Jiang, Jin Xiong, and Ninghui Sun. Hikv: A hybrid index key-value store for dram-nvm memory systems. In *2017 {USENIX} Annual Technical Conference ({USENIX}{ATC} 17)*, pp. 349–362, 2017.

[28] Jian Xu and Steven Swanson. {NOVA}: A log-structured file system for hybrid volatile/non-volatile main memories. In *14th {USENIX} Conference on File and Storage Technologies ({FAST} 16)*, pp. 323–338, 2016.

[29] Jian Yang, Juno Kim, Morteza Hoseinzadeh, Joseph Izraelevitz, and Steve Swanson. An empirical guide to the behavior and use of scalable persistent memory. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pp. 169–182, 2020.

[30] 吉岡弘隆, 合田和生, 小沢健史, 喜連川優. 不揮発メモリデバイスを対象とする結合演算のマルチスレッド実行性能に関する実験と一考察. *DEIM2021*, pp. B25–1, 2021.

[31] 吉岡弘隆, 合田和生, 喜連川優. 不揮発メモリデバイスの性能評価のためのマイクロベンチマークに関する初期検討. 研究報告データベースシステム (DBS), Vol. 120, No. 158, pp. 7–12, 2020.

[32] 吉岡弘隆, 合田和生, 喜連川優. 不揮発メモリデバイスを対象とするデータベース演算の実行コスト測定方式に関する検討. 信学技法 (DE), Vol. 120, No. 305, pp. 36–41, 2020.