PUSHtap: PIM-based In-Memory HTAP with Unified Data Storage Format

Yilong Zhao
Shanghai Jiao Tong University
Shanghai, China
Shanghai Qi Zhi Institute
Shanghai, China
sjtuzyl@sjtu.edu.cn

Mingyu Gao
Tsinghua University
Beijing, China
Shanghai Qi Zhi Institute
Shanghai, China
gaomy@tsinghua.edu.cn

Huanchen Zhang
Tsinghua University
Beijing, China
Shanghai Qi Zhi Institute
Shanghai, China
huanchen@tsinghua.edu.cn

Fangxin Liu*
Shanghai Jiao Tong University
Shanghai, China
Shanghai Qi Zhi Institute
Shanghai, China
liufangxin@sjtu.edu.cn

Gongye Chen
Shanghai Jiao Tong University
Shanghai, China
gongye_chen@sjtu.edu.cn

He Xian
Shanghai Qi Zhi Institute
Shanghai, China
51265900021@stu.ecnu.edu.cn

Haibing Guan
Shanghai Jiao Tong University
Shanghai, China
hbguan@sjtu.edu.cn

Li Jiang*
Shanghai Jiao Tong University
Shanghai, China
Shanghai Qi Zhi Institute
Shanghai, China
jiangli@cs.sjtu.edu.cn

Abstract

Hybrid transaction/analytical processing (HTAP) is an emerging database paradigm that supports both online transaction processing (OLTP) and online analytical processing (OLAP) workloads. Computing-intensive OLTP operations, involving row-wise data manipulation, are suitable for row-store format. In contrast, memory-intensive OLAP operations, which are column-centric, benefit from column-store format. This *data-format dilemma* prevents HTAP systems from concurrently achieving three design goals: performance isolation, data freshness, and workload-specific optimization. Another background technology is Processing-in-Memory (PIM), which integrates computing units (PIM units) inside DRAM memory devices to accelerate memory-intensive workloads, including OLAP.

Our key insight is to combine the interleaved CPU access and localized PIM unit access to provide two-dimensional access to address the data format contradictions inherent in HTAP. First, we propose a unified data storage format with novel data alignment and placement techniques to optimize the effective bandwidth of CPUs and PIM units and exploit the PIM's parallelism. Second, we implement the multi-version concurrency control (MVCC) essential for single-instance HTAP. Third, we extend the commercial PIM architecture to support the OLAP operations and concurrent access from PIM and CPU. Experiments show that PUSHtap can achieve 3.4×/4.4× OLAP/OLTP throughput improvement compared to multi-instance PIM-based design.

CCS Concepts: • **Computer systems organization** → **Heterogeneous (hybrid) systems**; • **Information systems** → **Database design and models**.

Keywords: Processing-in-Memory (PIM), Hybrid Transactional/Analytical Processing (HTAP), DRAM, Unified Data Format

ACM Reference Format:
Yilong Zhao, Mingyu Gao, Huanchen Zhang, Fangxin Liu, Gongye Chen, He Xian, Haibing Guan, and Li Jiang. 2025. PUSHtap: PIM-based In-Memory HTAP with Unified Data Storage Format. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems,*

*This work was partially supported by the National Key Research and Development Program of China (2024YFE0204300), National Natural Science Foundation of China (Grant No.62402311), and Natural Science Foundation of Shanghai (Grant No.24ZR1433700). Fangxin Liu and Li Jiang are the corresponding authors.

ASPLOS '25, Rotterdam, Netherlands
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-1080-3/2025/03
https://doi.org/10.1145/3676642.3736120

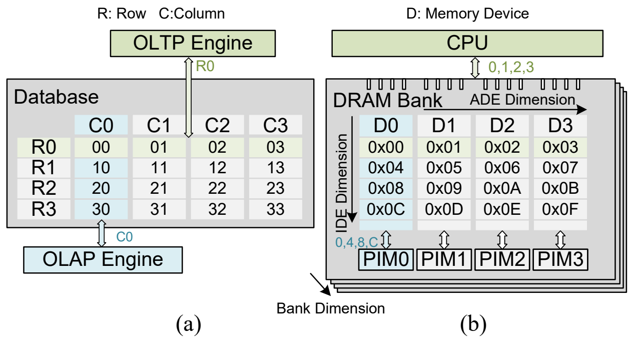


Figure 1. (a) HTAP system: OLTP and OLAP engines process rows and columns, respectively. (b) DRAM-based PIM architecture: CPU interleaves data across memory devices (ADE), and PIM units access data inside the device (IDE).

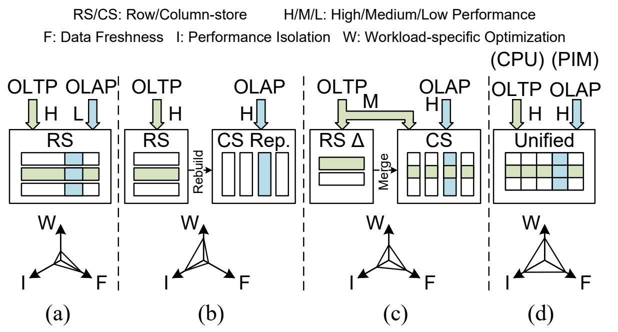

RS/CS: Row/Column-store H/M/L: High/Medium/Low Performance
F: Data Freshness I: Performance Isolation W: Workload-specific Optimization
OLTP OLAP
H L
RS
W I F
(a)
OLTP OLAP
H H
RS
Rebuild
CS Rep.
(b)
OLTP OLAP
M H
RS Δ
Merge
CS
(c)
(CPU) (PIM)
OLTP OLAP
H H
Unified
(d)

Figure 2. Different HTAP data format design. (a) Single-instance design with single data format. (b) Multiple-instance design with different data formats. (c) Single-instance design with mixed data format. (d) PUSHtap with single-instance and unified data format.

Volume 3 (ASPLOS '25), March 30-April 3, 2025, Rotterdam, Netherlands. ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3676642.3736120

1 Introduction

Hybrid transaction/analytical processing (HTAP) is an emerging processing architecture that allows one database system to support two processing sets: online transaction processing (OLTP) and online analytical processing (OLAP) [48]. There has been extensive research and design for HTAP [6, 20, 27, 53, 54, 57]. As shown in Figure 1 (a), the OLTP engine processes transactions, which are single-record operations on rows, including read, insert, update, and delete. On the contrary, the OLAP engine processes analytical queries to solve multidimensional analysis problems on columns. OLTP includes read and write operations, whereas all OLAP operations are read operations. An ideal HTAP system should meet the following three design goals [6, 37, 41]. (1) *Workload-specific optimizations*: optimized performance for both OLTP and OLAP workloads. (2) *Performance isolation*: limited performance degradation for concurrent execution of transactions and analytical queries. (3) *Data refreshes*: analytical queries need up-to-date data. However, current HTAP systems cannot fulfill three goals simultaneously due to the diverse data storage formats required by OLTP and OLAP, as analyzed below.

Figure 2 (a,b,c) summarizes three predominantly adopted data formats in current HTAP systems. (1) *Single-instance with single data format*, either column or row store (Figure 2 (a)) [20, 57]. This design can achieve high data freshness because the OLAP engine is always visible to the latest data that is updated by the OLTP engine. However, a single data format results in sub-optimal workload-specific performance. For example, performing analytical queries on row-store has 50% performance degradation when over 95% of the accessed data are unused; while transactions on column-store have more than 20% degradation [52]. (2) *Multi-instance design* [6, 54]. As shown in Figure 2 (b), both transactions and analytical queries process on their preferred data formats, and thereby can obtain optimal workload-specific performance and high-performance isolation. However, the HTAP system must rebuild the column-store replication from the latest row-store data to ensure data freshness, which requires a significant latency. The rebuilding cannot be executed frequently, leading to low data freshness [37]. (3) *Single-instance design with mixed data format* [27, 53]. This design is composed of a primary column-store data and a row-store delta, as shown in Figure 2 (c). The row-store delta maintains the newest version of rows updated by transactions. OLTP and OLAP engines must scan the two format regions to acquire the complete data. For example, the OLAP engine first scans the column-store data with high bandwidth, then acquires the newest data version by scanning the row-store delta with low bandwidth, and finally merges the data. This approach ensures the freshness of the data but compromises the performance isolation of OLAP and OLTP.

Processing-in-memory (PIM) integrates computing units, i.e., PIM units, inside memory devices to accelerate memory-intensive workloads [11, 25, 26, 33, 34]. PIM units directly access data on their own devices, bypassing the prolonged and limited memory bus to utilize the internal memory bandwidth better. For example, a commercial PIM architecture can achieve over 3.3× bandwidth improvement and 10× access energy reduction [11]. One key limitation of PIM is its localized access, preventing efficient access to remote data residing on other devices. For example, in Figure 1 (b), PIM 0 can directly access data `0x00,04,08,0C` in the device `D0` within several nanoseconds. While for those in device `D1,D2,D3`, they need CPU to move the data to their local device, costing around 0.2μs due to the mode-switch overhead [11].

Memory interleaving is ubiquitously employed to improve CPU's memory bandwidth. Figure 1 (b) presents a memory space interleaved to four devices. Contiguous data blocks, `0x00-04`, are split and mapped to the same location of devices, respectively. CPU can access four blocks in parallel within a single memory access, thereby maximizing the memory bandwidth.

This work advocates combining PIM's localized access and CPU's interleaved access to provide a new perspective for two-dimensional access to every memory bank: the CPU accesses data *across devices* (ADE) in parallel; massive PIM units simultaneously access local data *inside devices* (IDE) with low latency. We can map the HTAP row to the ADE dimension and the column to the IDE dimension. Accordingly, the CPU works as an OLTP engine, and PIM units work as an OLAP engine. Other memory hierarchies, e.g., bank (can scale to rank and channel), form the third access dimension for database scalability.

In this work, we propose PUSHtap, a PIM-based single-instance in-memory HTAP with the unified data storage format as outlined earlier, as shown in Figure 2 (d). OLTP and OLAP engines can achieve optimal performance and data freshness through two-dimensional access on the instance. The contribution of this paper is as follows:

- **Unified HTAP Architecture:** We propose PUSHtap, a PIM-based single-instance HTAP architecture with a unified data storage format. The key insight behind PUSHtap is the combination of interleaved CPU access and localized PIM unit access to address the data format contradictions inherent in HTAP. We introduce a data layout algorithm specifically designed for PUSHtap to enhance effective bandwidth across databases with varying column widths.
- **Concurrency Control Operations:** We design the multi-version concurrency control (MVCC) and the corresponding snapshot operations for PUSHtap. These features are crucial for a single-instance database environment to minimize data transfer between CPU and PIM units, thus optimizing overall system efficiency.
- **Architecture Support:** We extend the memory controller of DRAM-based general-purpose PIM (UPMEM-like) [11], with a hardware interface to support the OLAP operations and the concurrent access by CPU and PIM units, which is crucial to PUSHtap.

2 Background

2.1 DRAM-based PIM

DRAM-based PIM architectures are proposed to accelerate memory-intensive applications [11, 26, 34]. [11] is a representative commercial general-purpose PIM architecture. A PIM unit, along with two scratchpad SRAMs, WRAM (buffer operands), and IRAM (buffer instructions), is integrated into each memory bank. A PIM interface is added to each rank to facilitate CPU control over PIM units. Memory-intensive operations are offloaded to PIM units to better utilize the internal bandwidth of DRAM.

A general-purpose PIM is designed with two modes: CPU mode and PIM mode [11]. In CPU mode, CPU accesses DRAM banks as conventional memory. In PIM mode, PIM units fully control data flow and computation, and DRAM banks are locked to prevent CPU access. When switching from CPU mode to PIM mode, CPU needs to send messages to all the PIM units to hand over the bank access control and invoke the PIM units. After that, CPU polls the PIM units until all the PIM units are finished. As there are thousands of PIM units in a server (4 channels, 8 DDR4 DIMMs [11]), it takes **tens of microseconds** to invoke and poll them. With these offloading overheads, PIM is beneficial only for coarse-grained PIM tasks [18]. The two-mode design is a general design for PIM, and it is also adopted by other PIM architectures, such as HBM-PIM [26].

2.2 PIM-based HTAP Systems

OLAP operations are performed on massive columns and are memory-intensive [4, 6, 13, 21]. OLAP operations are proposed to be accelerated by PIM. This approach prevents the lengthy data movement between memory and CPU. Moderian [13] proposes an execution model for analytical queries on HMC-based PIM. The data are divided and moved to the destination vault for the following computation during each query, introducing additional memory movement and violating the PIM's design principle. Polynesia [6] adopts a multi-instance design on HBM, with instances stored in both CPU and PIM memory space. Their approach is still the same as the multi-instance design with mixed data formats (Figure 2 (b)) and requires rebuilding the column-store instance through logs. The rebuilding involves transferring both transaction logs and new-versioned data to the PIM memory space. Despite the integration of supplementary hardware for merging the log, this procedure still negatively impacts analytical query performance. Specifically, with 8M transactions, the query time is increased by 18% according to their experimental results (Figure 9 (b) of [6]). In summary, these approaches remain limited to existing data format designs and do not fully exploit PIM's advantage of minimizing data movement.

2.3 Database Concurrency Control

MVCC is a widely used concurrency control method for single-instance database systems. A *metadata* is maintained for every row to facilitate transactions and analytical queries [14, 22, 23]. The metadata contains three fields: a *read timestamp*, a *write timestamp*, and a *pointer*. The write timestamp records the transaction that creates the version, and the read timestamp records the transaction of the most recent read. The pointer of this version points to the previous version of this row. Upcoming transactions to the same row may form a *version chain*. Before analytical queries, the timestamps and the pointers are scanned to create a *snapshot* to summarize the visible version of data for OLAP engine processing

[6, 27, 53]. The snapshot is a collection of data pointers that have a consistent version. The analytical query is processed on the snapshot to maintain the version's consistency and make sure they do not scan the new versions created by concurrently issued transactions. For large-scale databases, the snapshot is continuously updated based on newly arriving transactions rather than being rebuilt from scratch each time [68]. Moreover, the memory space becomes fragmented because MVCC frequently allocates new memory for new-versioned rows and releases memory for stale rows. This usually reduces the memory access efficiency [43] because memory *defragmentation* is processed periodically in the database with MVCC.

3 Challenges

In this work, we propose PUSHtap, motivated by the potential to optimize HTAP through integrated interleaved CPU and localized PIM unit access. This approach brings new opportunities to achieve all three design goals using a unified data format and single-instance design. However, we still face some challenges as presented below.

(1) Data Format Challenge—Data Alignment for Bandwidth Effectiveness. We refer to the data size of each element on ADE dimension and IDE dimension's intersection as *interleave granularity*, shown as the data blocks in Figure 1 (b), indicating the minimum data size the CPU and PIM can read from a memory sub-module during each access. The interleave granularity is fixed to 8B in DIMM-based PIM due to the specification of the protocol.

According to our insight, the rows and columns are aligned to the ADE and IDE dimensions, respectively. However, with variable-sized column width, the traditional row-store or column-store cannot satisfy the requirement. For example, in table CUSTOMER of CH-Benchmark [8–10], the column width varies from 2 to 9 bytes. The conventional row-/column-stored format is shown in Figure 3 (a). If the table is row-stored, the columns are not aligned to IDE because the row size is not a multiple of the device number. For example, Column `id` of the first row is mapped to devices 0 and 1, while that of the second row is mapped to devices 1 and 2. On the other hand, if the table is column-stored as in [57], rows are not aligned to the ADE dimension. Elements of a row, e.g., `id` and `zip` of the second row, are distributed across different cache lines. This data format costs the CPU multiple bursts to access a row during transactions, resulting in low effective bandwidth.

In section 4.1, we propose a novel compact aligned format to maintain hardware alignment while optimizing bandwidth usage and preserving data integrity.

(2) Data Format Challenge—PIM Parallelism. Analytical queries are executed on only several columns. As presented in Figure 5 (a) of section 4.2, mapping a whole column to the IDE dimension actually worsens the parallelism of PIM units. We thus propose a block-circulant format to fully utilize the parallelism of PIM units to optimize analytical queries' performance in section 4.2.

(3) Data Movement Challenge in MVCC. In single-instance database design with MVCC, snapshot and defragmentation are two necessary operations. Unlike CPU-based HTAP systems, the snapshot should be transferred to PIM units in PUSHtap. This requires encoding the snapshot to minimize the data transfer. Moreover, PUSHtap focuses on compact data placement to ensure effective bandwidth. The fragmentation caused by MVCC brings severe performance degradation. We need to reduce the defragmentation overhead by transferring less data and utilizing the large PIM bandwidth to allow more frequent defragmentation execution. The snapshot and defragmentation operation designed for PUSHtap is presented in section 5.

(4) Hardware Challenge. Current PIM designs presented in section 2.1 cannot satisfy PUSHtap's requirements in the following two aspects. Firstly, in a single-instance HTAP database, the OLTP and OLAP engines process the same data instance concurrently, requiring CPU and PIM units to concurrently access the same banks. However, current PIM architectures only benefit from coarse-grained PIM tasks due to the significant offloading overhead. During the coarse-grained PIM task, PIM occupies the bank for over seconds, whether or not it is accessing the banks. This results in long transaction latency and cannot fulfill many scenarios: Many databases are timely databases and require a microsecond-level delay[15]. In section 6, we design the hardware and software interface for OLAP operations by automatically controlling the PIM units.

4 The Unified Data Format

In this section, we present the unified data format of PUSHtap that optimizes the performance of both row and column-wise operations in HTAP. We first present a *compact aligned* format method for a database with variable column widths to fit the fixed interleave granularity.

4.1 Aligned Data Format

4.1.1 Naïve Aligned Format. One naïve aligned format that all the rows and columns are aligned with ADE and IDE, respectively, is presented in Figure 3 (b). Since our example has only four devices, the table is divided into two parts. The columns `id`, `d_id`, `w_id`, and `zip` are stored in the first part, while `state` and `credit` are stored in the second part. The two parts are mapped to different memory channels so the CPU can access them in parallel. The widest column is `zip` (9 bytes), and all other columns are aligned to this width by padding zeros for a naïve alignment. In the following discussion, we refer to the width of the widest column as *row width*. Part 1's row width is 9, and Part 2's is 2.

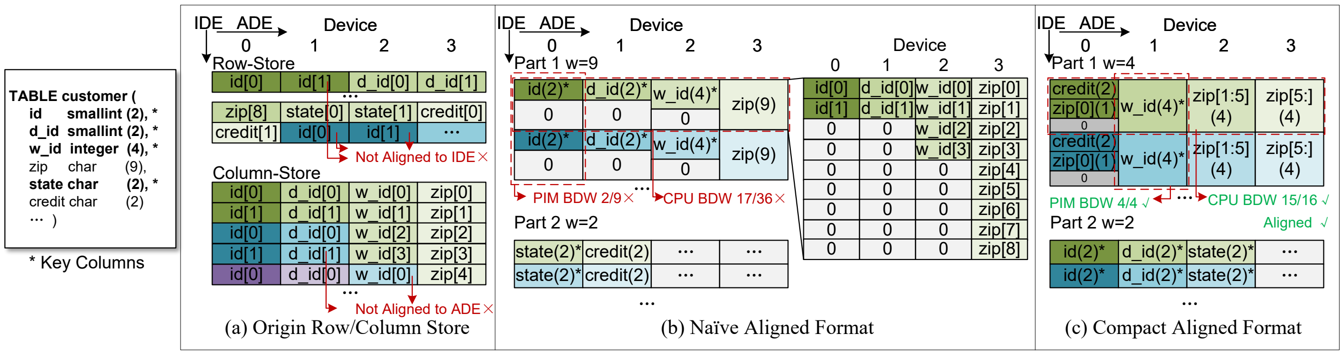


Figure 3. (a) Row-store format in current CPU-based HTAP and column-store format in current PIM-based HTAP. Different colors represent different rows. (b) A naïve aligned format for a database with various column widths. (c) The proposed compact aligned format achieves both alignment and high effective bandwidth. (w: Column width)

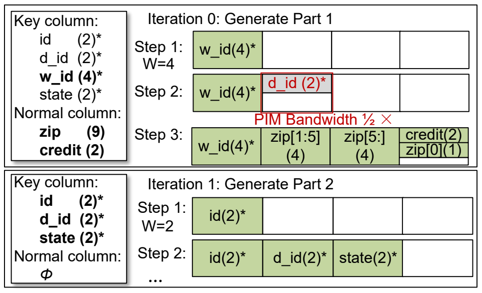


Figure 4. Generating the compact aligned format.

This naïve format wastes not only memory capacity but also CPU's and PIM's bandwidth. When the CPU reads one row in part 1, it reads 9 bytes from each device. However, only 17 of the 4 × 9 bytes contain actual data. The same situation also occurs when PIM units access DRAM devices. When the PIM unit processes column id, 8-byte contiguous data is loaded to the SRAM buffer in each access because the data wire is 64-bit wide [11]. Only 2 out of 8 bytes contain actual data, wasting 75% of the PIM bandwidth.

4.1.2 Compact Aligned Format.

To further improve the effective bandwidth, we propose a compact aligned format for PUSHtap to improve the bandwidth efficiency of PIM and CPU. We can take advantage of the following two observations. First, bytes in a row can be reordered, and columns of similar widths can be mapped to the same part, reducing zero padding. Second, some columns are not scanned in any frequent analytical queries. These OLAP-free columns can be split and mapped to multiple devices, leading to a smaller row width and reducing the number of dummy '0's padded.

In the following discussion, we denote the columns scanned by analytical queries as *key columns* and other columns as *normal columns*. For example, column id is scanned by Query 3. While column zip is not operated by any query in CH-benchmark [8, 10]. Therefore, id is a key column, while zip is a normal column.

Based on the above observations, we present a strategy based on the bin-packing algorithm to generate a compact aligned format, as shown in Figure 4. The format can keep the alignment to ADE and IDE dimensions and simultaneously achieve a high effective bandwidth. We introduce a hyperparameter, the threshold th. In our example, we set $th = 3/4$. In each iteration, we generate the format of one part of the table. Firstly, we start with an empty part and select the widest key column w_id and place it in the first device of this part. Therefore, this part's row width is determined as 4 bytes. Then in the second step, we select from the rest key columns whose width $\geq th \cdot 4$B (3B). In our example, there is no rest key column satisfying the condition. If we place a key column with a too-small width, for example, d_id of 2B width, 50% of PIM bandwidth is wasted when scanning the column. We would rather place them in the following part. In the third step, we fill the rest of the bytes with the normal columns. These columns can be divided into bytes and placed in arbitrary order. In the following iterations, we generate the format of other parts using the same strategy.

Design Trade-off by th. The underlying trade-off of setting th is as follows: A larger th ensures a highly effective PIM bandwidth as all the key columns are more compact. However, this may split the table into too many parts. CPU needs to access more cache lines to reform the row, leading to low effective CPU bandwidth. This trade-off is validated through the experiments conducted in section 7.2. The threshold is system- and workload-dependent, primarily influenced by transaction/analytical query rates and system bandwidth. Specifically, if the workload is predominantly OLTP, a lower th value can be selected to optimize CPU bandwidth. Conversely, if the workload is predominantly OLAP, a higher th value should be chosen to maximize PIM bandwidth. This threshold often remains constant during runtime. Adjustment is typically unnecessary when query rates are stable. However, specific conditions may warrant threshold

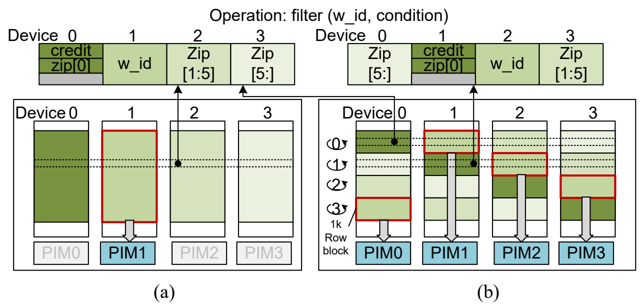


Figure 5. (a) Compact aligned format with low parallelism. (b) Block-circulant placement to fully exploit parallelism.

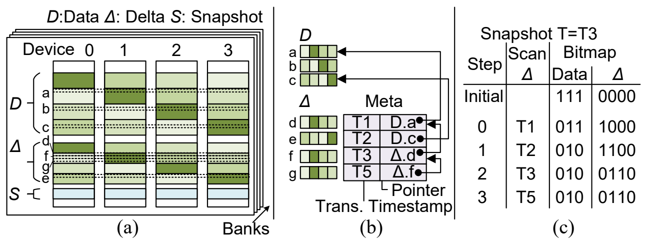


Figure 6. (a) Data region and delta region for MVCC. (b) The metadata format. (c) Snapshotting.

modifications. For instance, when CPU-side DRAM bandwidth significantly exceeds OLTP requirements, increasing the threshold can optimize OLAP performance.

Discussion on Key Column. Although for an actual HTAP database, all columns can be scanned with analytical queries, it does not mean that we need to conservatively regard all columns as key columns. We can still perform analytical queries on normal columns that are distributed across devices through the CPU, albeit with a performance loss. The indivisibility of key columns restricts the opportunity to generate a data layout with high effective bandwidth. We justify this conclusion in section 7.2. Therefore, in the actual deployment, we can prioritize the performance of frequent queries and choose fewer key columns.

Our architecture maintains compatibility with traditional methods for handling variable-width columns, though it does not specifically optimize for them. In practical implementations, variable-width columns are typically handled using traditional storage methods, such as length-prefixed encoding or separate metadata structures [57].

4.2 Block-circulant Data Placement for PIM Parallelism

With the compact alignment in the ADE dimension, one placement strategy for the IDE dimension is to align all the rows to the devices, as shown in Figure 5 (a). However, the probability of being analyzed in OLAP workloads differs across the columns. For example, eight queries analyze column `id`, while only three queries analyze column `state` [8, 10]. Moreover, these two columns are analyzed by different queries, and it is hard to schedule the two analysis tasks in parallel. Load imbalance emerges across PIM units as a "hotspot" column may be mapped to one PIM device.

We present a block-circulant data-placement strategy to fully exploit the parallelism of PIM, as shown in Figure 5 (b). The table is divided into *blocks* across the rows along the IDE dimension, and each block contains B (suppose B=1024) rows of data. In the first block (rows 0-1023), column $i (i = 0, ..., 3)$ is mapped to device i. Then, in the second block (rows 1024-2047), the columns are rotated, i.e., column i is mapped to device $(i + 1)\%4$. Subsequent blocks perform the same rotation. With block-circulant data placement, each column is evenly distributed to all the PIMs. PIM parallelism can be fully utilized when scanning any column.

The block should be large enough to prevent the formation of too many discontinuous small blocks. The block size should at least cover a row buffer of DRAM to ensure a relatively high row hit rate. We set the block size to 1024.

5 MVCC Support

MVCC is essential for single-instance HTAP databases, and PUSHtap supports MVCC. For OLTP workload, PUSHtap retains the same operational processes as traditional HTAP systems, with modifications only to the data storage format of MVCC. However, for OLAP workloads, we optimize the two MVCC operations, snapshotting and defragmentation, to minimize data communication overhead.

5.1 MVCC Storage

The data store format for supporting MVCC in PUSHtap is shown in Figure 6 (a). The storage of a table is divided into two regions, *data region* and *delta region*. The data region contains rows of original versions, while newer versions created by transactions are stored in the delta region. As metadata is not required by PIM units, it is stored in CPU memory. The new versions of a row have the same rotation as its origin row in the data region, so that we can directly use PIM units to move the newest version back to the data region during defragmentation. Therefore, the delta region is also organized into blocks.

Figure 6 (b) depicts an example of version chain generated by transactions in PUSHtap. When a transaction T1 updates row a in block 1, this row is right-rotated by 1 column, and therefore, its newer version should be put in block 1 of the delta region, which has the same rotation. The CPU allocates an empty row d in this block and records the transaction timestamps and the pointer to the origin row as metadata. Therefore, the newer version's column is aligned to its origin row. If a transaction updates a row that already has a new version, for example, transaction T3, the CPU allocates an empty row f in the same block. The pointer points to the most recent version d and forms a version chain.

5.2 Snapshotting

Snapshotting is executed before analytical queries to make PIM units operate on the rows of the correct version. In a traditional database, the snapshot is a collection of valid row pointers and is updated before every analytical query. We adopt a similar strategy to maintain the snapshots for PUSHtap. However, in PUSHtap, each row is distributed across multiple devices. Therefore, each device should maintain a copy of the snapshot. To reduce the snapshot storage overhead, we encode the snapshots into bitmaps as the address of each row is relatively fixed in PUSHtap. The snapshot contains two bitmaps containing the visible information of the data and delta regions, respectively. Each bit in the bitmap indicates the visible state of a row in the snapshot. For example, bit '1' in the i^{th} position indicates the i^{th} row is visible in the snapshot, while bit '0' indicates that the row is invisible. To minimize the data communication between CPU and PIM units, we maintain a dedicated region in memory banks to store the bitmap of rows in this bank, as shown in Figure 6 (a). In the proposed unified data format, each row is distributed across the devices in a bank; therefore, each device in this bank should store one copy of this bitmap. During snapshotting, CPU updates the bitmap according to the metadata. The bitmaps in these devices are also aligned across the ADE dimension so that CPU can update them simultaneously.

An example of snapshotting is shown in Figure 6 (c). Suppose at time `T4`, we start to execute one analytical query, and the last query is issued at `T0`. Transactions `T1`, `T2`, and `T3` are generated after the last analytical query; therefore, these three transactions have not been updated to the bitmap. During snapshotting, we need to update the bitmap according to the metadata one after another. `T1` updates row a with new version stored in row d. Therefore, the bit related to row a is set to '0', and the bit related to row d is set to '1'. Scanning `T2` and `T3` have the same operation as scanning `T1`. `T5` is generated after this analytical query is issued; therefore, it is skipped during snapshotting.

5.3 Defragmentation.

After a certain period, the transactions have updated numerous rows, and the original version of these rows in the data region is no longer used. To clean up these outdated rows, PUSHtap performs defragmentation periodically. The newest version rows in the delta region are moved to the data region and overwrite their origin rows. OLTP is paused during defragmentation to avoid data contention.

There are two candidate strategies to process the data movement. The first strategy is to move data with the CPU, which has two steps: (1) CPU reads the metadata from DRAM and merges it. (2) CPU processes the data movement according to the metadata. Due to the low bandwidth of the memory bus, copying rows is inefficient for tables with large row widths. The second strategy is to move with PIM units. Due to the data format in PUSHtap, each row is distributed across the devices. Therefore, the pointer field of the metadata should be broadcast to these devices so that PIM units in every device know where to copy the data. This strategy has three steps: (1) CPU reads out the metadata from DRAM. (2) CPU broadcasts the metadata to the devices. (3) PIM units merge the metadata and copy the new versioned data according to the metadata. Although this strategy can utilize the high bandwidth of PIM units, broadcasting the metadata involves additional communication. Therefore, this strategy is suitable when the row width is much larger than the metadata size.

We can quantify the communication overhead to apply different defragmentation strategies according to the table's row width. Suppose the DRAM rank has d devices, the row width is w, and the metadata has m bytes. The delta region has n rows, and p of these rows are the newest version and need to be copied back to the data region. CPU memory bandwidth is bdw_{CPU}, and the summation of PIM units bandwidth is bdw_{PIM}. Therefore, the metadata has a total of mn bytes. np rows of dw bytes, a total of $npdw$ bytes, are moved from the delta region to the data region. For the first strategy, the overall communication overhead is:

$$comm_{CPU} = \frac{mn + 2npdw}{bdw_{CPU}} \tag{1}$$

The first term indicates reading the metadata, and the second term is the data movement overhead. The overall communication overhead of the second strategy is:

$$comm_{PIM} = \frac{mn + dmn}{bdw_{CPU}} + \frac{dmn + 2npdw}{bdw_{PIM}} \tag{2}$$

The first term indicates CPU reading out the metadata, and the second term is CPU broadcasting them. The third term is PIM units reading metadata, and the fourth term is PIM units moving the rows. From Equation. 1 and 2, the second strategy is better when:

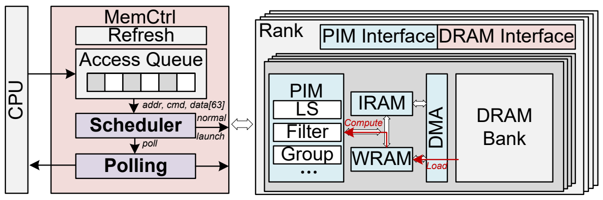
$$w > \frac{bdw_{PIM} + bdw_{CPU}}{2p \cdot (bdw_{PIM} - bdw_{CPU})} \cdot m \tag{3}$$

For example, suppose $m = 16, p \approx 1$, and $bdw_{PIM} : bdw_{CPU} = 3 : 1$, the defragmentation is better to be executed with PIM units when $w > 16$.

6 Architecture Support

6.1 Architecture Overview

To support OLAP operations, and its concurrency with normal CPU access in OLTP workload, we extend the memory controller of commercial general-purpose PIM architecture [11] with two additional components, polling module and scheduler, as shown in Figure 7 (a). The scheduler is responsible for recognizing the PIM unit control requests and orchestrating these requests and normal CPU access. The polling module is responsible for automatically polling the PIM units



(a)

<table>
<tr><th rowspan="2">Phase</th><th colspan="2">Data (64B)</th><th rowspan="2">Bank Ctrl</th></tr>
<tr><th>Type (1)</th><th>Input Parameters (63)</th></tr>
<tr><td rowspan="2">Load</td><td>LS</td><td>result_addr(3), result_len(2), result_offset(2), result_stride(2), op0_addr(3), op0_len(2), op0_offset(2),op0_stride(2),...</td><td rowspan="2">PIM</td></tr>
<tr><td>Defragment</td><td>meta_addr(3), data_addr(3), data_stride(2), delta_addr(3), delta_stride(2)</td></tr>
<tr><td rowspan="5">Compute</td><td>Filter</td><td>bitmap_offset(2), data_offset(2), result_offset(2), data_width(1), condition(8)</td><td rowspan="5">CPU</td></tr>
<tr><td>Group</td><td>bitmap_offset(2), data_offset(2), dict_offset(2), result_offset(2), data_width(1)</td></tr>
<tr><td>Aggregation</td><td>bitmap_offset(2), data_offset(2), index_offset(2), result_offset(2), data_width(1)</td></tr>
<tr><td>Hash</td><td>bitmap_offset(2), data_offset(2), result_offset(2), hash_function(4), data_width(1)</td></tr>
<tr><td>Join</td><td>hash1_offset(2), hash2_offset(2), result_offset(2), data_width(1)</td></tr>
</table>

(b)

Figure 7. (a) Architecture. (b) Data fields of launch requests.

and returning the PIM unit's finish signal to CPU. With the help of the two modules, CPU only sends one message to the DRAM controller instead of to every PIM unit when controlling PIM units. This reduces the communication overhead of each PIM task offloading and improves DRAM bandwidth utilization when CPU and PIM alternatively access DRAM.

There are two types of PIM unit control requests to be issued from CPU to execute OLAP operations, *launch* and *poll*. These requests are disguised as normal memory accesses to a special physical address. This special address is chosen from the unused DRAM address range and is preconfigured during the system boot. The launch request is disguised as a memory write. The data for writing contains the operation type and input parameters, summarized in Figure 7 (b). The poll request is disguised as a memory read. The scheduler can recognize the two requests according to their addresses and access type. When recognizing a launch request, the scheduler broadcasts the operation type and input parameters to PIM units and launches PIM units by operating on the PIM interface [62]. Note that for general-purpose DRAM-based PIM [11], the launch procedure has two main steps: handing over the control of the DRAM bank to PIM units and booting the PIM units. In PUSHtap, the scheduler only hands over the DRAM bank control to PIM units when the operation type is `LS` and `Defragment`, as other operations are processed on WRAM and do not require DRAM access. To process a poll request, the scheduler notifies the polling module to poll the PIM units. After all PIM units are finished, it returns a message to CPU through the DRAM read protocol.

Discussion on Architecture Versatility. The architecture demonstrates significant flexibility through the design of the launch request protocol. This adaptability stems from two key features: First, the CPU possesses complete configuration control over data fields, allowing tailored setups for different operations. Second, PIM units maintain programmability to interpret these customized fields. This dual-level configurability enables the architecture to support diverse scenarios, including AI and other mixed PIM-CPU tasks.

6.2 OLAP Operations Execution

The operations described in Figure 7 (b) are single-column operations processed by PIM units. In this section, we present the execution of these operations. In the conventional PIM program, the compute and load instructions are intertwined [62]. However, during the entire offloading process, the CPU's normal access is blocked, even when PIM units are executing computing instructions instead of loading data from DRAM, which does not efficiently utilize the available DRAM bandwidth. To improve the bandwidth utilization, we present a *two-phase execution* for OLAP operations. Each OLAP operations are split into *load phase* and *compute phase*, and PIM units alternatively execute the two phases until the entire column is processed. We take the filter operation as an example to present the two phases. In the load phase, the CPU program sends a launch request with operation type `LS`. DRAM bank access control is handed over to PIM units, and normal access from CPU is blocked. According to the input parameter, PIM units store the results of the last compute phase that are buffered in WRAM (offset = `result_offset`) back to DRAM and load new data to WRAM for the next compute phase. Note that we use the block-circulant data placement described in section 4.2, the real DRAM address of the data `op0` loaded by PIM unit i is calculated by `op0_stride`$*i+$ `op0_addr`. The WRAM size is 64 kB in [11] configuration, and the WRAM also serves as the operating memory for PIM units. Therefore, we use only half of the WRAM (32 kB) to store the data. According to our evaluation, it only takes 300 μs to load the 32kB data, meaning that the CPU normal memory access is blocked for no more than this period. This blocking time is short enough for most second- and microsecond-level real-time OLTP databases [28, 67]. In the compute phase, CPU sends a compute request of type `Filter`. DRAM bank access control is not handed over to PIM units, and CPU can normally access the DRAM and execute transactions. PIM units perform the filtering operation on the loaded data and store the result in the WRAM. With the two-phase execution model, the normal access from CPU is not blocked when PIM units perform the computation and can better utilize the DRAM bandwidth.

6.3 APIs for OLTP and OLAP Operations

Transaction Commit. In PUSHtap, the data in DRAM should be updated in time to ensure freshness for the OLAP workloads. Therefore, we insert additional `clflush` instructions on the rows and memory barriers at the end of commits.

Analytical Queries. PUSHtap provides a set of APIs to support the analytical queries, including filter, aggregation, and hash join. The last two operations are multi-column operations and require cooperation between the CPU and PIM units. For aggregation with a form of `SUM(col1) GROUPBY col2`, the PIM units first execute the `Group` operation to scan column `col2` and compute indices of each data. Then, CPU transfers the indices to the bank that stores the corresponding segment of column `col1` and launches PIM units to perform `Aggregation` operation. For the hash join of two index columns, we adopt the same task division in [38]. PIM units first compute the hash value of the two columns with `Hash` operation. After that, CPU fetches the hash values, divides them into buckets, and transfers them to the PIM units. Finally, PIM units perform `Join` operation in their own buckets and get the join results. In multi-column queries, columns are scanned serially, with PIM parallelism fully utilized during each scan due to block-circulant placement (section 4.2). Databases typically have sufficient rows to balance load across PIM units, ensuring efficient resource utilization.

Data Re-layout. Byte-level re-layout is a common operation in current PIM systems, typically handled by the PIM runtime [62]. In PUSHtap, we modify this function to support our data format. This function takes three inputs: the data format information of tables, the row index, and the data buffer. The data re-layout function is only invoked when 1) loading data from DRAM and 2) pushing the modified row back to memory during the transaction commit. After data is loaded, it is stored in cache in its original format, allowing CPU to perform transactions on it directly. This ensures that data re-layout only occurs when necessary, minimizing overhead and maintaining system efficiency.

7 Evaluation

7.1 Experimental Setup

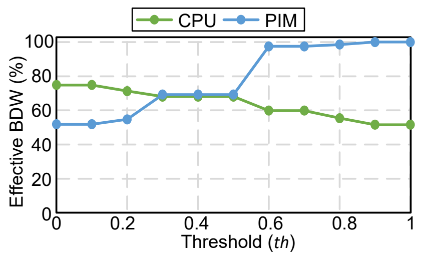
Benchmarks. We evaluate PUSHtap with CH-benchmark [8], which is a combination of two prominent database benchmarks, TPC-C [9] and TPC-H [10]. TPC-C is designed to measure OLTP performance by simulating a wholesale distribution business consisting of nine tables. We simulate two transaction types, *Payment* and *New order*, which account for approximately 90% of the TPC-C workload. We execute transactions based on an open-sourced database, DBx1000 [68], which supports the TPC-C workload and implements the MVCC scheme with row-store format. To evaluate the performance of the column store and the unified data format in PUSHtap, we extend DBx1000 with the corresponding data format. TPC-H is an OLAP workload that evaluates data warehousing performance through complex queries. We select three analytical queries from TPC-H for evaluation: aggregation-heavy query Q1, selection-heavy query Q6, and join-heavy query Q9. The three analytical queries

Table 1. System Configuration

Host CPU	
Processor	16 × O3CPU @3.2GHz
L1I/L1D	32kB / 32kB, Assoc: 8
L2 / L3	1MB Assoc: 16 / 22MB, Assoc: 22
Cache Line	64 B
DRAM DIMM	
DRAM	DDR5-3200, 8×8, 8GB/Rank
Ba / De / Ro / Co	8 / 8 / 131072 / 1024
Timing Param.	tBURST=2.5ns tRCD=tCL=tRP=7.5ns
	tRAS=16.3ns tRRD=2.5ns
	tRFC=121.9ns tWR=15.0ns tWTR=11.2ns
	tRTP=3.75ns tRTW=tCS=4.4ns tREFI=3.9us
PIM Units	
PIM Unit	500MHz, 16 tasklets, 1GB/s bandwidth [11]
	64kB WRAM, 64-bit PIM-DRAM wire width
Num	64 per Rank, at Bank level inside Devices
System Configuration	
CPU System	4 Channels ×4 Ranks normal DRAM
	4 Channels ×4 Ranks with PIM units
HBM-based System Configuration	
Host CPU and PIM Units	Same as DIMM-based system
PIM DRAM	32 Channels with PIM units
	HBM3-2Gbps, 8Gb/Bank
Pch / Bg / Ba / Ro / Co	2 / 4 / 4 / 32768 / 64
Timing Param.	tBURST=2.0ns tRCD=tCL=tRP=3.5ns
	tRAS=8.5ns tRRD=2.0ns
	tRFC=175.0ns tWR=4.0ns tWTR=1.5ns
	tRTP=1.0ns tRTW=tCS=1.5ns tREFI=2.0us

are chosen to represent different types of analytical workloads. We implement the three queries and include both the PIM and CPU overheads described in section 6.3. The row number of table `ITEM`, `STOCK`, `CUSTOMER`, `ORDER`, `ORDERLINE`, `NEWORDER`, and `HISTORY` is set to 20M, 20M, 6M, 6M, 60M, 60M, and 6M, respectively. The tables occupy 20 GB of memory storage. The queries are scheduled using the method described in [45]. We use the hash index in DBX1000 to speed up the transaction and snapshotting during analytical queries.

Simulation. The performance of PUSHtap and baselines is evaluated with the ramulator-pim [16]. We use the "OOO core" CPU model of the simulator. We extend the ramulator-pim simulator with Ramulator2 [40] for DDR5 DRAM modeling. A PIM unit frontend is integrated into Ramulator2 to support both single-bank and parallel access. We add two address mappings to support the two-dimensional access from CPU and PIM units. We extend the Ramulator2's memory controller model with the two additional modules described in section 6.1. System configuration is summarized in Table 1. The latency of handing over the bank access control is set to 0.2 μs per rank, which is measured on a real general-purpose DRAM-based PIM server with an Intel Xeon CPU of 3.2GHz [11, 62]. Half of the WRAM (32 KB) is used to store the temporary data during the load phase. The hardware modules



(a)

th=0.6

Item	Storage
Data	96.9%
Padding 0	0.8%
Snapshot	2.3%

(b)

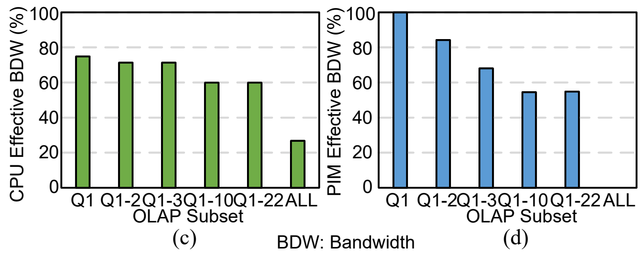


Figure 8. (a) CPU and PIM effective bandwidth under different *th*s. (b) Memory storage breakdown. (c) (d) Maximum CPU (PIM) effective bandwidth that ensures PIM (CPU) effective bandwidth > 70%, under different OLAP subsets. *Q1-10* means the subset contains from *Q1* to *Q10*.

in the memory controller are derived through Synopsys Design Compiler with TSMC 90nm technology library at the frequency of 2.4GHz.

To ensure a fair comparison with prior work [6], we extended PUSHtap to support HBM in addition to the default DIMM-based implementation. The detailed configuration of the HBM-based system is provided in Table 1. Compared to DIMM-based system, only the PIM DRAMs are replaced with HBMs. The PIM units and CPU-side configuration are kept the same. The bank number of the HBM-based system is the same as the DIMM-based system. Note that only the workload-wise performance comparison (section 7.3) contains the comparison with HBM-based systems. Other experiments are all on the default DIMM-based system.

7.2 Results on Unified Data Format

We evaluate our unified data format with the CH-benchmark. Figure 8 (a) plots the effective bandwidth of both PIM and CPU under different values of the hyperparameter *th*, consistent with the trade-off analyzed in section 4.1.2. With $th = 0$, the best CPU effective bandwidth of 74.8% is achieved while PIM effective bandwidth is the lowest at 51.9%. When *th* is set to 1, PIM effective bandwidth is maximized while CPU effective bandwidth drops significantly. To balance PIM and CPU bandwidth, we choose $th = 0.6$ in our experiment. At this value, PIM effective bandwidth reaches 97.4%, while CPU effective bandwidth is 59.8%, which is 15.0% lower than the ideal bandwidth. This effective bandwidth is sufficient for processing transactions, as they are compute-bound workloads. Simultaneously, PIM can achieve a high effective bandwidth to ensure high OLAP performance.

The breakdown of the storage space is presented in Figure 8 (b). The compact aligned format introduces negligible zero padding to the database storage, indicating its efficiency in space utilization. The snapshot bitmap for supporting MVCC in PUSHtap occupies only 2.3% additional memory storage, demonstrating its minimal overhead.

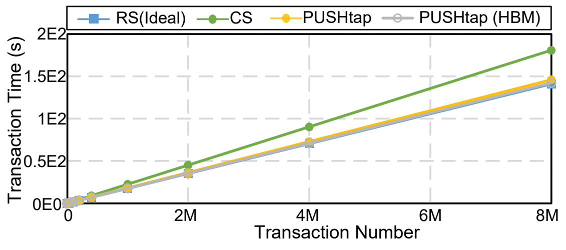
To present the impact of key columns on the HTAP performance, we plot the maximum CPU (PIM) effective bandwidth at the minimum (maximum) *th* value that ensures PIM (CPU) effective bandwidth > 70%, under different OLAP workload subsets, as shown in Figure 8 (c) and (d). More queries in the workload subset lead to more columns being regarded as key columns. For example, the subset *Q1-1* contains only 4 key columns, while the subset *Q1-3* contains 32 key columns. *ALL* represents all the columns are key columns, degraded to the naïve aligned format. With more key columns, it becomes more difficult for both CPU and PIM units to achieve high effective bandwidth. The maximum CPU effective bandwidth decreases from 74.8% to 26.7% when the workload subset increases from *Q1-1* to *ALL*. The maximum PIM effective bandwidth decreases from 100% to 54.7%. For *ALL*, the CPU effective bandwidth never exceeds 70%. Therefore, in practice, it is better to select as few key columns as possible according to the OLAP workload.

To demonstrate the generality of our format algorithm, we also tested it on HTAPBench [23]. The results show that we achieve 57%/98% CPU/PIM bandwidth utilization when th=0.55 (not shown in the figure).

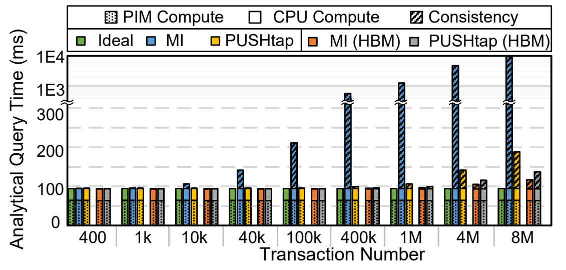
7.3 OLTP and OLAP Performance

7.3.1 OLTP Performance. We first evaluate the OLTP performance of PUSHtap, as shown in Figure 9 (a). We compare PUSHtap's data format against row-store (***RS***) and column-store (***CS***) format on DIMM-based system. The *RS* Format is considered ideal for OLTP workloads as it aligns perfectly with their row-wise processing requirements. In contrast, transactions using the *CS* format require 28.1% more execution time. This is because the *CS* format requires accessing data from every column to reconstruct the rows, leading to inefficiencies. The analytical query latency is longer than [6] because our database scale is larger. Therefore, more transactions can be executed during each analytical query. In comparison, PUSHtap only incurs a 3.5% increase in execution time compared to *RS*, which is attributed to the additional data re-layout operation. This is because PUSHtap can utilize the CPU's interleaving, as data is aligned along the ADE dimension. Our compact aligned format optimizes this by splitting the data by bytes, allowing for compact arrangement. The minor overhead observed in PUSHtap is primarily due to the data reforming.

OLTP Performance on HBM-based system. Implementing PUSHtap on HBM-based system (*PUSHtap (HBM)*)



(a)



(b)

Figure 9. (a) Execution time of transactions with row-store (*RS*), column-store (*CS*) and PUSHtap's unified data format. (b) Analytical Query time breakdown of ideal, multi-instance (*MI*) design and PUSHtap, with different transaction numbers that update the data before the analytical query. The *consistency* time includes the rebuilding (*MI*) and snapshot & defragmentation (PUSHtap).

yields merely a 2.5% speedup compared to DIMM-based system. This marginal improvement stems from two fundamental mismatches: First, the OLTP workload's non-memory-intensive nature fails to saturate HBM's high-bandwidth capabilities. Second, the large interleave granularity of HBM necessitates the loading of more data per transaction (discussed in section 8). The bandwidth advantage of HBM is offset by the increased data transfer requirements.

7.3.2 OLAP Performance. We compare PUSHtap with two DIMM-based baselines, ***MI*** and ***ideal***, to present the OLAP performance. ***ideal*** assumes that all the columns are already compact, and the execution time only includes the scanning time. ***MI*** represents the multi-instance PIM-based HTAP system with data instances on both PIM and CPU memory space, utilizing suitable column-store and row-store formats, respectively [6]. To perform a fair comparison, we adapt the architecture from [6] to the same general-purpose DIMM-based PIM architecture as PUSHtap (Figure 7 (a)) while maintaining the same methodological approach. Specifically, we replaced the PIM memory with the same DIMM DRAM modules used in PUSHtap. Instead of employing a dedicated hardware module for the rebuilding operations, we utilize the general-purpose PIM unit in the *MI* system. During the rebuilding stage, CPUs transfer all the new-versioned rows and corresponding metadata to DRAM banks, after

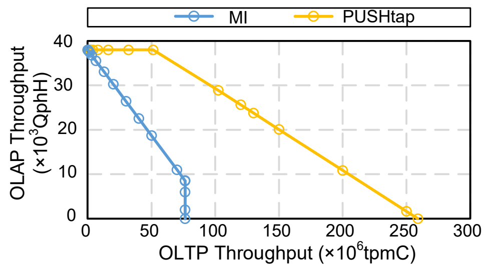


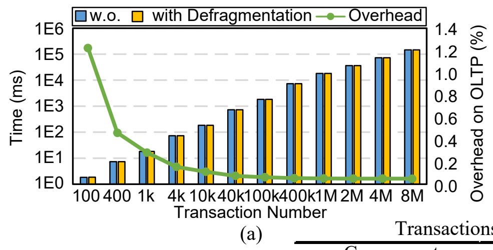
Figure 10. Throughput frontier for *MI* and PUSHtap.

which PIM units merge the metadata and copy the new-versioned data. PUSHtap generates a snapshot before every analytical query and ensures that PIM units skip old-versioned data when scanning the columns. Defragmentation (presented in section 5.3) is executed after every 10k transactions. This number is chosen based on the observation in section 7.4. The result of analytical query execution time is plotted in Figure 9 (b). Compared to *ideal*, with 1M transactions, *MI* introduces 123.3% rebuilding overhead, while PUSHtap incurs only 1.5% overhead due to snapshot and defragmentation. As the number of transactions increases (e.g., 1M), the rebuilding overhead in *MI* causes a 13.3× slowdown in the analytical queries, significantly degrading system throughput. In contrast, PUSHtap's snapshot and defragmentation overhead remains acceptable at 12.6%.

OLAP Performance on HBM-based system. We also compare the performance of PUSHtap with *MI* on HBM-based system. ***MI (HBM)*** utilizes a dedicated rebuilding accelerator to perform the rebuilding operations, which is the same as [6]. As the details of the dedicated rebuilding accelerator are not presented in [6], we estimate the rebuilding overhead by the relative value to the CPU-based consistency, which is provided in [6]. In contrast, *PUSHtap (HBM)*'s snapshotting and defragmentation are performed by CPU and general-purpose PIM units, which is the same as PUSHtap on DIMM-based system.

The result is shown in Figure 9 (b). Compared to PUSHtap on DIMM-based system, *PUSHtap (HBM)* achieves 1.4× speedup when the transaction number is 8M, primarily due to a 2.1× reduction in defragmentation time thanks to HBM's high bandwidth. With a dedicated rebuilding accelerator, *MI (HBM)* (orange bar) only introduces 24.1% rebuilding overhead, which is 4.1× lower than that of PUSHtap on DIMM-based system, leading to a 1.6× increase in OLAP throughput. However, the performance gains are relatively modest, allowing for the high cost of HBM and custom accelerators. PUSHtap on a general-purpose DIMM-based PIM architecture is a more cost-effective solution.

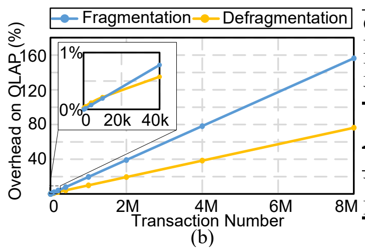
7.3.3 Performance Isolation. Figure 10 plots the frontier [42] of OLTP and OLAP throughput for *MI* and PUSHtap on DIMM-based system. Compared to *MI*, the frontier of PUSHtap is shifted to the upper right, indicating improved



Transactions

Components	Pct.
Computation	36.65%
Memory Allocation	44.10%
Indexing	19.25%
Version Chain Traverse	<0.1%

(c)



Defragmentation

Components	Pct.
Version Chain Traverse	26.39%
Data Copy	73.61%

(d)

Figure 11. (a) Execution time of OLTP w/w.o. defragmentation. We additionally display defragmentation overhead on a secondary axis. (b) OLAP overhead caused by fragmentation. *Fragmentation* is the performance degradation without defragmentation. *Defragmentation* is the cost of periodic defragmentation. (c) (d) Breakdown of (c) transaction and (d) defragmentation. Fixed overhead is not included.

performance isolation due to the elimination of rebuilding overhead in single-instance design. When the OLTP throughput of PUSHtap <51.2M tpmC (transaction per minute, TPC-C), the OLAP throughput is constant at 38.0k QphH (query per hour, TPC-H), which is the peak OLAP throughput for PUSHtap. As the OLTP throughput increases, the OLAP throughput decreases because the memory system reaches the maximum overall bandwidth. Compared to *MI*, PUSHtap can achieve 3.4× peak OLTP throughput. When the OLTP throughput reaches 76.3 MtpmC, which is the peak value of *MI*, PUSHtap can still have 4.4× OLAP throughput, indicating better performance isolation.

7.4 Defragmentation Operation

To show the necessity of defragmentation, we plot the defragmentation overhead and the performance degradation caused by fragmentation on OLAP in Figure 11(b). Without defragmentation, the analytical queries' execution time increases linearly with the transaction number, as more rows accumulate in the delta region. The rows of old versions are skipped during analytical queries. However, many row widths are smaller than 8 bytes, which is the minimum access granularity of PIM units [11]. Skipping such discrete bytes does not save PIM bandwidth usage, and PIM units still load this unnecessary data. As a result, fragmentation significantly degrades OLAP performance. Periodically executing defragmentation is necessary to ensure OLAP performance. When the transaction number exceeds 10k, the overhead caused by fragmentation is larger than the defragmentation

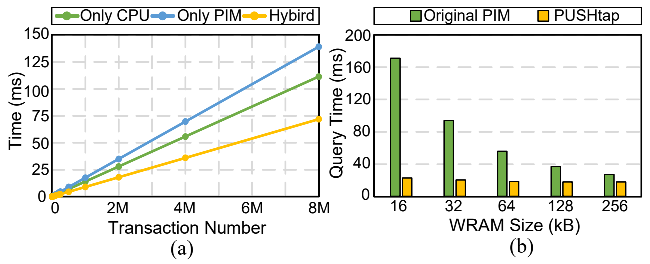


Figure 12. (a) Defragmentation time with (1) purely CPU, (2) purely PIM units, and (3) strategy in section 5.3 (*Hybrid*). (b) *Q6* execution time across different WRAM sizes.

overhead (2.05×). This is because the fixed overhead, including thread creation and PIM units activation, is amortized when the number of transactions is large. Therefore, we execute defragmentation every 10k transactions to minimize the overhead while preserving OLAP performance.

The defragmentation overhead on OLTP across varying transaction numbers is shown in Figure 11 (a). It represents the ratio of defragmentation time to total transaction time. In contrast with OLAP, the defragmentation only introduces <1.5% overhead to OLTP. Figure 11 (c) and (d) present the breakdown of transaction and defragmentation time. Less than 0.1% time of each transaction is spent on traversing the version chain, as only one version chain is traversed per transaction. Most time is occupied by indexing, memory allocation, and computation. Defragmentation on a row involves traversing the version chain and copying the data, which is negligible compared to a transaction.

Figure 12 (a) plots the defragmentation time with (1) only CPU, (2) only PIM units, and (3) strategy presented in section 5.3 (denoted as *Hybrid*). Neither CPU- nor PIM-only strategy can achieve optimal defragmentation efficiency. With our unified data format, the table parts' row width varies from 2 bytes to over 20 bytes. According to our conclusion in section 5.3, these parts are suitable for different strategies. The *hybrid* selects different strategies depending on the tables' row widths and can achieve the best efficiency.

7.5 Architecture Comparison

We compare the PIM architecture of PUSHtap with the original general-purpose DRAM-based PIM architecture [11]. Both architectures adopt the two-phase execution presented in section 6.2, with the only difference on the communication overhead presented in section 6.1. Figure 12 (b) depicts the execution time under different WRAM sizes. With a larger WRAM, fewer load phases are required, reducing the CPU-PIM mode switch overhead. The execution time of the original PIM architecture decreases by 6.4× when increasing WRAM size from 16 kB to 256 kB, as the mode switching overhead drops from 88.8% to 35.3% of the computing time. Accordingly, the period of the load phase increases to > 1 ms when the WRAM size increases to 256 kB, limiting the usage of real-time OLTP. The CPU-PIM mode switch overhead has minimal effect on the performance of PUSHtap, as the mode switch function is offloaded to memory channels and only accounts for 7.0% of the computing time on average. For the default 64kB configuration, PUSHtap can achieve 3.0× speedup compared to the original PIM architecture.

7.6 Area Overhead.

The additional hardware modules introduce minimal area overhead of 0.115 mm^2 in an 8-channel memory controller, with the scheduler occupying 0.112 mm^2 and the polling module requiring only 0.003 mm^2. This overhead is negligible compared to the total memory controller area of approximately 13 mm^2 [44].

8 Discussion

PIM Technique Selection. As memory interleaving is a well-established technique used in various memory systems, making it feasible to integrate PIM units into these existing systems [11, 26, 30, 34]. For PIM-based HTAP architecture with unified data format, the interleave granularity should be fine enough so that both CPU and PIM can efficiently access the small data elements in databases. For instance, `ORDERLINE` table's `amount` column is only 8 bytes in size. If the interleave granularity is set to 64 bytes, we should access an additional 48 bytes to acquire this 8-byte valid data, which results in bandwidth waste.

We compare three techniques — HBM, DIMM, and SSD — on their interleave granularity. We take the CH-benchmark as an example [8, 10]. Its column width varies from 2 bytes to 152 bytes. SSD's interleave granularity is ≈ 1 MB. HBM provides a 64-byte (or 32-byte) granularity. DIMM offers the finest granularity at 8 bytes. DIMM's finest granularity allows us to optimize access waste through mapping methods. Therefore, we choose DIMM DRAM as the PIM storage.

Compact Aligned Format on Command-Driven PIM. The data format method employed in PUSHtap is also well-suited for command-driven PIM architectures [21, 34]. These architectures typically require a high degree of customization of the database instruction set. In such an architecture, we can design specialized accelerators like in [6] to expedite time-consuming operations, such as defragmentation and data layout, which were identified as performance bottlenecks in our evaluations. By leveraging these accelerators, we can further reduce the performance gap between OLTP/OLAP operations and their ideal performance benchmarks. This enhancement ultimately boosts overall system throughput. We plan to explore this potential in future research to fully realize the benefits of these optimizations.

9 Related Work

PIM. In PIM, PIM units can be located in various memory hierarchies, including cache [47, 64–66], DRAM memory (DDR[11, 19, 24, 29, 38],GDDR[34], HBM[6, 26]), and SSD [30, 63]. There is no direct connection between the PIM units distributed in sub-modules of memory; the communication has to go through the CPU, resulting in high-cost inter-PIM-unit communication. Current works treat the PIM units as distributed systems. They are devoted to static task division to ensure PIM load balance and low communication overhead [1, 5, 36, 38, 39, 71]. While others propose to add additional connections and access modes to realize an automatic communication and load balance [58, 60, 61, 70].

The distributed PIM units also face the conflict that memory interleaving prevents PIM units from being visible to a contiguous block of data. Some workloads, e.g. element-wise operations, can coexist with interleaving as PIM units still have access to complete data elements [7, 24, 46]. For a general-purpose scenario, some works[11, 12, 26] divide a dedicated PIM memory space from the main memory and manually re-layout the data when writing to the space. UM-PIM [69] proposes to use dynamic address mapping to enable two different memory pages with different data layouts to co-exist in the PIM system. In this work, instead of resolving the conflict, we utilize PIM and memory interleaving to provide two-dimensional memory access. We exploit its benefit in an appropriate scenario – HTAP.

HTAP Architecture. HTAP research mainly focuses on data storage format [27, 28, 49, 53, 54], query optimization [3, 32, 35, 56], indexing technique [23, 51], scheduling [50, 53, 55], and accerleration with computational storage [31, 63].

In-Memory Database. In-memory database is proposed to provide real-time transaction processing [59]. Compute Express Link (CXL) technique is introduced to improve in-memory database's scalability [2] and to provide a new solution for durability [17]. The CXL latency can be hidden by properly prefetching data to local memory or cache in OLTP workloads [2].

10 Conclusion

This work proposes PUSHtap, a PIM-based HTAP system with a unified data storage format tailored for both OLTP and OLAP workloads. By combining the access of PIM units and CPU, we create a two-dimensional access memory space. We introduce a unified data format to enhance effective bandwidth, ensure PIM load balance, and support MVCC. PUSHtap also allows concurrent CPU and PIM access, optimizing HTAP performance. Extensive experiments show that PUSHtap can deliver satisfactory performance gain.

Acknowledgments

We sincerely thank our shepherd, Prof. Onur Mutlu, and the anonymous reviewers for their comments and suggestions to improve the paper. We also thank Huan Zhou for improving the figures.

References

[1] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. A scalable processing-in-memory accelerator for parallel graph processing. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. 105–117. doi:10.1145/2749469.2750386

[2] Minseon Ahn, Andrew Chang, Donghun Lee, Jongmin Gim, Jungmin Kim, Jaemin Jung, Oliver Rebholz, Vincent Pham, Krishna Malladi, and Yang Seok Ki. 2022. Enabling CXL Memory Expansion for In-Memory Database Management Systems. In *Proceedings of the 18th International Workshop on Data Management on New Hardware* (Philadelphia, PA, USA) *(DaMoN '22)*. Association for Computing Machinery, New York, NY, USA, Article 8, 5 pages. doi:10.1145/3533737.3535090

[3] Raja Appuswamy, Manos Karpathiotakis, Danica Porobic, and Anastasia Ailamaki. 2017. The Case For Heterogeneous HTAP. https://infoscience.epfl.ch/handle/20.500.14299/132929

[4] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. 2012. Workload analysis of a large-scale key-value store. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems* (London, England, UK) *(SIGMETRICS '12)*. Association for Computing Machinery, New York, NY, USA, 53–64. doi:10.1145/2254756.2254766

[5] Daehyeon Baek, Soojin Hwang, and Jaehyuk Huh. 2024. pSyncPIM: Partially Synchronous Execution of Sparse Matrix Operations for All-Bank PIM Architectures. In *Proceedings of the 2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)* (Buenos Aries, Argentina) *(ISCA'24)*. 354–367.

[6] Amirali Boroumand, Saugata Ghose, Geraldo F. Oliveira, and Onur Mutlu. 2022. Polynesia: Enabling High-Performance and Energy-Efficient Hybrid Transactional/Analytical Databases with Hardware/Software Co-Design. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. 2997–3011. doi:10.1109/ICDE53745.2022.00270

[7] Benjamin Y. Cho, Yongkee Kwon, Sangkug Lym, and Mattan Erez. 2020. Near Data Acceleration with Concurrent Host Access. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 818–831. doi:10.1109/ISCA45697.2020.00072

[8] Richard Cole, Florian Funke, Leo Giakoumakis, Wey Guy, Alfons Kemper, Stefan Krompass, Harumi Kuno, Raghunath Nambiar, Thomas Neumann, Meikel Poess, Kai-Uwe Sattler, Michael Seibold, Eric Simon, and Florian Waas. 2011. The mixed workload CH-benCHmark. In *Proceedings of the Fourth International Workshop on Testing Database Systems* (Athens, Greece) *(DBTest '11)*. Association for Computing Machinery, New York, NY, USA, Article 8, 6 pages. doi:10.1145/1988842.1988850

[9] Transaction Processing Performance Council. 2010. TPC BENCHMARK C Standard Specification Revision 5.11. https://www.tpc.org/TPC_Documents_Current_Versions/pdf/tpc-c_v5.11.0.pdf Accessed on July, 2024.

[10] Transaction Processing Performance Council. 2022. TPC BENCHMARK H Standard Specification Revision 3.0.1. https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-H_v3.0.1.pdf Accessed on July, 2024.

[11] Fabrice Devaux. 2019. The true Processing In Memory accelerator. In *2019 IEEE Hot Chips 31 Symposium (HCS)*. 1–24. doi:10.1109/HOTCHIPS.2019.8875680

[12] Alexandar Devic, Siddhartha Balakrishna Rai, Anand Sivasubramaniam, Ameen Akel, Sean Eilert, and Justin Eno. 2022. To PIM or not for emerging general purpose processing in DDR memory systems. In *Proceedings of the 49th Annual International Symposium on Computer Architecture* (New York, New York) *(ISCA '22)*. Association for Computing Machinery, New York, NY, USA, 231–244. doi:10.1145/3470496.3527431

[13] Mario Drumond, Alexandros Daglis, Nooshin Mirzadeh, Dmitrii Ustiugov, Javier Picorel, Babak Falsafi, Boris Grot, and Dionisios Pnevmatikatos. 2017. The mondrian data engine. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. 639–651. doi:10.1145/3079856.3080233

[14] Michael Freitag, Alfons Kemper, and Thomas Neumann. 2022. Memory-optimized multi-version concurrency control for disk-based database systems. *Proc. VLDB Endow.* 15, 11 (jul 2022), 2797–2810. doi:10.14778/3551793.3551832

[15] Anil K. Goel, Jeffrey Pound, Nathan Auch, Peter Bumbulis, Scott MacLean, Franz Färber, Francis Gropengiesser, Christian Mathis, Thomas Bodner, and Wolfgang Lehner. 2015. Towards scalable real-time analytics: an architecture for scale-out of OLxP workloads. *Proc. VLDB Endow.* 8, 12 (aug 2015), 1716–1727. doi:10.14778/2824032.2824069

[16] SAFARI Research Group. 2023. ZSim+Ramulator - A Processing-in-Memory Simulation Framework. https://github.com/CMU-SAFARI/ramulator-pim Accessed on July, 2024.

[17] Yunyan Guo and Guoliang Li. 2024. A CXL- Powered Database System: Opportunities and Challenges. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 5593–5604. doi:10.1109/ICDE60146.2024.00447

[18] Juan Gómez-Luna, Izzat El Hajj, Ivan Fernandez, Christina Giannoula, Geraldo F. Oliveira, and Onur Mutlu. 2022. Benchmarking a New Paradigm: Experimental Analysis and Characterization of a Real Processing-in-Memory System. *IEEE Access* 10 (2022), 52565–52608. doi:10.1109/ACCESS.2022.3174101

[19] Liu Ke, Udit Gupta, Benjamin Youngjae Cho, David Brooks, Vikas Chandra, Utku Diril, Amin Firoozshahian, Kim Hazelwood, Bill Jia, Hsien-Hsin S. Lee, Meng Li, Bert Maher, Dheevatsa Mudigere, Maxim Naumov, Martin Schatz, Mikhail Smelyanskiy, Xiaodong Wang, Brandon Reagen, Carole-Jean Wu, Mark Hempstead, and Xuan Zhang. 2020. RecNMP: Accelerating Personalized Recommendation with Near-Memory Processing. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 790–803. doi:10.1109/ISCA45697.2020.00070

[20] Alfons Kemper and Thomas Neumann. 2011. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In *2011 IEEE 27th International Conference on Data Engineering*. 195–206. doi:10.1109/ICDE.2011.5767867

[21] Donghyuk Kim, Jae-Young Kim, Wontak Han, Jongsoon Won, Haerang Choi, Yongkee Kwon, and Joo-Young Kim. 2023. Darwin: A DRAM-based Multi-level Processing-in-Memory Architecture for Data Analytics. *arXiv preprint arXiv:2305.13970* (2023).

[22] Jongbin Kim, Hyunsoo Cho, Kihwang Kim, Jaeseon Yu, Sooyong Kang, and Hyungsoo Jung. 2020. Long-lived Transactions Made Less Harmful. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 495–510. doi:10.1145/3318464.3389714

[23] Jongbin Kim, Jaeseon Yu, Jaechan Ahn, Sooyong Kang, and Hyungsoo Jung. 2022. Diva: Making MVCC Systems HTAP-Friendly. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 49–64. doi:10.1145/3514221.3526135

[24] Youngeun Kwon, Yunjae Lee, and Minsoo Rhu. 2019. TensorDIMM: A Practical Near-Memory Processing Architecture for Embeddings and Tensor Operations in Deep Learning. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture* (Columbus, OH, USA) *(MICRO '52)*. Association for Computing Machinery, New York, NY, USA, 740–753. doi:10.1145/3352460.3358284

[25] Yongkee Kwon, Kornijcuk Vladimir, Nahsung Kim, Woojae Shin, Jongsoon Won, Minkyu Lee, Hyunha Joo, Haerang Choi, Guhyun Kim, Byeongju An, Jeongbin Kim, Jaewook Lee, Ilkon Kim, Jaehan Park,

Chanwook Park, Yosub Song, Byeongsu Yang, Hyungdeok Lee, Seho Kim, Daehan Kwon, Seongju Lee, Kyuyoung Kim, Sanghoon Oh, Joonhong Park, Gimoon Hong, Dongyoon Ka, Kyudong Hwang, Jeongje Park, Kyeongpil Kang, Jungyeon Kim, Junyeol Jeon, Myeongjun Lee, Minyoung Shin, Minhwan Shin, Jaekyung Cha, Changson Jung, Kijoon Chang, Chunseok Jeong, Euicheol Lim, Il Park, Junhyun Chun, and Sk Hynix. 2022. System Architecture and Software Stack for GDDR6-AiM. In *2022 IEEE Hot Chips 34 Symposium (HCS)*. 1–25. doi:10.1109/HCS55958.2022.9895629

[26] Young-Cheon Kwon, Suk Han Lee, Jaehoon Lee, Sang-Hyuk Kwon, Je Min Ryu, Jong-Pil Son, O Seongil, Hak-Soo Yu, Haesuk Lee, Soo Young Kim, Youngmin Cho, Jin Guk Kim, Jongyoon Choi, Hyun-Sung Shin, Jin Kim, BengSeng Phuah, HyoungMin Kim, Myeong Jun Song, Ahn Choi, Daeho Kim, SooYoung Kim, Eun-Bong Kim, David Wang, Shinhaeng Kang, Yuhwan Ro, Seungwoo Seo, JoonHo Song, Jaeyoun Youn, Kyomin Sohn, and Nam Sung Kim. 2021. 25.4 A 20nm 6GB Function-In-Memory DRAM, Based on HBM2 with a 1.2TFLOPS Programmable Computing Unit Using Bank-Level Parallelism, for Machine Learning Applications. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 64. 350–352. doi:10.1109/ISSCC42613.2021.9365862

[27] Tirthankar Lahiri, Shasank Chavan, Maria Colgan, Dinesh Das, Amit Ganesh, Mike Gleeson, Sanket Hase, Allison Holloway, Jesse Kamp, Teck-Hua Lee, Juan Loaiza, Neil Macnaughton, Vineet Marwah, Niloy Mukherjee, Atrayee Mullick, Sujatha Muthulingam, Vivekanandhan Raja, Marty Roth, Ekrem Soylemez, and Mohamed Zait. 2015. Oracle Database In-Memory: A dual format in-memory database. In *2015 IEEE 31st International Conference on Data Engineering*. 1253–1258. doi:10.1109/ICDE.2015.7113373

[28] Per-Åke Larson, Adrian Birka, Eric N. Hanson, Weiyun Huang, Michal Nowakiewicz, and Vassilis Papadimos. 2015. Real-time analytical processing with SQL server. *Proc. VLDB Endow.* 8, 12 (aug 2015), 1740–1751. doi:10.14778/2824032.2824071

[29] Donghun Lee, Jinin So, MINSEON AHN, Jong-Geon Lee, Jungmin Kim, Jeonghyeon Cho, Rebholz Oliver, Vishnu Charan Thummala, Ravi shankar JV, Sachin Suresh Upadhya, Mohammed Ibrahim Khan, and Jin Hyun Kim. 2022. Improving In-Memory Database Operations with Acceleration DIMM (AxDIMM). In *Proceedings of the 18th International Workshop on Data Management on New Hardware* (Philadelphia, PA, USA) *(DaMoN '22)*. Association for Computing Machinery, New York, NY, USA, Article 2, 9 pages. doi:10.1145/3533737.3535093

[30] Joo Hwan Lee, Hui Zhang, Veronica Lagrange, Praveen Krishnamoorthy, Xiaodong Zhao, and Yang Seok Ki. 2020. SmartSSD: FPGA Accelerated Near-Storage Data Analytics on SSD. *IEEE Computer Architecture Letters* 19, 2 (July 2020), 110–113. doi:10.1109/LCA.2020.3009347

[31] Kitaek Lee, Insoon Jo, Jaechan Ahn, Hyuk Lee, Hwang Lee, Woong Sul, and Hyungsoo Jung. 2023. Deploying Computational Storage for HTAP DBMSs Takes More Than Just Computation Offloading. *Proc. VLDB Endow.* 16, 6 (feb 2023), 1480–1493. doi:10.14778/3583140.3583161

[32] Rubao Lee, Minghong Zhou, Chi Li, Shenggang Hu, Jianping Teng, Dongyang Li, and Xiaodong Zhang. 2021. The art of balance: a RateupDB™ experience of building a CPU/GPU hybrid database product. *Proc. VLDB Endow.* 14, 12 (jul 2021), 2999–3013. doi:10.14778/3476311.3476378

[33] Sukhan Lee, Shin-haeng Kang, Jaehoon Lee, Hyeonsu Kim, Eojin Lee, Seungwoo Seo, Hosang Yoon, Seungwon Lee, Kyounghwan Lim, Hyunsung Shin, Jinhyun Kim, O Seongil, Anand Iyer, David Wang, Kyomin Sohn, and Nam Sung Kim. 2021. Hardware Architecture and Software Stack for PIM Based on Commercial DRAM Technology : Industrial Product. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 43–56. doi:10.1109/ISCA52012.2021.00013

[34] Seongju Lee, Kyuyoung Kim, Sanghoon Oh, Joonhong Park, Gimoon Hong, Dongyoon Ka, Kyudong Hwang, Jeongje Park, Kyeongpil Kang, Jungyeon Kim, Junyeol Jeon, Nahsung Kim, Yongkee Kwon, Kornijcuk Vladimir, Woojae Shin, Jongsoon Won, Minkyu Lee, Hyunha Joo, Haerang Choi, Jaewook Lee, Donguc Ko, Younggun Jun, Keewon Cho, Ilwoong Kim, Choungki Song, Chunseok Jeong, Daehan Kwon, Jieun Jang, Il Park, Junhyun Chun, and Joohwan Cho. 2022. A 1ynm 1.25V 8Gb, 16Gb/s/pin GDDR6-based Accelerator-in-Memory supporting 1TFLOPS MAC Operation and Various Activation Functions for Deep-Learning Applications. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 65. 1–3. doi:10.1109/ISSCC42614.2022.9731711

[35] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How good are query optimizers, really? *Proc. VLDB Endow.* 9, 3 (nov 2015), 204–215. doi:10.14778/2850583.2850594

[36] Marzieh Lenjani, Alif Ahmed, Mircea Stan, and Kevin Skadron. 2022. Gearbox: a case for supporting accumulation dispatching and hybrid partitioning in PIM-based accelerators. In *Proceedings of the 49th Annual International Symposium on Computer Architecture* (New York, New York) *(ISCA '22)*. Association for Computing Machinery, New York, NY, USA, 218–230. doi:10.1145/3470496.3527402

[37] Guoliang Li and Chao Zhang. 2022. HTAP Databases: What is New and What is Next. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 2483–2488. doi:10.1145/3514221.3522565

[38] Chaemin Lim, Suhyun Lee, Jinwoo Choi, Jounghoo Lee, Seongyeon Park, Hanjun Kim, Jinho Lee, and Youngsok Kim. 2023. Design and Analysis of a Processing-in-DIMM Join Algorithm: A Case Study with UPMEM DIMMs. *Proc. ACM Manag. Data* 1, 2, Article 113 (jun 2023), 27 pages. doi:10.1145/3589258

[39] Haifeng Liu, Long Zheng, Yu Huang, Chaoqiang Liu, Xiangyu Ye, Jingrui Yuan, Xiaofei Liao, Hai Jin, and Jingling Xue. 2023. Accelerating Personalized Recommendation with Cross-level Near-Memory Processing. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (Orlando, FL, USA) *(ISCA '23)*. Association for Computing Machinery, New York, NY, USA, Article 66, 13 pages. doi:10.1145/3579371.3589101

[40] Haocong Luo, Yahya Can Tuğrul, F. Nisa Bostancı, Ataberk Olgun, A. Giray Yağlıkçı, and Onur Mutlu. 2023. Ramulator 2.0: A Modern, Modular, and Extensible DRAM Simulator. arXiv:2308.11030 [cs.AR] https://arxiv.org/abs/2308.11030

[41] Darko Makreshanski, Jana Giceva, Claude Barthels, and Gustavo Alonso. 2017. BatchDB: Efficient Isolated Execution of Hybrid OLTP+OLAP Workloads for Interactive Applications. In *Proceedings of the 2017 ACM International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '17)*. Association for Computing Machinery, New York, NY, USA, 37–50. doi:10.1145/3035918.3035959

[42] Elena Milkai, Yannis Chronis, Kevin P. Gaffney, Zhihan Guo, Jignesh M. Patel, and Xiangyao Yu. 2022. How Good is My HTAP System?. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 1810–1824. doi:10.1145/3514221.3526148

[43] MySQL. 2024. MySQL 8.0 Reference Manual: OPTIMIZE TABLE Statement. https://dev.mysql.com/doc/refman/8.0/en/optimize-table.html Accessed on July, 2024.

[44] Nevine Nassif, Ashley O. Munch, Carleton L. Molnar, Gerald Pasdast, Sitaraman V. Lyer, Zibing Yang, Oscar Mendoza, Mark Huddart, Srikrishnan Venkataraman, Sireesha Kandula, Rafi Marom, Alexandra M. Kern, Bill Bowhill, David R. Mulvihill, Srikanth Nimmagadda, Varma Kalidindi, Jonathan Krause, Mohammad M. Haq, Roopali Sharma, and Kevin Duda. 2022. Sapphire Rapids: The Next-Generation Intel Xeon Scalable Processor. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 65. 44–46. doi:10.1109/ISSCC42614.2022.9731107

[45] Thomas Neumann. 2011. Efficiently compiling efficient query plans for modern hardware. *Proc. VLDB Endow.* 4, 9 (jun 2011), 539–550.

doi:10.14778/2002938.2002940

[46] Ataberk Olgun, Juan Gómez Luna, Konstantinos Kanellopoulos, Behzad Salami, Hasan Hassan, Oguz Ergin, and Onur Mutlu. 2022. PiDRAM: A Holistic End-to-end FPGA-based Framework for Processing-in-DRAM. *ACM Trans. Archit. Code Optim.* 20, 1, Article 8 (nov 2022), 31 pages. doi:10.1145/3563697

[47] Marcelo Orenes-Vera, Esin Tureci, David Wentzlaff, and Margaret Martonosi. 2023. Dalorex: A Data-Local Program Execution and Architecture for Memory-bound Applications. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 718–730. doi:10.1109/HPCA56546.2023.10071089

[48] Massimo Pezzini, Donald Feinberg, Nigel Rayner, and Roxane Edjlali. 2014. Hybrid transaction/analytical processing will foster opportunities for dramatic business innovation. *Gartner (2014, January 28) Available at https://www.gartner.com/doc/2657815/hybrid-transactionanalyticalprocessing-foster-opportunities* (2014), 4–20.

[49] PingCAP. 2024. TiDB: a distributed SQL database. https://github.com/pingcap/tidb Accessed on July, 2024.

[50] Aunn Raza, Periklis Chrysogelos, Angelos Christos Anadiotis, and Anastasia Ailamaki. 2020. Adaptive HTAP through Elastic Resource Scheduling. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 2043–2054. doi:10.1145/3318464.3389783

[51] Christian Riegger, Tobias Vinçon, Robert Gottstein, and Ilia Petrov. 2020. MV-PBT: multi-version indexing for large datasets and HTAP workloads. In *Advances in data base technology-EDBT 2020: 23rd International Conference on Extending Database Technology, Copenhagen, Denmark, March 30-April 2, 2020: proceedings*. Open Proceedings. org, Univ. of Konstanz, 217–228.

[52] Shahin Roozkhosh, Denis Hoornaert, Ju Hyoung Mun, Tarikul Islam Papon, Ulrich Drepper, Renato Mancuso, and Manos Athanassoulis. 2021. Relational Memory: Native In-Memory Accesses on Rows and Columns. *CoRR* abs/2109.14349 (2021). arXiv:2109.14349 https://arxiv.org/abs/2109.14349

[53] Vishal Sikka, Franz Färber, Wolfgang Lehner, Sang Kyun Cha, Thomas Peh, and Christof Bornhövd. 2012. Efficient transaction processing in SAP HANA database: the end of a column store myth. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (Scottsdale, Arizona, USA) *(SIGMOD '12)*. Association for Computing Machinery, New York, NY, USA, 731–742. doi:10.1145/2213836.2213946

[54] SingleStore. [n. d.]. SingleStore Documentation, SingleStore Helios. [Online]. https://docs.singlestore.com/.

[55] Utku Sirin, Sandhya Dwarkadas, and Anastasia Ailamaki. 2021. Performance Characterization of HTAP Workloads. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. 1829–1834. doi:10.1109/ICDE51399.2021.00162

[56] Haoze Song, Wenchao Zhou, Feifei Li, Xiang Peng, and Heming Cui. 2023. Rethink Query Optimization in HTAP Databases. *Proc. ACM Manag. Data* 1, 4, Article 256 (dec 2023), 27 pages. doi:10.1145/3626750

[57] Mike Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Sam Madden, Elizabeth O'Neil, Pat O'Neil, Alex Rasin, Nga Tran, and Stan Zdonik. 2018. *C-store: a column-oriented DBMS*. Association for Computing Machinery and Morgan & Claypool, 491–518. https://doi.org/10.1145/3226595.3226638

[58] Weiyi Sun, Zhaoshi Li, Shouyi Yin, Shaojun Wei, and Leibo Liu. 2021. ABC-DIMM: Alleviating the Bottleneck of Communication in DIMM-based Near-Memory Processing with Inter-DIMM Broadcast. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 237–250. doi:10.1109/ISCA52012.2021.00027

[59] Kian-Lee Tan, Qingchao Cai, Beng Chin Ooi, Weng-Fai Wong, Chang Yao, and Hao Zhang. 2015. In-memory Databases: Challenges and Opportunities From Software and Hardware Perspectives. *SIGMOD Rec.* 44, 2 (Aug. 2015), 35–40. doi:10.1145/2814710.2814717

[60] Boyu Tian, Qihang Chen, and Mingyu Gao. 2023. ABNDP: Co-optimizing Data Access and Load Balance in Near-Data Processing. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 3–17. doi:10.1145/3582016.3582026

[61] Boyu Tian, Yiwei Li, Jiang Li, Shuangyu Cai, and Mingyu Gao. 2024. NDPBridge: Enabling Cross-Bank Coordination in Near-DRAM-Bank Processing Architectures. In *Proceedings of the 51st Annual International Symposium on Computer Architecture (ISCA '24)*. Association for Computing Machinery, Article 1, 16 pages.

[62] UPMEM. 2023. The UPMEM DPU toolchain – UPMEM DPU SDK 2023.2.0 Documentation. https://sdk.upmem.com/2023.2.0/.

[63] Tobias Vinçon, Christian Knödler, Leonardo Solis-Vasquez, Arthur Bernhardt, Sajjad Tamimi, Lukas Weber, Florian Stock, Andreas Koch, and Ilia Petrov. 2022. Near-data processing in database systems on native computational storage under HTAP workloads. *Proc. VLDB Endow.* 15, 10 (jun 2022), 1991–2004. doi:10.14778/3547305.3547307

[64] Zhengrong Wang, Christopher Liu, Aman Arora, Lizy John, and Tony Nowatzki. 2023. Infinity Stream: Portable and Programmer-Friendly In-/Near-Memory Fusion. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 359–375. doi:10.1145/3582016.3582032

[65] Zhengrong Wang, Christopher Liu, Nathan Beckmann, and Tony Nowatzki. 2023. Affinity Alloc: Taming Not - So Near-Data Computing. In *2023 56th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 784–799.

[66] Zhengrong Wang, Jian Weng, Sihao Liu, and Tony Nowatzki. 2022. Near-Stream Computing: General and Transparent Near-Cache Acceleration. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 331–345. doi:10.1109/HPCA53966.2022.00032

[67] Fangjin Yang, Eric Tschetter, Xavier Léauté, Nelson Ray, Gian Merlino, and Deep Ganguli. 2014. Druid: a real-time analytical data store. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (Snowbird, Utah, USA) *(SIGMOD '14)*. Association for Computing Machinery, New York, NY, USA, 157–168. doi:10.1145/2588555.2595631

[68] Xiangyao Yu, George Bezerra, Andrew Pavlo, Srinivas Devadas, and Michael Stonebraker. 2014. Staring into the abyss: an evaluation of concurrency control with one thousand cores. *Proc. VLDB Endow.* 8, 3 (nov 2014), 209–220. doi:10.14778/2735508.2735511

[69] Yilong Zhao, Mingyu Gao, Fangxin Liu, Yiwei Hu, Wang Zongwu, Han Lin, Ji Li, He Xian, Hanlin Dong, Tao Yang, Naifeng Jing, Xiaoyao Liang, and Li Jiang. 2024. UM-PIM: DRAM-Based PIM with Uniform & Shared Memory Space. In *Proceedings of the 51st Annual International Symposium on Computer Architecture (ISCA '24)*. Association for Computing Machinery, Article 1, 16 pages.

[70] Zhe Zhou, Cong Li, Fan Yang, and Guangyu Sun. 2023. DIMM-Link: Enabling Efficient Inter-DIMM Communication for Near-Memory Processing. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 302–316. doi:10.1109/HPCA56546.2023.10071005

[71] Youwei Zhuo, Chao Wang, Mingxing Zhang, Rui Wang, Dimin Niu, Yanzhi Wang, and Xuehai Qian. 2019. GraphQ: Scalable PIM-Based Graph Processing. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture* (Columbus, OH, USA) *(MICRO '52)*. Association for Computing Machinery, New York, NY, USA, 712–725. doi:10.1145/3352460.3358256