The Transactions of the Korea Information Processing Society
TKIPS Vol.13, No.11, pp.579~584
eISSN: 3022-7011
https://doi.org/10.3745/TKIPS.2024.13.11.579

Recent Trends in Leveraging Processing-in-Memory for Accelerating Relational Database Management Systems

Taehun Kim† · Gyutae Kim† · Youngsok Kim†† · Joonsung Kim†††

ABSTRACT

In this paper, we present a comprehensive survey on recent research exploiting Processing-In-Memory (PIM) to enhance the performance of Relational Database Management Systems (RDBMSs). PIM integrates In-Memory Processors (IMPs) near memory banks, allowing memory-intensive operations to leverage the high internal memory bandwidth and computational capabilities of IMPs. Given that many key operations of RDBMSs are memory-heavy and require a large number of frequent memory accesses, PIM has emerged as a promising solution for accelerating RDBMSs. We provide an overview of the key architectural characteristics of commodity PIM-enabled memory devices, examine how recent studies have applied PIM to accelerate RDBMSs, and explore potential research directions in PIM-accelerated RDBMSs.

Keywords : Processing-In-Memory, Relational Database Management Systems, Hardware Acceleration

프로세싱-인-메모리를 이용한 관계형 데이터베이스 관리 시스템 가속의 최신 연구 동향

김 태 훈† · 김 규 태† · 김 영 석†† · 김 준 성†††

요 약

이 논문에서는 최근 연구 출판물이 PIM(Processing-In-Memory)을 활용하여 관계형 데이터베이스 관리 시스템(RDBMS)을 가속화하는 방법에 대한 조사를 수행한다. PIM은 메모리 뱅크 근처에 IMP(In-Memory Processor)를 구현하여 메모리 집약적 작업이 IMP의 높은 집계 내부 메모리 대역폭과 계산 처리량을 활용할 수 있도록 한다. RDBMS의 많은 핵심 작업이 메모리 집약적이고 많은 수의 메모리 액세스를 초래하는 것으로 알려져 있기 때문에, PIM은 RDBMS를 가속화하는 데 매력적인 선택이 되었다. 본 논문은 상용 PIM 지원 메모리 장치의 주요 구조적 특성과 최근 연구가 PIM을 활용하여 RDBMS를 가속화하는 방법을 정리하고, 더 나아가 PIM 가속화 RDBMS의 잠재적 연구 방향을 제시한다.

키워드 : 프로세싱-인-메모리, 관계형 데이터베이스 관리 시스템, 하드웨어 가속

1. Introduction

Relational Database Management Systems (RDBMSs) allow users to extract meaningful information and findings from large amounts of data. On the RDBMS, users can issue analytic queries on a set of tables. Queries are typically written in Structured Query Language (SQL) [1]. The users can construct various SQL queries using multiple relational operations (e.g., *SCAN*, *JOIN*) for their own purposes. In RDBMS, each table consists of rows, and each row has multiple columns. For a given query, the RDBMS performs the relational operations on the SQL query's input tables and produces the output for the SQL query.

Aimed at avoiding the excessive data access latency of storage devices, some recent RDBMSs have adopted an in-memory design. In-memory RDBMSs store their tables entirely in memory, rather than on traditional storage devices, to exploit the low access latency provided by Dynamic Random Access Memory (DRAM) devices [2, 3, 4]. As

※ 이 논문은 2024년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임(RS-2024-00395134, 차세대 AI 반도체를 위한 DPU 중심의 데이터센터 아키텍처)
※ 이 논문은 한국연구재단의 지원을 받아 수행된 연구임(RS-2022-NR071917, 시공간적 멀티태스킹 NPU 아키텍처 및 시스템 소프트웨어).
† 비 회 원 : 성균관대학교 전자전기공학부 학사과정
†† 비 회 원 : 연세대학교 컴퓨터과학과 부교수
††† 정 회 원 : 성균관대학교 반도체시스템공학과 조교수
Manuscript Received : October 8, 2024
Accepted : November 14, 2024
* Corresponding Author : Joonsung Kim(joonsungkim@skku.edu)

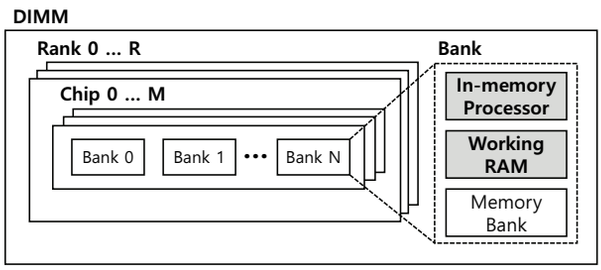

Fig. 1. The High-level Overview of DIMM-based PIM Architecture

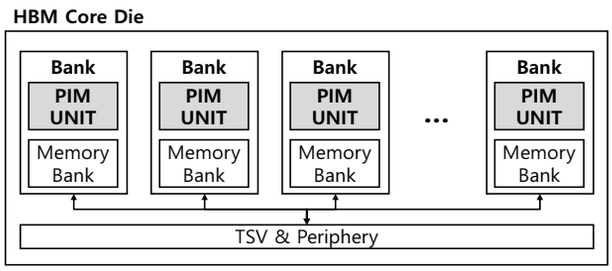

Fig. 2. The High-level Overview of HBM-based PIM Architecture

all tables are stored in memory, executing SQL query's relational operations on in-memory RDBMSs often involves frequent memory accesses [5, 6].

One promising solution to mitigate the frequent memory accesses of in-memory RDBMSs is Processing-In-Memory (PIM). PIM places In-Memory Processors (IMPs) near memory banks. By doing so, applications can offload their memory-intensive operations to IMPs, allowing for fully exploiting the large aggregate internal memory bandwidth and computational capabilities provided by IMPs. Recently, PIM has been shown to successfully accelerate key memory-intensive operations of various application domains (e.g., machine learning [7-9], bioinformatics [10-12]).

Recognizing the significant potential of PIM for accelerating in-memory RDBMSs, this paper presents a comprehensive survey of recent research trends using PIM for in-memory RDBMSs. Based on the survey, this paper then outlines some potential research directions on RDBMS acceleration using PIM. In particular, the paper focuses on recent research publications which accelerate relational operations using commodity PIM-enabled memory devices. Although there are some studies proposing completely new PIM-enabled memory architectures to accelerate RDBMSs, their practical applicability remains uncertain until they are implemented in real-world systems. Therefore, we mostly ***focus on the RDBMS acceleration using commodity PIM-enabled memory devices*** (e.g., UPMEM).

2. Commodity Processing-In-Memory Devices

Processing-In-Memory (PIM) can be implemented in many forms, depending on the underlying memory architecture (e.g., DDR, LPDDR, GDDR, HBM). Among various implementations of PIM, two types have been shown practical through the availability of real-world PIM-enabled memory devices: PIM on Dual In-line Memory Module (DIMM) [13, 16], and PIM on High-Bandwidth Memory (HBM) [14, 15, 17, 18]. In this section, we discuss representative commodity implementations of PIM on DIMM and PIM on HBM.

2.1 PIM on Dual In-line Memory Module

PIM on DIMM uses DIMMs, currently the de-facto standard for implementing host memory, as the underlying memory architecture. PIM-enabled DIMMs typically associate an IMP to each memory bank, making the number of IMPs equal to the number of memory banks.

Fig. 1. shows one of representative implementations of PIM-enabled DIMM (i.e., UPMEM DIMMs) [16]. An UPMEM DIMM extends a DDR4-2400 DIMM by adding a scalar, multi-threaded IMP per each 64-MB memory bank. The IMP cannot directly access the data stored in its associated memory bank; however, it can access the data stored in a 64-KB Working RAM (i.e., *WRAM*) which acts as an intermediate buffer for the data stored in the memory bank. Each UPMEM DIMM contains 128 350-MHz DPUs each of which provides an internal memory bandwidth of 700 MB/s. This allows an UPMEM DIMM to achieve an aggregate internal memory bandwidth of 87.5 GB/s and computational throughput of 43.75 GOPS in total.

2.2 PIM on High-Bandwidth Memory

Similar to how PIM can be implemented by extending DIMMs with IMPs, High-Bandwidth Memory (HBM) devices can also be used as underlying memory for PIM. Unlike DIMMs, HBM devices employ stacked memory architectures comprising a logic die and multiple memory dies stacked vertically.

Samsung's Aquabolt-XL [17, 18] is a representative implementation of PIM on HBM devices. Aquabolt-XL extends an HBM2 device by partitioning memory banks into eight chips: four PIM-enabled ones and four standard

ones. Fig. 2 shows the high-level overview of HBM-based PIM's core die architecture. The PIM-enabled chips associate an IMP to each pair of their memory banks, allowing the IMP to load and perform arithmetic operations (i.e., addition, multiplication) on the data stored in the pair of memory banks. Aquabolt-XL's 128 IMPs (4 PIM-enabled chips, 32 IMPs per chip) collectively provide an internal memory bandwidth of 4.92 TB/s in total.

3. Accelerating RDBMSs Using PIM

To take advantage of PIM's high memory bandwidth for accelerating in-memory RDBMSs, recent studies have proposed using PIM for RDBMSs to implement relational operations. Among possible PIM implementations, this paper focuses on accelerating relational operations on UPMEM DIMMs, currently the only publicly-available PIM-enabled DIMMs. While PIM-enabled HBM devices such as Aquabolt-XL is promising, those devices are not publicly available yet; thus the research community has been mostly focusing on PIM-enabled DIMMs with UPMEM DIMMs.

As representative relational operations, we select SCAN and JOIN. When a predicate and table are given, the SCAN operation examines the predicate on each tuple of row/column in the table and returns the tuple(s) only satisfying the predicate. For example, executing "*SELECT * FROM R WHERE R.key = 10*" SQL query returns the rows of table R whose key column's value is 10. The JOIN operation, on the other hand, takes as input a predicate and two input tables, and returns a pair of the tables' tuples which satisfy the predicate. For instance, a SQL query "*SELECT R.key, S.key FROM R, S WHERE R.key = S.key*" returns the pairs of R's key column and table S's key column whose key values are equal.

The SCAN and JOIN operations are widely used in various data analytic queries, and are known to be memory-intensive. The SCAN operations need to sweep the values of an input table's entire rows and columns, and the JOIN operations incur frequent and redundant accesses to their input tables' tuples.

3.1 Accelerating SCANs Using PIM-Enabled DIMMs

Accelerating the SCAN operations using PIM-enabled DIMMs have been extensively studied, thanks to the fact that SCANs map well to the architectural characteristics of PIM-enabled DIMMs. The SCAN operations can be easily parallelized to multiple IMPs as the evaluation of a predicate on one tuple does not have any data dependencies with examining the predicate on the other tuples. In other words, the SCAN operations exhibit a tuple-level parallelism, and thus the evaluation of a predicate on different tuples can be partitioned and distributed to different IMPs easily.

Baumstark et al. [19, 20] leverage this characteristic and make a prototype to evaluate the performance of UPMEM DIMMs for the SCAN operations. For a given table, they evenly partition and distribute the tuples of the table to all available IMPs. Then, they minimize the data transfer overhead between the host memory and UPMEM DIMMs by parallelizing and overlapping the transfer of different data chunks per each IMP. After that, on each IMP, they spawn multiple tasklets which evaluate a predicate on its corresponding tuples. Once evaluating the predicate is finished, they store the results into a single bit-vector whose i-th value indicating whether the i-th tuple satisfies the predicate or not; the value is set to 1 if the associated tuple satisfies the predicate, and 0 if it does not match the predicate. For the evaluation, they use the Social Network Benchmark (SNB) and four UPMEM DIMMs to show that the SCAN operations can be greatly accelerated by using UPMEM (i.e., commodity PIM-enabled DIMMs) compared to the baseline CPU system equipped with standard DIMMs.

3.2 Accelerating JOINs Using PIM-Enabled DIMMs

The JOIN operations are considered as one of the key relational operations in RDBMSs, since they can combine multiple tables to produce a new table [1]. Since the JOIN operations need to access the entire input tables, they are also memory-intensive similar to the SCAN operations. Due to their importance and memory-intensive nature, there have been lots of studies aiming to accelerate the JOIN operations using PIM-enabled DIMMs. Compared to the SCAN operations, the JOIN operations require more complex memory access and computation behaviors. For example, the SCAN operations simply read a tuple and evaluate a predicate on the tuple; however, the JOIN operations necessitate evaluating the predicate on all possible pairs of tuples from two input tables. This makes it difficult to map the JOIN operations to the memory banks and IMPs in a straightforward way.

There have been several studies to fully leverage the PIM-enabled DIMMs for the JOIN operations. Shlomo et al. [21] has released an implementation of an in-memory

join algorithm optimized for the UPMEM DIMMs. The proposed join algorithm evenly partitions the tuples of two input tables to all IMPs of UPMEM DIMMs, invokes single-IMP hash-join algorithm on each IMP, and then makes the host CPUs collect the join results from the IMPs and merge them. Despite achieving speedups over the CPU baseline, the proposed join algorithm suffers from a limited applicability; the algorithm has several constraints and only works with the same-sized input tables, which is often unrealistic in the real-world use cases.

Meanwhile, Lim et al. [22] has proposed PID-Join, a novel in-memory join algorithm designed and optimized for the UPMEM DIMMs. Similar to [21], PID-Join also evenly partitions input tables to all the IMPs; however, PID-Join achieves higher applicability than [21] by assuming that all the input tuples initially reside in the host memory, not the UPMEM DIMMs, and supporting arbitrary capacity ratios between two input tables. To support different input table capacity ratios, PID-Join proposes a fast communication mechanism, called Rotate-and-Stream, between the host memory and the UPMEM DIMMs. In addition, PID-Join supports not only hash joins, but also nested-loop and sort-merge joins. Supporting non-hash joins allows PID-Join to support a much wider join predicates than the in-memory join algorithm proposed by Shlomo et al. [21], which significantly increases their applicability.

4. Potential Future Research Directions

Based on our extensive survey of the recent research trends on leveraging PIM for accelerating the in-memory RDBMSs, we identify the following research directions with significant potentials for performance improvement and broader impact.

4.1 Accelerating Other Relational Operations

Although the SCAN and JOIN operations have been extensively analyzed and investigated because of their importance in the RDBMSs, many other relational operations still have a substantial impact on the SQL query execution speeds. For example, aggregations, which typically appear later in SQL queries, are feasible acceleration targets for PIM as their input sizes are large but their outputs are typically a single scalar value. Since the PIM has high internal memory bandwidth, this kind of large-input-small-output operation can significantly benefit from the PIM-enabled DIMMs.

Similar to the SCAN and JOIN operations, it would be worth to consider a GROUP-BY operations as an acceleration target of PIM as well. The GROUP-BY operation partitions the input tuples into multiple groups based on given key values, which inevitably incurs several reads/writes to the same tuple multiple times. Different from the SCAN and JOIN operations, the GROUP-BY operations have their unique challenges when it comes to PIM-assisted acceleration as they can be implemented in multiple ways.

For example, some can implement the GROUP-BY operations by sorting the key values first then partitioning the sorted tuples into multiple groups to increase the partitioning efficiency. In contrast, others often employ hashing before partitioning instead of using the sorting-based approach. Note that these two different versions of implementation have their own trade-off in terms of performance characteristics, memory usage, and flexibility. For example, sorting is generally more computationally expensive than hashing (i.e., sorting O(NlogN) vs. hashing O(N)), which makes the sorting-based implementation relatively not suitable for larger datasets. For the memory usage, hashing requires additional memory to store the hash table, which linearly grows as the number of groups increases. Therefore, for the system having relatively lower memory capacity, the hashing-based implementation might not be a good option than the sorting-based one. These complicated trade-offs between sorting- and hashing-based implementations for the GROUP-BY operations would present many more technical challenges than relatively simpler relational operations such as SCAN and JOIN operations.

4.2 Achieving Skew-tolerant PIM Acceleration

While in-memory JOIN operations have been accelerated with the PIM-enabled DIMMs by the efforts from Shlomo et al. [21] and Lim et al. [22], one key limitation is that their proposals do not consider high data skew scenarios. When input tables are highly skewed, an even distribution of input tuples to IMPs become infeasible as a few popular tuple values appear more frequently than the other less popular tuple values within a table. This can result in a significant load imbalance across the IMPs, causing many of the IMPs to wait for the longest-running and heaviest-load IMP for a given in-memory join scenario. Therefore, the proposed in-memory join algo-

rithms need to be reconsidered to deal with the highly skewed input tables. For example, some can propose a new skew-resistant in-memory join algorithm for the PIM-enabled DIMMs.

4.3 Finding Other Applications for PIM in RDBMSs

In addition to accelerating the relational operations in the RDBMSs, it is worth to focus on accelerating the internal operations (e.g., compaction) in the DBMS itself. In modern database management systems, they often adopt the log-structured merge-tree (LSM tree) for their internal data structure because of its scalability and efficiency. For instance, the LSM tree stores incoming new write requests (e.g., INSERT) sequentially, which can significantly improve the DBMS's storage efficiency. To do so, they need to perform the periodic compaction operations to convert multiple small sub trees into a single merged tree. Here, these compaction operations cause huge compute/memory overheads, which is one of the major performance challenges in the modern DBMSs.

Fortunately, the compaction operations would be good acceleration targets using PIM. The compaction operations consist of sequential read and write operations which require a huge amount of memory bandwidth, but the output size is quite smaller than the size of inputs. Therefore, we can leverage the huge internal bandwidth of the PIM-enabled DIMM for the compaction operations.

5. Conclusion

In this paper, we presented a comprehensive survey of some recent research efforts on leveraging PIM for accelerating RDBMSs. Our survey reveals that, although the existing proposals achieve notable speedups over conventional CPU-based systems, their applicability remains limited and needs further improvement. Based on the observed limitation, we then presented a few potential future research directions towards PIM-assisted RDBMS acceleration. We hope our survey and potential future research directions help researchers gain good understandings on the recent trends and the research challenges, and provide some research insights.

References

[1] C. J. Date, "A guide to the SQL standard," Addison-Wesley Longman Publishing Co., Inc., 1989.

[2] A. Boroumand, S. Ghose, G. F. Oliveira, and O. Mutlu, "Polynesia: Enabling high-performance and energy-efficient hybrid transactional/analytical databases with hardware/software co-design." In *IEEE 38th International Conference on Data Engineering (ICDE)*, 2022.

[3] S. I. F. G. N. Nes and S. M. S. M. M. Kersten, "MonetDB: Two decades of research in column-oriented database architectures," *Data Engineering*, 40 (2012).

[4] S. R. dos Santos, F. B. Moreira, T. R. Kepe, and M. A. Alves. "Advancing Database System Operators with Near-Data Processing," In *2022 30th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, 2022.

[5] S. K. Begley, Z. He, and Y. P. P. Chen, "Mcjoin: A memory-constrained join for column-store main-memory databases," In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2012.

[6] H. Gao and N. Sakharnykh, "Scaling Joins to a Thousand GPUs," In ADMS@ VLDB, 2021.

[7] P. Chi et al., "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," *ACM SIGARCH Computer Architecture News 44*, 2016.

[8] A. Shafiee, et al., "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, Vol.44, No.3, pp.14-26, 2016.

[9] M. Imani, S. Gupta, Y. Kim, and T. Rosing, "Floatpim: In-memory acceleration of deep neural network training with high precision," In *Proceedings of the 46th International Symposium on Computer Architecture*, 2019.

[10] F. Zhang, S. Angizi, N. A. Fahmi, W. Zhang, and D. Fan. "Pim-quantifier: A processing-in-memory platform for mrna quantification," In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021.

[11] S. Angizi, N. A. Fahmi, W. Zhang, and D. Fan, "Pim-assembler: A processing-in-memory platform for genome assembly," In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020.

[12] S. Angizi, J. Sun, W. Zhang, and D. Fan, "PIM-Aligner: A processing-in-MRAM platform for biological sequence alignment," In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020.

[13] L. Ke et al., "Near-memory processing in action: Accelerating personalized recommendation with axdimm," IEEE Micro 42, 2021.

[14] M. He et al., "Newton: A DRAM-maker's accelerator-in-memory (AiM) architecture for machine learning," In *53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020.
[15] S. Lee et al., "Hardware architecture and software stack for PIM based on commercial DRAM technology: Industrial product," In *ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021.
[16] F. Devaux, "The true processing in memory accelerator," In *IEEE Hot Chips 31 Symposium (HCS)*, 2019.
[17] J. H. Kim et al., "Aquabolt-XL: Samsung HBM2-PIM with in-memory processing for ML accelerators and beyond," In *IEEE Hot Chips 33 Symposium (HCS)*, 2021.
[18] J. H. Kim et al., "Aquabolt-XL HBM2-PIM, LPDDR5-PIM with in-memory processing, and AXDIMM with acceleration buffer," IEEE Micro 42, 2022.
[19] B. Alexander M. A. Jibril, and K.-U. Sattler, "Accelerating large table scans using processing-in-memory technology," in *Proceedings of the 20th Conference on Database Systems for Business, Technology and Web (BTW)*, 2023.
[20] B. Alexander, M. A. Jibril, and K.-U. Sattler, "Processing-in-memory for databases: Query processing and data transfer," in *Proceedings of the 19th International Workshop on Data Management on New Hardware (DaMoN)*, 2023.
[21] S. Roee, J. Legriel, A. Moisson, and S. Brocard, UPMEM-PIM Evaluation for SQL Query Acceleration [Internet], https://github.com/upmem/dpu_olap.
[22] C. Lim et al., "Design and Analysis of a Processing-in-DIMM Join Algorithm: A Case Study with UPMEM DIMMs," in *Proceedings of the 2023 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2023.

Taehun Kim
https://orcid.org/0009-0009-6094-6379
e-mail : kth3971@skku.edu
He is an undergraduate student in the Department of Electrical and Electrical Engineering at Sungkyunkwan University. His research interests are in the area of computer architecture and network system.

Gyutae Kim
https://orcid.org/0009-0003-0363-2839
e-mail : kktt9909@skku.edu
He is an undergraduate student in the Department of Electrical and Electrical Engineering at Sungkyunkwan University. His research interests are in the area of big data processing and database system.

Youngsok Kim
https://orcid.org/0000-0002-1015-9969
e-mail : youngsok@yonsei.ac.kr
He is an associate professor in the Department of Computer Science and Engineering at Yonsei University. He received PhD and BSc in Computer Science and Engineering from POSTECH. His research interests are in computer architecture and system software research.

Joonsung Kim
https://orcid.org/0000-0002-5432-7813
e-mail : joonsungkim@skku.edu
He is an assistant professor in the Department of Semiconductor Systems Engineering at Sungkyunkwan University. He received his Ph.D. in Electrical and Computer Engineering at Seoul National University and his M.S. and B.S. in Computer Science and Engineering from POSTECH. His research interests are in system performance modeling and datacenter-scale system architecture.