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

吉岡　弘隆† 　小沢　健史†† 　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,ozawa,kgoda,kitsure}@tkl.iis.u-tokyo.ac.jp

あらまし　近年不揮発性メモリデバイス（NVM）が注目を浴びている．SSD/HDD のように不揮発性でありながら DRAM のように高速にバイト単位でアクセスが可能である．OS やデータベース管理システムは主記憶が揮発性であることを前提に構築されている．しかしながら NVM のデータベースにおける動作特性（レイテンシ，スループットなど）は未だに十分に明らかになっているとは言えない．そこで NVM を利用した場合のデータベース演算，特に結合演算実行性能に関して報告する．ビルドフェーズにおいては十分なスケーラビリティを確認できなかった．

キーワード　NVM, DBMS, microbenchimarking, database operation cost

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 [16], [17]. 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．

通常のメモリデバイスとして考えた場合，従来の製品の進化はいわゆるムーアの法則にのっとった，デバイスの高密度化（大規模化），高速化，低価格化などでソフトウェアからみたアーキテクチャの質的変化はほとんどなかった．

一方，NVM の場合は，単なるバイト単位アクセス可能なメモリとして見ることもできるが，記憶内容が不揮発であるという質的な変化がある．そのため，主記憶メモリが揮発性であるという前提で構築されている，OS，ファイルシステム，データベース管理システムなどで根本的な再構築が必要になる．

しかしながら NVM の動作特性は，NVM 製品の一つである Intel Optane 製品の出荷まもないということもあり未だに十分に明らかになっているとは言えない．またデータベースに適応した動作特性の報告も多くない．そこで本研究に先立ち，NVM の製品である Intel Optane DC Persistent Memory Module (DCPMM) [7] 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 [27], [28] ．

- 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
  - 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

load レイテンシは NVM は DRAM の 3 倍程度だか，store レイテンシは同程度である．一方，load スループットは NVM は DRAM の 7 割弱，store スループットは 7 割強である．このような動作特性が実際のデータベース管理システムやアプリケーションにどのような影響を及ぼすか十分に明らかになっているとは言えないのでデータベース管理システムの演算コスト，特に結合演算 (Hash Join Probe フェーズ) の実行性能に注目し計測した [28]．今回は，Hash Join の Build フェーズにおける実行性能について報告する．

本論文は 2 章で NVM のプログラミングモデルを紹介し，NVM の実装である Intel DCPMM を紹介する．続く 3 章でデータベース演算の実行コストのモデルを示す．4 章で評価実験の詳細を示し，5 章で関連研究，6 章で最後にまとめと今後の課題を記す．

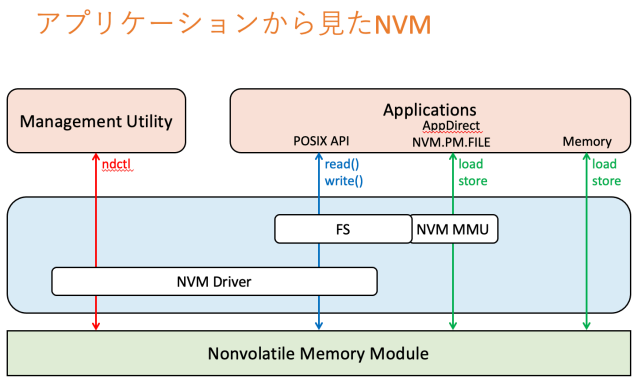

図 1　NVM プログラミングモデル

2　NVM プログラミングモデル

本章では，アプリケーションないしシステム・ソフトウェアから見た NVM のプログラミングモデルについて述べる．こ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 [20] が NVM Programming Model Specification として定義し公開している [19].

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 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.

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

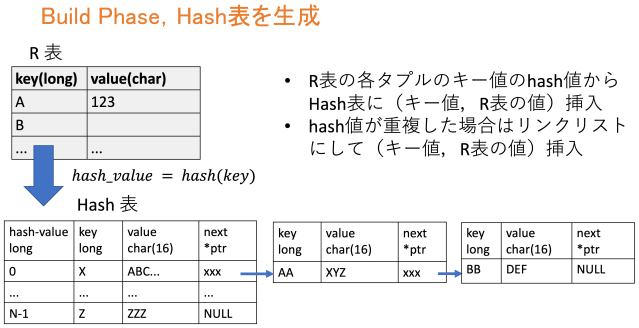

図 2　Build フェーズ

3　データベース演算の実行コスト

本章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 join を利用した.

ここでは簡単のため表 1 に示すような 2 つの表を join するモデルを考える．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 join する．Hash Join 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.

図 2 で示したように build フェーズではまず R 表のキーを読み込み join 用表 (ハッシュ表と呼ぶ) をビルドする．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）の三組をハッシュ表に登録する．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 表に挿入したらビルドフェーズは終了となる．次に図 3 で示したように 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ハッシュ表と join をする.

今回 hash join の build フェーズをマルチスレッド化しその効果を検証した．先行実験で probe フェーズをマルチスレッド化して検証した．スレッド数を増加させ性能評価したところ，スレッド数に比例して論理コア数までスループットが向上した [28].

3.1　build フェーズのマルチスレッド化

build フェーズではハッシュ表を生成する．マルチスレッド化するにはハッシュ表の更新に関して同期をとる必要がある．同期

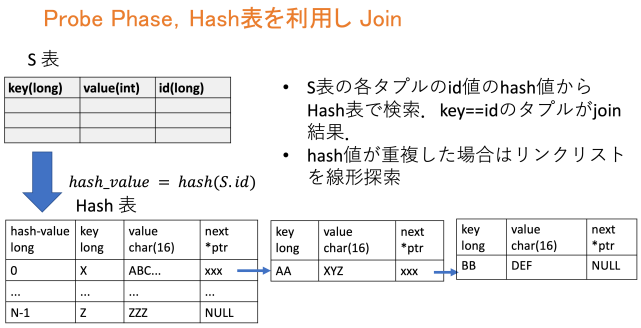

図 3　Probe フェーズ

表 2　実験環境 [27]

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

をとるにあたって，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．

4　評 価 実 験

4.1　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 (NVM) 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．

4.2　データ生成

R 表と S 表のタプルをそれぞれ生成した．S 表の外部キー (S.id) の値は R.key の値域を取るように一様乱数で生成した．外部キーの分布に偏りがないので，キャッシュミスの割合は偏りがある場合に比べて高くなることが期待される．R.key は重複なしの整数とした．R.value と S.value はそれぞれ char(16), int[16] の定数とした．今回の実験結果は R 表のタプル数が 5000 万件，S 表のタプル数が 10 億件で，データサイズはそれぞれ 1.2GB，80GB となった．

4.3　Build フェーズ

R 表をマルチスレッドで実行するために，R 表をスレッド数で水平分割した．例えば，10 スレッドの場合，各スレッドは 500 万タプル処理する．R 表は SSD ないし NVM から read()

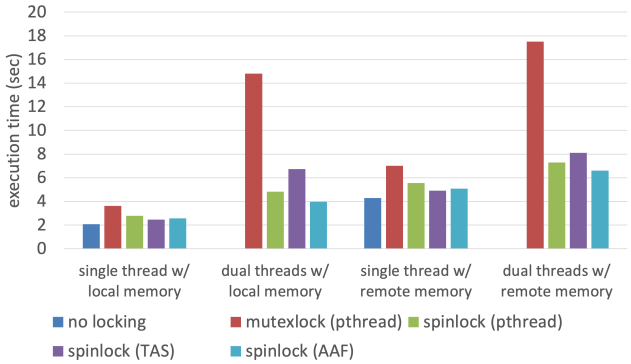

図 4　ハッシュ表のビルドに要した実行時間（ハッシュ表を DRAM に配置）

してメモリ空間に展開した．各スレッドは R 表のタプルごとに hash 値を求め，ハッシュ表の当該 hash 値を持つエントリに挿入していく．もしすでにエントリーが存在していたら，すなわち同じ hash 値だったら，リンクリストにして挿入する．すべてのタプルをハッシュ表に挿入し終わったら build フェーズは終了する．

ハッシュ表への挿入は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になる．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．図 4 および図 5 にそれぞれの実行時間を示した．図 4 はハッシュ表を DRAM に生成した場合，図 5 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．

5　関 連 研 究

ストレージのサーベイとして [6] がある．フラッシュメモリの基本性能特性については [22] 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 [10], [21] 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

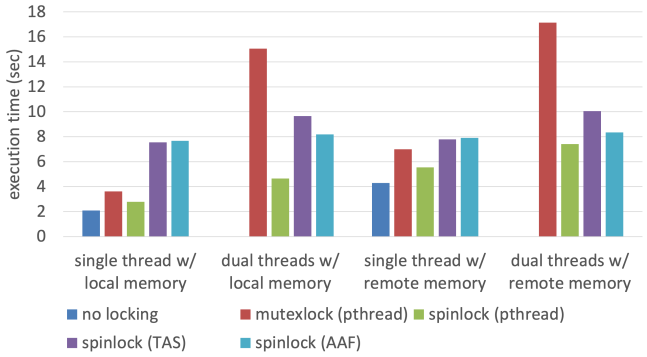

図 5　ハッシュ表のビルドに要した実行時間（ハッシュ表を NVM に配置）

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 [26]．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 [15]．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 [13]．Intel の NVM 向けライブラリ（NVML）についての初期の評価は [18] にある．当該ライブラリは現在，pmdk (persistent memory development kit) [8] 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 [11] 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 [14], Bztree [2]，また索引データ構造の比較検討 [9], [12] 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 [25], Hikv(KVS) [24], DBMS の実装 [3], Write-behind logging [4] などがある．

High Performance Computing 分野での評価は [23] 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．

6　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データベース演算の実行コスト測定方法，特に Hash Join の測定方法ついて報告した．Build フェーズのマルチスレッド化は，ジャイアントロックを利用した場合，全くスケールしなかった．

今後の課題として，NVM の永続化特性の解明とそれの活用について検討していく．

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

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

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

[18] 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.

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

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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. 信学技法 (DE), Vol. 120, No. 158, pp. 7–12, 2020.

[28] 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.