一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
DE2023-22(2023-09)

不揮発メモリを対象とする空間索引の永続化手続きの効率化手法

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　不揮発メモリ (PMEM) は揮発メモリ (DRAM) とほぼ同等のレイテンシーを持ちながらも HDD/SSD 同様永続性を持つ．不揮発メモリへの書き込みは OS が非同期で行うが一貫性を担保したい場合は永続性を持たせるために明示的な flush および同期 (fence) が必要となる．空間索引構造の作成においてデータの挿入はノードのアップデートが必要となるが従来の手法では冗長な書き込みが発生した．空間索引構造を作成するコストの削減を目的に本提案では書き込みを最適化することによって永続化コストを削減し空間索引構造作成の実行時間を短縮した．

キーワード　データベース技術，不揮発性メモリ，NVM，空間索引

An efficient method of persistence procedures for spatial indexes on non-volatile memory

Hirotaka YOSHIOKA[†], Kazuo GODA[††], and Masaru KITSUREGAWA[††]

† Graduate School of Information Science and Technology, The University of Tokyo
4–6–1 Komaba, Meguro-ku, Tokyo, 153–8508 Japan
†† Institute of Industrial Science, The University of Tokyo
4–6–1 Komaba, Meguro-ku, Tokyo, 153–8508 Japan
E-mail: †{hyoshiok,kgoda,kitsure}@tkl.iis.u-tokyo.ac.jp

1. はじめに

近年，不揮発メモリと呼ばれるバイト単位アクセス可能なメモリが出荷されている．主記憶メモリに利用される DRAM のような高速性とストレージのような永続性を備え持つデバイスとして注目を浴びている．不揮発メモリの基本的な動作特性（レイテンシー，スループット等）やデータベースへの適応事例などが紹介され B+ 木やハッシュ索引への実装についての報告もされてきている．しかしながら不揮発メモリを対象とする空間検索構造の実装方式の報告については FBR [9], PMR [21] などに見られる程度である．そこで本研究では不揮発メモリを対象とする空間検索構造の実装方式の検討を行い，永続化手続きの効率化手法を提案する．.

Intel Optane DC Persistent Memory Module （DCPMM）のような不揮発メモリと呼ばれるバイト単位アクセス可能なメモリは DRAM と同程度の機能をもちつつも，SSD (Solid State Disk) ないし HDD (Hard Disk Drive) のような不揮発性 (永続性) を持つ．しかも SSD ないし HDD よりはるかに高速で DRAM よりわずかに遅い程度のレイテンシを持つ [32], [34].

その特徴をまとめると，

- 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，
- 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）
- 記憶容量は DRAM より大きい
- バイト単位のアクセスが可能である（DRAM と同様）
- CPU cache coherent である（DRAM と同様）
- DMA （Direct Memory Access）と RDMA （Remote Direct Memory Access）をサポートする（DRAM と同様），
- 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．

様々な興味深い特徴を持つが，Intel Optane DCPMM 製品出荷後，広く実用化されているとは言い難く，研究段階での提案にとどまっているのが現状である．その原因の一つは NVM を利用するためには，ソフトウェアを NVM 向けに変更する必要があり，OS や DBMS などを NVM 向けに変更することは必ずしも自明ではない．従来ハードウェアの進化は主に半導体製造プロセスの発展（いわゆるムーアの法則による微細化，高速化など）に起因するもので，それらは基本的にはソフトウェアの改変を必要としなかった．例えば，メインメモリの巨大化はムーアの法則によって 18 ヶ月〜2 年程度で倍になるが，既存のソフトウェアの変更なしでも，そのメリットを享受することができた．

一方で，NVM は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) [17] などのライブラリを利用することになる．

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 [35] が NVM Programming Model Specification として定義し公開している．図 1 で示した．

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．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．Intel x86 アーキテクチャの場合，通常の MOV 命令になる．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．

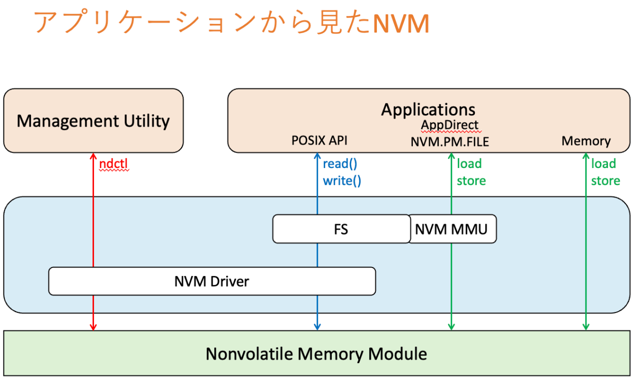

図 1　NVM プログラミングモデル

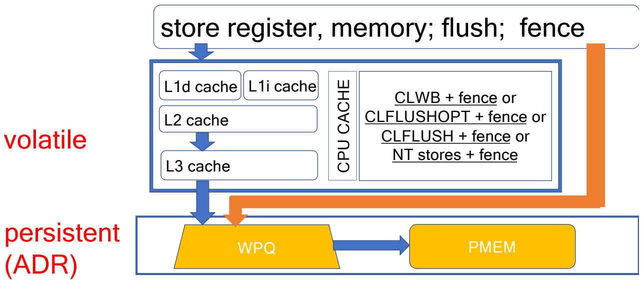

図 2　CPU とキャッシュ，メモリの関係

一方で，従来の揮発性メモリと違い，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，その動作特性の違いは単にスループットやレイテンシーだけでなく，アルゴリズムやデータ構造のあり方にも影響を与える．

不揮発メモリは従来のメモリ同様にバイト単位でアクセス可能であるが，CPU から見て不揮発メモリへのアクセスは通常キャッシュ経由でなされる．図 2 に示すようにキャッシュは揮発性なので永続性を必要とする場合は適宜キャッシュ内容を flush する必要がある．Intel x86 の場合，CLFLUSH 命令などによって明示的に行う．flush 関連の命令は CLFLUSH の他，CLWB，CLFLUSHOPT があり，キャッシュをバイパスする Non-temporal 命令もある．また，メモリへの書き込みを同期するために fence が必要になる．

不揮発メモリは従来のメモリ同様にバイト単位でアクセス

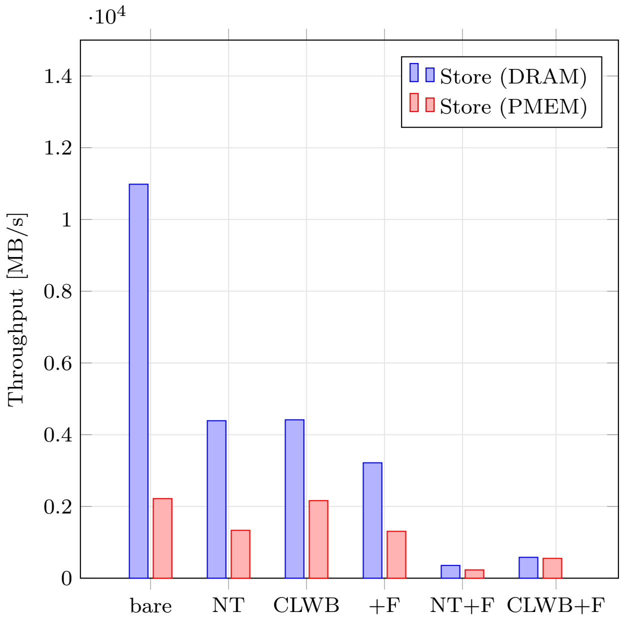

図 3　64 byte sequenial store，bare，NT，CLWB はデフォルトの store 命令および Non-temporal 命令，CLWB 命令を追加したもの，＋ F はさらに FENCE 命令を追加したものを示す [43]

可能であるが，CPU から見て不揮発メモリへのアクセスは通常キャッシュ経由でなされる．図 2 に示すようにキャッシュは揮発性なので永続性を必要とする場合は適宜キャッシュ内容を flush する必要がある．Intel x86 の場合，CLFLUSH 命令などによって明示的に行う．flush 関連の命令は CLFLUSH の他，CLWB，CLFLUSHOPT があり，キャッシュをバイパスする Non-temporal 命令もある．また，メモリへの書き込みを同期するために fence が必要になる．

また記憶の永続性を担保する動作についてはファイルシステムでは本質的な問題だが，メインメモリに於いては永続性は存在しなかったので一般的な問題ではなかった．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．しかし不揮発メモリに於いてはストア命令によって記憶が永続化されるので，ディスクへの書き込みと同様な問題が生じる．ここで言うクラッシュコンシステンシーとは，情報の変更を首尾一貫した状態で行うことを指す．

またこのような各種の flush や fence が性能に与える影響は必ずしも自明ではない．そこでわれわれは，先行研究として pmmeter というマイクロベンチマークを作成し，flush 命令と同期命令が与えるオーバーヘッドを評価した ([43])．64 バイトのシーケンシャルアクセスのスループットは DRAM で 10.9GB/s，PMEM で 2.2GB/s だが flush 命令を追加すると 4.4GB/s と 2.1GB/s にそれぞれ低下する．さらに fence 命令を追加すると 0.58GB/s と 0.55GB となる (図 3 参照)．flush と fence は大きなコストが発生する．

そのような特性を踏まえ，従来の広く利用されている索引技術を不揮発メモリへ適用する研究が近年活発に行われている．

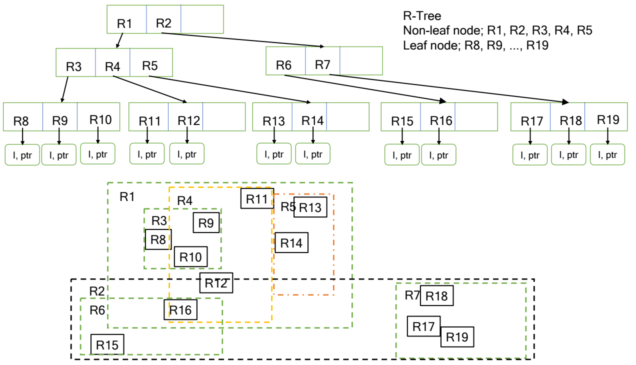

図 4　R-Tree の例

以下に本論文の構成をしめす．本論文では 2 章で本研究で取り上げた空間索引構造を簡単に紹介し，続く 3 章で空間索引構造を利用した実験，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．

2. 空間索引構造

2. 1　R-Tree

多次元データを高速に検索する方法として空間索引構造が知られている．その例としてここでは R-Tree を取り上げる (図 4 参照)． R-Tree [12] は Guttman によって提案された空間索引構造であり，次のような特徴を持つ．

- R-tree の leaf ノードは (I, tuple 識別子) という形式で，I は多次元のオブジェクト，tuple 識別子は当該オブジェクトを一意に識別する．
- leaf ノードでない場合 (non-leaf ノード) は，(I, 子ノードのポインタ) という形式である．
- 子ノードは leaf ノードか non-leaf ノードである．
- 各 leaf ノードはルートノードでなければ m から M 個の索引レコードを持つ．
- leaf ノードの各索引レコードは n 次元の最小矩形である．
- ルートノードは leaf ノードでないかぎり 2 個の子ノードを持つ．
- 全ての leaf は等しいレベルである（leaf ノードまでの高さは等しい）

図 4 では，ルートノードが R1 と R2 の子ノードを持ち，R1 は R3,R4,R5 の子ノードを持つ，R3 は同様に R8,R9,R10 を持ち，それらは leaf ノードになる．leaf ノードは多次元オブジェクトデータと tuple 識別子を持つ．B+tree と同様に，leaf ノードにデータを格納していくと，leaf までの高さを一定に保つために適宜分割などが発生する．

2. 2　不揮発メモリのプログラミングモデル

不揮発メモリをストレージデバイスとして利用するときにはいくつかの注意すべき点がある．従来のプログラミングにおいてはデータの永続化はファイルシステムやデータベースなどを利用して行っているが，不揮発メモリの場合，ファイルやデータベースの利用ではなく，プログラミングの変更が必

表 1　実験環境 [48]

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

要となる．

2.2.1　明示的な情報の永続化

不揮発メモリに情報を格納すると永続化されるが，そのタイミングは一般には非同期であり，通常はキャッシュに書き込まれるだけで，メディアへの書き込みはプログラマは意識しない (図 2 参照)．また書き込む順も不定である．そのために，明示的にキャッシュからフラッシュし，書き込み順を同期する必要がある．アプリケーションによっては書き込みの順番が一貫性を保持するために必要な場合がある．不揮発メモリの場合，内容が永続化する前にシステムがクラッシュして内容が消失した場合，不整合が発生する場合があるので注意が必要である．

2.2.2　アトミックな書き込みは最大 8 バイト

Intel Optane DC Persistent Module (以下 Intel DCPMM と称する) の場合アトミックに書き込めるのは最大 8 バイトである．それ以上に大きな塊での書き込みはアトミックには行われない．したがって途中でクラッシュすると一貫性が保証されない．例えば R-Tree への書き込みはアトミックに行われなければならない．

2.2.3　クラッシュコンシステンシーの実装

繰り返し述べているように不揮発性メモリにおいては単なるメモリへの転送だけでは一貫性を保持することができない．何らかの方法でクラッシュコンシステンシーを確保する必要がある．

B+tree と同様に R-Tree でもページへのレコードの追加はページ分割が発生する場合があり，それは leaf ノードから root ノードまで波及する場合がある．木構造を変更するためにアトミックに変更する必要があり，その一貫性制御が重要である．

先行研究の一つ FBR [9] では mutex lock を利用してクラッシュコンシステンシーを実装した．PMR [21] では不揮発メモリ向け lock free アルゴリズム [37] を利用して評価した．また複数バイトへのアトミックな CAS (compare and swap) として MwCAS [13] を利用した．

3.　実験

3.1　ハードウェア，ソフトウェア構成

実験環境は表 1 に示した．

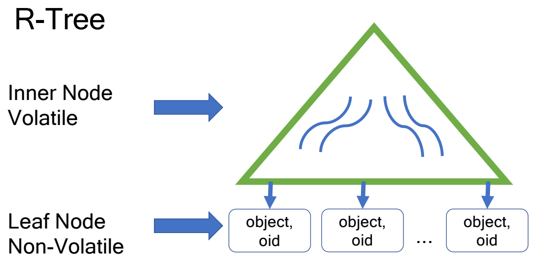

図 5　Inner ノードは PMEM にあるが変更時に flush や同期をとらない．leaf ノードは flush や同期を取る．トランザクションの最後に flush する

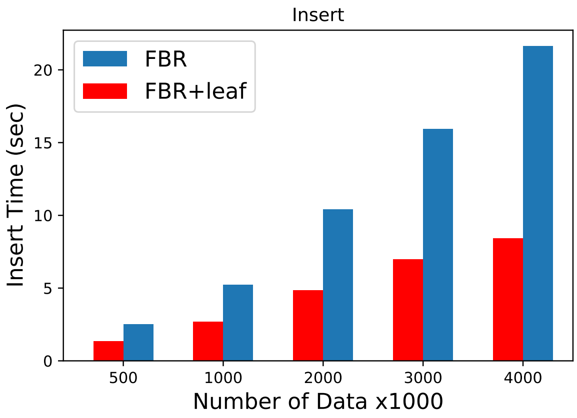

図 6　leaf node 以外は明示的に flush しない，inner node の同期はトランザクション終了時のみに行う

3.2　評価

われわれの先行研究 [43](図 3) によれば，永続性を確保するための明示的な flush および同期命令 (FENCE) のコストは高い．そこで cache flush を削減することを考えた ([49])．R-Tree を inner node と leaf node と分けて考え，明示的な flush と同期は leaf node のみとして実験を行った．inner node の同期はトランザクションの終了時にのみ行うようにした．(図 5)

不揮発メモリ上に R-Tree を作成し，シングルスレッドでランダムに生成したデータを挿入し，Cache flush 削減の効果を測定した．

実験結果は図 6 に示した．縦軸が実行時間，横軸が挿入したデータ数．実行時間が少ないほど（グラフが低いほど）性能が高い．FBR（青）がベースラインで提案手法（赤）が変更した方式である．いずれの場合も leaf node のみを flush したほうが性能が高い．実行時間の比で 23.4 から 25.4% であった．

leaf node 以外は明示的には flush せず，最後に flush した．一件一件 flush していないので，途中でクラッシュするとデータは失われる．

4. 関連研究

Intel Opetane DCPMMそのものの性能評価は[19], [36]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．

既存のデータ構造をNVMに拡張し評価したものとして[22]などがある．B^{+}Tree及びそのNVM拡張系の提案として，Persistent B^{+}-Trees [6],FPTree [29],Bztree [2]，また索引データ構造の比較検討[18], [23]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 [41],Hikv(KVS) [40],DBMSの実装[3],Write-behind logging [4]などがある．

High Performance Computing分野での評価は[38]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．

不揮発メモリ向けの範囲索引について[14], [16], [23]などが様々な提案の比較を行った．ハッシュ索引[15]，ファイルシステム[31]，基本的な動作特性[30], [38]などの報告がある．不揮発メモリ向けの範囲索引について[23]はBzTree [1], FPTree [29], NV-Tree [42], wBTree [5], について評価している．[14]は, [23]で取り上げられなかった以下の範囲索引LB+-Tree [25], uTree [7], DPTree [45], ROART [27], PACTree [20]をそれぞれ評価した．主にIntel Optane DC Persistent Memory Module出荷後に提案されたものを実機で評価している．

同様に[15]はハッシュ索引を比較し，Level hashing [46], Clevel hashing [8], CCEH [28], Dash [26], PCLHT [22], SOFT [47]を評価した．ファイルシステム[31]，データベースエンジン[44]，不揮発メモリ向けのデータ構造[11]，それぞれのカテゴリで比較検討している．Intel Optane DCPMMの性能評価は[30], [38]にある．[24]は不揮発メモリを大規模メインメモリとして利用する様々なアプリケーションについて調査している．

不揮発メモリ向けR-Treeの研究は近年徐々に増えてきている．[33]はFBRのスケーラビリティの欠如の問題について検討し，many coreマシンを対象に，MPR-Treeを提案した．NUMAをサポートし更新のスケーラビリティを向上させた．

SSD(Flash storage)対応のR-treeの研究として[39]と[10]がある．前者はSSD向けの最適化について，後者は不揮発メモリとSSDのハイブリッドな構成での最適化について検討している．空間索引構造についてはまだ十分に検討がされていない．そこで本研究では，不揮発メモリを対象とする空間検索構造の実装方式を検討し予備実験をおこなった．

5. まとめと今後の課題

不揮発メモリを対象とする空間索引構造の実装方式の検討を行った．FBRの実装を確認し，その挿入性能を計測した．不揮発メモリのflushコストが高いことに着目し，inner nodeの明示的なflushは行わずleaf nodeのみflushと同期を取るようにした．ただし，全く同期を取らないとクラッシュしたときに不整合が発生するので，トランザクションの終了時にflushをするようにした．その結果，従来の方法に比べて実行時間で23.4から25.4%に削減された．

今回の実験評価はシングルスレッドで行った．今後はマルチスレッド環境での性能を評価していきたい．

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 *Proc' of the 2017 ACM SIGMOD*, p. 1753 – 1758, 2017.

[4] Joy Arulraj, Matthew Perron, and Andrew Pavlo. Write-behind logging. *Proc' 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. *Proceedings of the VLDB Endowment*, Vol. 8, No. 7, pp. 786–797, 2015.

[6] 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.

[7] Youmin Chen, Youyou Lu, Kedong Fang, Qing Wang, and Jiwu Shu. utree: a persistent b+-tree with low tail latency. *Proceedings of the VLDB Endowment*, Vol. 13, No. 12, pp. 2634–2648, 2020.

[8] Zhangyu Chen, Yu Hua, Bo Ding, and Pengfei Zuo. Lock-free concurrent level hashing for persistent memory. In *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference*, pp. 799–812, 2020.

[9] Soojeong Cho, Wonbae Kim, Sehyeon Oh, Changdae Kim, Kwangwon Koh, and Beomseok Nam. Failure-atomic byte-addressable r-tree for persistent memory. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 32, No. 3, pp. 601–614, 2020.

[10] Athanasios Fevgas, Leonidas Akritidis, Miltiadis Alamaniotis, Panagiota Tsompanopoulou, and Panayiotis Bozanis. A study of r-tree performance in hybrid flash/3dxpoint storage. In *2019 10th International Conference on Information, Intelligence, Systems and Applications (IISA)*, pp. 1–6. IEEE, 2019.

[11] Philipp Götze, Arun Kumar Tharanatha, and Kai-Uwe Sattler. Data structure primitives on persistent memory: an evaluation. In *Proceedings of the 16th International Workshop on Data Management on New Hardware*, pp. 1–3, 2020.

[12] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pp. 47–57, 1984.

[13] Timothy L Harris, Keir Fraser, and Ian A Pratt. A practical multi-word compare-and-swap operation. In *International Symposium on Distributed Computing*, pp. 265–279. Springer, 2002.

[14] Yuliang He, Duo Lu, Kaisong Huang, and Tianzheng Wang. Evaluating persistent memory range indexes: Part two. *arXiv preprint arXiv:2201.13047*, 2022.

[15] Daokun Hu, Zhiwen Chen, Jianbing Wu, Jianhua Sun, and Hao Chen. Persistent memory hash indexes: An experimental evaluation. *Proc. VLDB Endow.*, Vol. 14, No. 5, p. 785 – 798, mar 2021.

[16] Kaisong Huang, Yuliang He, and Tianzheng Wang. The past, present and future of indexing on persistent memory. *Proc. VLDB Endow.*, Vol. 15, No. 12, p. 3774 – 3777, sep 2022.

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

[18] 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 (MSST 2020)*, pp. 2–2, 2020.

[19] 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:1903.05714*, pp. 1–61, 2019.

[20] Wook-Hee Kim, R Madhava Krishnan, Xinwei Fu, Sanidhya Kashyap, and Changwoo Min. Pactree: A high performance persistent range index using pac guidelines. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pp. 424–439, 2021.

[21] Brandon Lavinsky and Xuechen Zhang. Pm-rtree: A highly-efficient crash-consistent r-tree for persistent memory. In *Proceedings of the 34th International Conference on Scientific and Statistical Database Management*, pp. 1–11, 2022.

[22] Se Kwon Lee, Jayashree Mohan, Sanidhya Kashyap, Taesoo Kim, and Vijay Chidambaram. Recipe: converting concurrent DRAM indexes to persistent-memory indexes. In *Proc' of 27th ACM SOSP*, pp. 462–477, 2019.

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

[24] Hai-Kun Liu, Di Chen, Hai Jin, Xiao-Fei Liao, Binsheng He, Kan Hu, and Yu Zhang. A survey of non-volatile main memory technologies: State-of-the-arts, practices, and future directions. *Journal of Computer Science and Technology*, Vol. 36, No. 1, pp. 4–32, 2021.

[25] Jihang Liu, Shimin Chen, and Lujun Wang. Lb+ trees: optimizing persistent index performance on 3dxpoint memory. *Proceedings of the VLDB Endowment*, Vol. 13, No. 7, pp. 1078–1090, 2020.

[26] Baotong Lu, Xiangpeng Hao, Tianzheng Wang, and Eric Lo. Dash: Scalable hashing on persistent memory. *arXiv preprint arXiv:2003.07302*, 2020.

[27] Shaonan Ma, Kang Chen, Shimin Chen, Mengxing Liu, Jianglang Zhu, Hongbo Kang, and Yongwei Wu. Roart: Range-query optimized persistent art. In *FAST*, pp. 1–16, 2021.

[28] Moohyeon Nam, Hokeun Cha, Young-ri Choi, Sam H Noh, and Beomseok Nam. Write-optimized dynamic hashing for persistent memory. In *FAST*, Vol. 19, pp. 31–44, 2019.

[29] 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 *Proc' of the 2016 SIGMOD*, pp. 371–386, 2016.

[30] 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.

[31] Gianlucca O. Puglia, Avelino Francisco Zorzo, César A. F. De Rose, Taciano Perez, and Dejan Milojicic. Non-volatile memory file systems: A survey. *IEEE Access*, Vol. 7, pp. 25836–25871, 2019.

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

[33] Abdul Salam, Safdar Jamil, Sungwon Jung, Sung-Soon Park, and Youngjae Kim. Future-based persistent spatial data structure for nvm-based manycore machines. *IEEE Access*, Vol. 10, pp. 114711–114724, 2022.

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

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

[36] Alexander van Renen, Lukas Vogel, Viktor Leis, Thomas Neumann, and Alfons Kemper. Persistent memory I/O primitives. In *Proc' of DaMoN'19*, pp. 1–7. ACM, 2019.

[37] Tianzheng Wang, Justin Levandoski, and Per-Ake Larson. Easy lock-free indexing in non-volatile memory. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pp. 461–472. IEEE, 2018.

[38] 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 *Proc' of SC '19*, pp. 1–19, 2019.

[39] Chin-Hsien Wu, Li-Pin Chang, and Tei-Wei Kuo. An efficient r-tree implementation over flash-memory storage systems. In *Proceedings of the 11th ACM international symposium on Advances in geographic information systems*, pp. 17–24, 2003.

[40] Fei Xia, Dejun Jiang, Jin Xiong, and Ninghui Sun. Hikv: A hybrid index key-value store for dram-nvm memory systems. In *USENIX/ATC 17*, pp. 349–362, 2017.

[41] Jian Xu and Steven Swanson. NOVA: A log-structured file system for hybrid volatile/non-volatile main memories. In *14th USENIX/FAST*, pp. 323–338, 2016.

[42] Jun Yang, Qingsong Wei, Chundong Wang, Cheng Chen, Khai Leong Yong, and Bingsheng He. Nv-tree: A consistent and workload-adaptive tree structure for non-volatile memory. *IEEE Transactions on Computers*, Vol. 65, No. 7, pp. 2169–2183, 2016.

[43] Hirotaka Yoshioka, Yuto Hayamizu, Kazuo Goda, and Masaru Kitsuregawa. pmmeter: A microbenchmark for understanding synchronization cost on persistent memory. In *2023 IEEE International Conference on Big Data and Smart Computing (BigComp2023)*, pp. 326–327. IEEE, 2023.

[44] Hao Zhang, Gang Chen, Beng Chin Ooi, Kian-Lee Tan, and Meihui Zhang. In-memory big data management and processing: A survey. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 27, No. 7, pp. 1920–1948, 2015.

[45] Xinjing Zhou, Lidan Shou, Ke Chen, Wei Hu, and Gang Chen. Dptree: differential indexing for persistent memory. *Proceedings of the VLDB Endowment*, Vol. 13, No. 4, pp. 421–434, 2019.

[46] Pengfei Zuo, Yu Hua, and Jie Wu. Write-optimized and high-performance hashing index scheme for persistent memory. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pp. 461–476, 2018.

[47] Yoav Zuriel, Michal Friedman, Gali Sheffi, Nachshon Cohen, and Erez Petrank. Efficient lock-free durable sets. *Proceedings of the ACM on Programming Languages*, Vol. 3, No. OOPSLA, pp. 1–26, 2019.

[48] 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.

[49] 吉岡弘隆, 早水悠登, 合田和生, 喜連川優. 不揮発メモリ性能を対象とする空間索引構造の実装方式の検討と予備実験. *DEIM2023*, pp. 2B–5–4, 2023.