pimDB: From Main-Memory DBMS to Processing-In-Memory DBMS-Engines on Intelligent Memories

Arthur Bernhardt
Data Management Lab,
Reutlingen University

Andreas Koch
Embedded Systems and Applications Group,
Technische Universität Darmstadt

Ilia Petrov
Data Management Lab,
Reutlingen University

ABSTRACT

The performance and scalability of modern data-intensive systems are limited by massive data movement of growing datasets across the whole memory hierarchy to the CPUs. Such traditional processor-centric DBMS architectures are bandwidth- and latency-bound. Processing-in-Memory (PIM) designs seek to overcome these limitations by integrating memory and processing functionality on the same chip. PIM targets near- or in-memory data processing, leveraging the greater in-situ parallelism and bandwidth.

In this paper, we introduce pimDB and provide an initial comparison of processor-centric and PIM-DBMS approaches under different aspects, such as scalability and parallelism, cache-awareness, or PIM-specific compute/bandwidth tradeoffs. The evaluation is performed end-to-end on a real PIM hardware system from UPMEM.

ACM Reference Format:
Arthur Bernhardt, Andreas Koch, and Ilia Petrov. 2023. pimDB: From Main-Memory DBMS to Processing-In-Memory DBMS-Engines on Intelligent Memories. In *19th International Workshop on Data Management on New Hardware (DaMoN '23), June 19, 2023, Seattle, WA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3592980.3595312

1 INTRODUCTION

Motivation. Modern data-intensive systems face exponentially increasing data sizes [2, 19], while workloads become both compute- and data-intensive. The low data-locality and present computer and system architecture require massive transfers of large datasets across the whole memory hierarchy. As a result, systems and algorithms become *bandwidth-* or *latency-bound*, yet bandwidth is scarce and latencies are not improving. Indeed, memory is getting colder as over the last two decades, DRAM capacity has improved 128×, bandwidth 20×, while latencies only 1.3× [12] and it takes several hundred CPU cycles to access it [20]. Performance, scalability or energy consumption and ultimately the degree to which modern many-core, main-memory systems are economical [10, 13] are bounded by *data movement* given the underlying phenomena such as the von Neumann bottleneck, Dennard scaling or the Memory Wall. Data transfers stall processing as they are performed over a slow and energy-hungry off-chip memory bus with limited width. Yet, the memory latencies are high and not improving, while the cache efficiency is low as cached data cannot be reused due to the poor data locality. Furthermore, today's computer architectures and programming models are *processor-centric* and *data-to-code*, aggravating matters as they treat memory and storage as *passive* components and mandate that data is transferred across the memory hierarchy to the CPUs, to be processed there.

Processing-In-Memory (PIM) is a general and well-known concept [4–6, 11, 14, 15, 17, 18, 24] that describes the ability to execute operations within the memory or on some processing element located near the physical memory. Thus, memory turns into an *active/computational* component, yielding a shift towards *code-to-data* paradigms and *data-centric* architectures, where operations are executed in or near the physical data location. This way, data transfers are performed in-situ, and off-chip data movement along the critical path up to the host CPU is significantly reduced. Prior approaches [5, 6, 14, 15, 17, 18, 24] got insufficient momentum due to the less advanced and less economical manufacturing processes integrating logic onto DRAM chips with sufficient density and volume, as well as the limited existence of critical applications or workloads.

Recent advances in novel semiconductor storage technologies, heterogeneous parallel processors, as well as manufacturing processes have significant potential to overcome the limitations of conventional computers for database use. Several trends support this observation. *Firstly*, the virtual memory hierarchy evolves from passive to active/computational memories on *all levels*. *Secondly*, novel memory technologies emerge, with different organisations (e.g. 2.5/3D) and interconnect to memory cells that yield significant *chip-level bandwidth that grows with chip density*. *Thirdly*, novel computational memories host PIM Compute Units (PCU), e.g., called DRAM Processing Units (DPUs) in the UPMEM architecture. Given the advances in fabrication technology, the number and the characteristics of PCUs also *improve with density* [23], yielding *massive compute parallelism*. In fact, even today UPMEM reports 8GB DIMMs with 128 DPUs [3] and systems with 2 560 DPUs.

pimDB. In this paper, we present the initial design of our PIM-memory-capable DBMS engine pimDB as a first step in exploring the design space of PIM in DBMS-settings. pimDB has been developed from scratch to target main-memory systems comprising PIM-capable memories. Our main goal is to compare processor-centric and data-centric (PIM) DBMS approaches under different aspects, such as scalability and parallelism, cache awareness in page layouts, or PIM-specific compute/bandwidth tradeoffs. The evaluation is performed in an end-to-end manner on a real PIM hardware system from UPMEM.

Our **contributions** are as follows:

DaMoN '23, June 19, 2023, Seattle, WA, USA

ACM ISBN 979-8-4007-0191-7/23/06…$15.00
https://doi.org/10.1145/3592980.3595312

- We investigate the impact of different page layouts (NSM, PAX) on PIM processing and scalability on a real system. We observe the necessity for custom PIM page layouts.
- We investigate the scale and the different levels of PIM-parallelism. Our exploration offers insights into the compute/bandwidth tradeoffs in PIM-processing and calls for compute/transfer interleaving primitives in PIM settings.
- We investigate the effect of the in-situ/PIM memory hierarchy and configurable PIM data transfers (as opposed to cacheline-sized transfers) on assumptions in cache-aware processing and data layouts.
- We investigate PIM allocation strategies, as PIM mandates data/operation co-placement and partitioning. In this context, we evaluate the kernel deployment and the PIM invocation overhead.

Organisation. We continue with a brief background on the UPMEM PIM architecture (Sect. 2) and introduce the architecture of pimDB in Sect. 3. We discuss the results of our experimental evaluation in Sect. 4 and highlight our lessons learned in Sect. 5 and draw our conclusions in Sect. 6.

2 BACKGROUND AND RELATED WORK

Next, we briefly overview exiting PIM infrastructures and approaches.

Architecture of UPMEM PIM Infrastructure. PIM technology is an emerging computing paradigm that aims to improve performance and energy efficiency by combining memory and computation in a single chip. UPMEM has developed the first commercially available, general-purpose PIM system that replaces standard DDR4 DIMMs and combines processing elements in the form of DPUs with DRAM chips to provide large amounts of bandwidth and compute while being more economical and energy-efficient than traditional compute systems [3]. A PIM DIMM module (Fig. 1) can reside in one or more memory channels. Moreover, a *PIM DIMM* module comprises two *rank*s, each of which is equipped with 8 UPMEM PIM chips. Each chip has 8 *DPUs*, which can execute instructions in parallel to memory operations. Individual DPUs have access to a 64MB DRAM bank, called *Main RAM (MRAM)*, which is also shared with the host CPU. Furthermore, each DPU has two distinct local/private memories: a fast cache-like memory, called *WRAM*, which is 64KB in size; and an *instruction memory (IRAM)* that as big as 24KB. In the current evolution of UPMEM's PIM technology (codename P21), the DPUs operate at a clock speed of 350 MHz. One of the key advantages of UPMEM's PIM technology is its high bandwidth. Each DPU can employ a DRAM-DPU bandwidth of up to 1 GB/s [3]. This is made possible by the direct access to DRAM and WRAM, eliminating the need for data movement between the memory and the CPU. Today, a fully equipped PIM system can support 20 PIM DIMMs, resulting in a total of 2560 DPUs and 160GB of MRAM.

UPMEM's PIM system currently coexists with conventional memory and is installed alongside the DIMMs in the server. A set of libraries and tools are provided by UPMEM, allowing developers to create PIM-enabled applications [22]. The MRAM is conveniently accessible through a host API, handling serial or parallel CPU-DPU and DPU-CPU transfers as well as DPU allocations and synchronous or asynchronous DPU invocations.

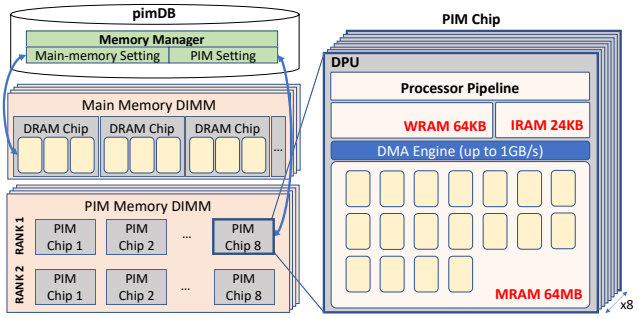

Figure 1: Processor- and PIM-centric architectural settings.

The *programming model* for UPMEM's PIM system is based on *software kernels* written in C and *tasklets*, which are software threads that can be executed in parallel by the DPU. The number of tasklets can only be set at compile time. Furthermore, there are compile-time options available to fine-tune the stack size of each tasklet, allowing for optimized WRAM usage. The DPUs WRAM is shared across all tasklets, however each tasklet operates on its own WRAM portion. Synchronization across DPU tasklets is performed by means of a set of synchronization primitives provided by the UPMEM PIM API. These include semaphores, mutexes, handshakes, and barriers.

Pre-compiled kernels can then be loaded and invoked on the desired number of allocated DPUs. DPU allocations can range from a single DPU up to utilizing all available PIM DIMMs. This adaptability enables simultaneous execution of different kernels on various DPUs. Notably, MRAM and WRAM are not shared among DPUs and no direct DPU-DPU communication mechanisms are available. These need to be managed by the host CPU.

The upcoming generation of UPMEM's PIM-enabled memory is expected to offer several key improvements. One notable improvement is the ability to access the WRAM from the host while the DPU owns the rank, providing greater flexibility on how data can be exchanged. Additionally, the DPUs are expected to operate at faster speeds of 466 MHz and its frequency can be adjusted chip-by-chip for more efficiency. Another improvement is the addition of new profiling features through different performance counters, allowing PIM application developers a more precise evaluation of their designs and potential optimizations. Finally, these new iterations are expected to be 30% to 40% more energy efficient than the current generation. These improvements will likely result in faster and more efficient data processing, making PIM hardware an even more attractive option for various applications.

Related Work. AxDIMM [9], similar to UPMEM's PIM-DIMMs, incorporates specialized hardware directly into memory modules. It combines a field-programmable gate array (FPGA) and memory into a PIM-DIMM module, allowing for the execution of customized logic operations within memory. While UPMEM offers general-purpose kernels to be run on DPUs, AxDIMM utilizes specific database acceleration (DBA) engines to offload database operations. Notably, AxDIMM can be used as regular memory interfaced via DDR protocol without requiring data to be copied from host memory to device memory first, as is the case with UPMEM's PIM technology.

Samsung HBM-PIM [16] is the first to combine high bandwidth memory (HBM) and optimized AI engines inside each memory bank, enabling parallel in-memory processing. The authors of [8] presented an experimental study that compared database SIMD operators between PIM and existing x86 processors. However, their study relied on architectural simulators for evaluation purposes.

A thorough assessment of the UPMEM PIM architecture was conducted in [7], revealing promising outcomes in various application domains. Although initial results of database operations are presented, the analysis was conducted on a simple input array without taking page layouts into account.

3 OVERVIEW OF pimDB

pimDB is a new DBMS built from scratch to target main-memory systems comprising PIM-capable memories. By exploring the design space of PIM in DBMS settings, pimDB seeks to identify concepts that are practicable for PIM-settings, as well as new PIM strategies and optimizations that can improve the performance and scalability of main-memory systems. One of the key features of pimDB is its support for multiple page formats, which allows for the evaluation of their impact on PIM processing. Currently, pimDB implements two page formats: row-based NSM for frequent transactional processing and column-based PAX for analytical workloads. Large datasets like TPC-H can be bulk loaded and transformed into the desired page layouts.

Another key feature of pimDB is a novel memory manager (Fig. 1), capable of storing pages in either traditional main-memory setting or in PIM-setting. In the traditional settings, pages are stored in pre-allocated memory chunks in the main-memory. Alternatively, pimDB can leverage UPMEM's PIM technology to store pages in PIM-enabled memory chips, utilizing the benefits of lower in-situ latencies, higher in-situ bandwidth, and efficient scalability offered by PIM processing.

Currently, pimDB supports scans, selections, projections, aggregations, and record materialization strategies for result-set handling. The operations are either hard-coded into the DB (CPU processing) or provided as parametrizable kernel functions utilizing schema embedding, in the case of PIM processing. The parameters of PIM-invocations in pimDB are transferred to all allocated DPUs during the invocation process. Typical parameter types comprise schema information, operation IDs, operation parameters, for example, how to evaluate predicates, and a page range to consider during processing.

An important property of the PIM hardware in contrast to CPU-based systems is its *dynamic configurability*, in terms of:

a) PIM-cache/ WRAM (opposed to fixed-sized L1/L2 caches),
b) DMA transfer-sizes (opposed to cacheline-sized transfers), or
c) level of parallelism (i.e., number of DPU tasklets).

To leverage dynamic configurability, pimDB allows the configuration of these properties for each operation or invocation. Firstly, compared to traditional CPU-centric system and their fixed caches, DPUs do not employ caches at all. Instead, they employ low-latency WRAM in combination with configurable DMA (Direct Memory Access) transfers, which enables flexibility in adapting to different transfer sizes, access patterns, operations, or data structures. In combination with tasklets, both of the above allow interleaving

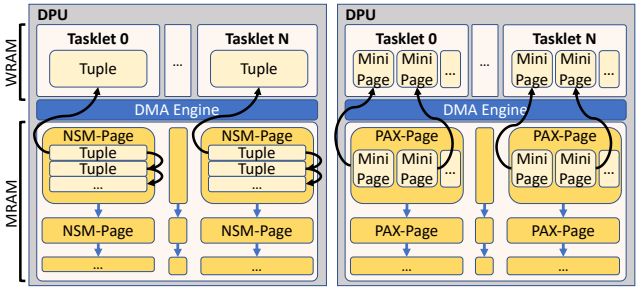

Figure 2: PIM's dynamic configurability and flexible memory hierarchy result in different ways to leverage scan parallelism. NSM scans cache portions of a tuple or multiple tuples. PAX scans cache portions of one or more mini-pages.

data processing and transfers, which becomes a crucial factor in designing efficient PIM algorithms.

Secondly, to leverage tasklet-parallelism, pimDB has two distinct implementations of the scan operation for NSM and PAX [1] (Fig. 2). To this end, the pages stored in the DPU-local memory (MRAM) are processed by a configurable number of tasklets. Each tasklet operates on its own WRAM space to load and cache data or to store temporary or intermediate results. In NSM, tasklets first load tuples from NSM pages (MRAM) to WRAM before continuing processing. The amount of data to be cached in pimDB is configurable, allowing for the caching of either partial tuples or multiple tuples simultaneously, and can be modified to suit various operations. In PAX, tasklets load and cache portions of a single mini-page or multiple mini-pages. If a predicate evaluates a single attribute, the corresponding mini-page is cached first. When additional attributes of a tuple need to be accessed, only the necessary portions of the mini-pages are loaded and cached, minimizing data movement to the DPU. The design goal is to achieve a compute/transfer balance: even though DPU bandwidth may be sufficient, DPUs may easily become compute-bound, not just due to the PIM-operation but also through I/O wait times, for unnecessary data.

With pimDB we seek to provide promising directions for research on main-memory systems and PIM technology. By exploring the design space of page formats, memory management strategies, operations and algorithms, data structures, and workloads, we plan to gain new insights into how to optimize PIM processing for optimal performance in DBMS.

4 EXPERIMENTAL EVALUATION

Experimental Setup. The experiments are conducted on a server-grade UPMEM system with two x86 CPU sockets as two NUMA nodes, equipped with Intel Xeon Silver 4110 CPUs and 125GB of DRAM. Each CPU has 8 cores and supports two threads per core, resulting in a total of 32 simultaneous threads (SMT). Furthermore, the system is equipped with 20 PIM-enabled UPMEM memory DIMMs (Codename P21) resulting in up to 2560 DPUs (128 DPUs/dual rank DIMM) and 160GB of memory (64MB per DPU). In our setup, some faulty DPUs reduced the usable computational power to a total of 2496 DPUs and the available memory to 156GB. Our evaluation relies on *TPC-H datasets* [21] with different scale factors.

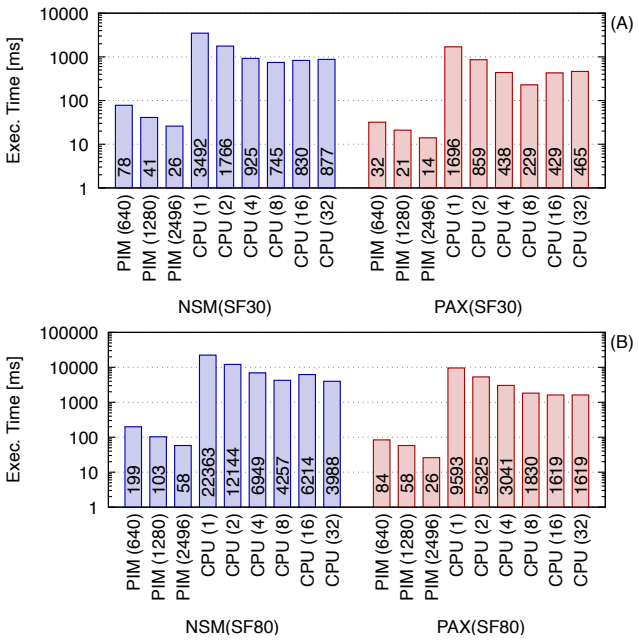

Figure 3: Performance of PIM-only and CPU-only execution in large scalings. The massive PIM parallelism and bandwidth yield significant improvements.

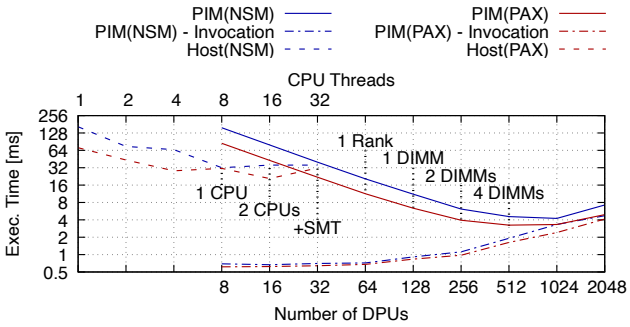

Figure 4: Scan-and-Select performance with varying levels of parallelism in CPU-only and PIM settings: the top X-axis shows CPU Threads, while the bottom X-axis depicts the number of DPUs. Minimizing data movement is crucial to utilizing PIM parallelism. PIM scales almost linearly up to 256 DPUs, beyond that invocation costs impact scalability.

Experiment 1: PIM vs. CPU. We open the evaluation with a general experiment investigating the impact of PIM scaling and parallelism. To investigate the effects, we execute a scan-and-select operation with both page layouts on PIM and the CPU, varying dataset size (SF) and increasing threads/DPUs utilized during execution. The selection predicate evaluates a single attribute (50% selectivity) on the TPC-H `orderline` table. We use 11 software threads (Tasklets) to fully utilize the DPU pipeline. For CPU execution, we implemented the use of CPU affinity bit masks to assign specific cores to individual threads, thereby ensuring thread-core pinning. Experiments with up to 8 threads are assigned to a single CPU, up to 16 threads assigned to both CPUs without SMT enabled and for 17 and more threads, SMT is enabled.

The performance differences between a PIM-only and a CPU-only execution for SF 30 and SF 80 are shown in Fig. 3. PIM provides excellent scaling up to 2496 DPUs for both page layouts with a performance improvement of up to 68× using NSM and 62× using PAX. The CPU only scales well up to 8 threads, and more threads result in smaller improvements, in some cases, even worse performance.

Additionally, to evaluate the performance of a single PIM memory chip up to multiple PIM DIMMs, we repeated the experiment for SF 1, allowing the data to fit into the memory assigned to 8 DPUs/single chip (Fig. 4). We observe that PIM scales almost linearly up to 256 DPUs. Beyond that (past 512 DPUs), the PIM/DPU invocation cost starts to impact the scalability, outweighing the data transfer savings at 2048 DPUs. However, the number of pages to process per DPU is tiny in these cases, resulting in dominating invocation costs, which involve transferring invocation parameters to all allocated DPUs in order to invoke the specific operation within the loaded kernel.

High levels of DPU and tasklet-parallelism and scalability are achievable for parallelizable DB operations. Despite NSM performing worse than PAX, it scales equally well for the given operation. The high PIM-DRAM bandwidth (1GB/s per DPU) and the combined compute capabilities allow a single PIM DIMM to outperform the CPU execution.

Insights: DBMS operations running in PIM settings can improve performance over traditional CPU-based settings. In particular this is the case for highly parallelizable and size-reducing PIM operations on partitioned datasets, due to tasklet-parallelism and PIM/DPU-scalability. Even though a single DPU is inferior to a CPU core, PIM scales more efficiently than CPU-based systems, as data is processed in-situ, reducing costly data transfers to the CPU. Additionally, large parallelizable workloads benefit from PIMs higher parallelism, as multiple operations are performed simultaneously on different parts of the data, avoiding CPU resource contention. Even though PIM/DPU invocations incur an overhead, it is very low for many DPUs (4ms with 2048DPUs) and will become negligible for larger datasets and more demanding operations.

Experiment 2: Page Formats. The in-situ memory hierarchy is an essential aspect of PIM architectures. For efficient processing, it is crucial to minimize the movement and optimize data placement across different memory levels. This in turn is essential for fully utilizing the PIM/DPU- and tasklet-parallelism.

To compare the impact on PAX and NSM, both of which have different characteristics regarding cache utilization, we perform a scan-and-select operation in PIM (64DPUs, 11 Tasklets) and the CPU (32 SMT threads) settings. The selection predicate evaluates a single attribute with a selectivity of 50%, row-wise in NSM, column-wise in PAX. We report the instructions per cycle (IPC) as a rough measure of the efficiency of the compute/transfer interleaving during DPU execution. Notably, the maximum number of instructions that can be executed per cycle per DPU is limited to 1. PAX performs as expected (Fig. 5.A), reducing costly transfers from main-memory

to the CPU and DPU WRAM, resulting in a faster execution than NSM. Interestingly, even in PIM-settings NSM is I/O-bound with IPC<0.58, while PAX is compute-bound with IPC>0.96 (Fig. 5.B).

Based on this observation, we move on to investigate how different DMA transfer sizes affect PIM performance opposed to cacheline-sized (64B) transfers, given the dynamic configurability of PIM. To evaluate the selection predicate, the attribute must be loaded from MRAM to WRAM first.

In NSM, this process leads to read amplification as unnecessary data of the tuple is transferred. To mitigate this issue, pimDB can configure DMA transfer sizes (i.e., 32B, 64B, 128B) which allow for the loading of larger chunks from MRAM to WRAM in a single DMA transfer. This effectively reduces the number of DPU cycles needed to initiate DMA transfers, and improves DPU utilization compared to multiple consecutive transfers with smaller sizes. This is natural given the DRAM latency characteristics of the MRAM. In the current experiment, where a tuple is 64B in size but only 4B of the tuple are necessary for the predicate evaluation, using 32B DMA transfer sizes demonstrates the lowest read amplification and achieves the best DPU utilization (Fig. 5.B). On the other hand, employing 64B sized DMA transfers leads to worse utilization since transferring an additional 32B of the tuple only increases read amplification and I/O wait time even given the high DPU bandwidth. Furthermore, 128B sized DMA transfers reduce the frequency of DMA invocations and enable the loading of two tuples at once into WRAM. This results in increased DPU utilization compared to 64B DMA transfers, but worse than 32B. However, minimizing data movement performs better than reducing the number of DMA invocations for the PIM scan-and-select operation.

PAX offers an inherent advantage in evaluating scan predicates on single attributes without encountering read-amplification issues. DPUs can operate on the data residing in WRAM for longer duration before needing to load new portions of the mini-page from MRAM. This reduces the frequency of data loading and enhances overall performance by enabling extended processing times without the need for frequent MRAM accesses. The impact of different DMA transfer granularities is reduced and results in compute-bound PAX. However, UPMEM expects the next iteration of PIM-enabled memory to employ DPUs with higher clock frequencies (up to 466 MHz), improving PIM compute power.

Insights: Data placement and the mitigation of I/O wait times are essential for fully utilizing the PIM/DPU- and tasklet-parallelism. Given the relatively weak DPU compute capabilities relative to the high in-situ bandwidth (1GB/s per DPU) we observe a PIM-specific compute/bandwidth tradeoff. With configurable cache-like memory (WRAM), PIM offers more flexibility than CPU-based cache-aware processing, allowing workload-tailored configurations of memory and transfer units (DMA). We expect PIM's dynamic configuration (and adaptivity) to influence the design of new PIM-tailored data structures and operations and the transition to invocation-based configurations.

Experiment 3: Partitioning and data placement are critical for harnessing the performance and scalability in PIM systems to ensure shared-nothing style PIM/DPU executions. Noticeably, the number of DPUs and the combined compute and bandwidth capabilities are proportional to the allocated (and the absolute)

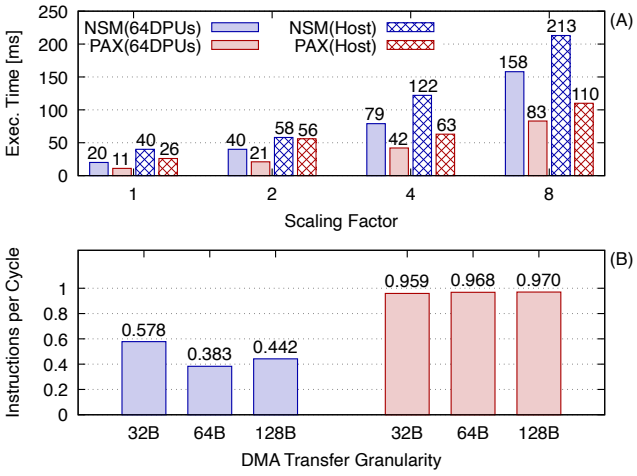

Figure 5: (A) Impact of page layouts on scan-and-select performance: PAX improves performance by reducing memory transfers in CPU and PIM settings. (B) DPU utilization: NSM is I/O-bound while PAX is compute-bound. Tuning PIM (DMA) transfer sizes improves NSM DPU utilization.

memory volume. To investigate the effect, we now execute a scan-and-select operation varying the number of DPUs and the data size (the scale factor).

When increasing the data size and the number of DPUs at the same rate (Fig. 6.A), constant performance is achievable, as the amount of data to be processed per DPU does not change. By keeping the data size fixed and distributing it evenly across an increasing number of DPUs (Fig. 6.B), the performance increases linearly as the amount of data to process per DPU decreases linearly.

Insight: With increasing data volumes, a PIM system can scale along while still providing constant performance. Furthermore, distributing the data across all available DPUs provides the highest compute and bandwidth capabilities and the best performance. However, when lowering data sizes, the PIM invocation cost may outweigh the data transfer savings, as indicated in Exp. 1, Fig. 4.

Experiment 4: Tasklet-parallelism. To fully saturate the execution pipeline of individual DPUs, it is necessary to partition the workload across multiple UPMEM *tasklets*. Tasklet-parallelism improves compute- and bandwidth-bound operations. However, data-intensive operations that saturate the PIM bandwidth will not benefit from additional tasklets. To demonstrate the impact of tasklet-parallelism, we execute a scan-and-select operation (50% selectivity) on 64 DPUs with SF 8, varying the number of tasklets and increasing the DMA transfer granularity. We also evaluate their impact on the IPC. We opt for 64 DPUs (single DPU Rank) to minimize the impact of invocation cost.

PAX can saturate the DPU at 12 or more tasklets, and increasing the DMA transfer size does not affect the overall throughput (Fig. 7). NSM can saturate the DPU at nine tasklets for 32B and six tasklets for 64B and 128B DMA transfers. Interestingly, despite increasing the bandwidth requirement of PAX with larger DMA transfers, the

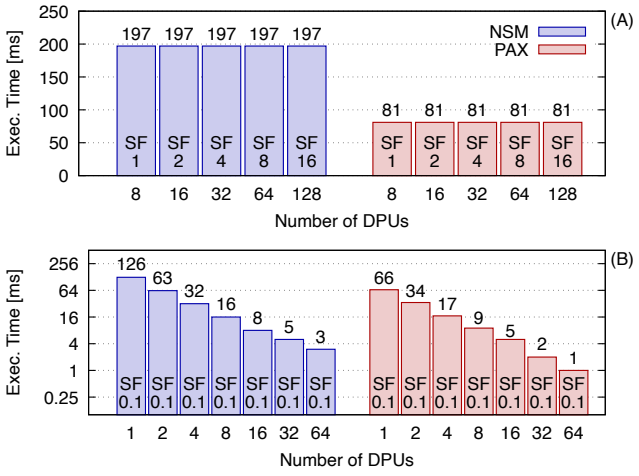

Figure 6: Impact of memory allocation and data placement on scalability: constant ratios DPUs per Unit of Volume possible, yielding constant performance of parallelizable operations with growing data sizes.

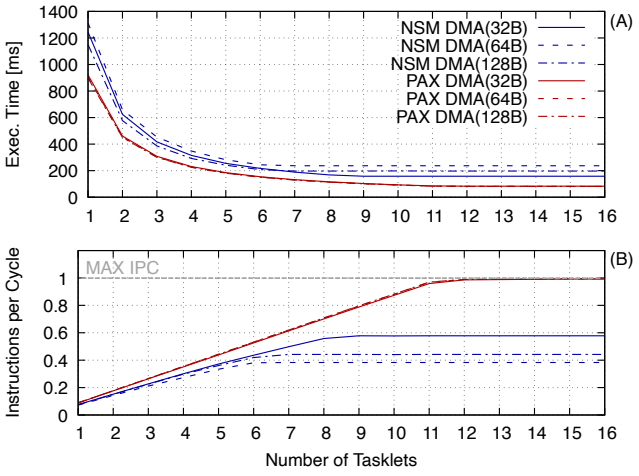

Figure 7: Tasklet-parallelism efficiently interleaves transfer times and computation with other tasklets. Increasing the number of tasklets with well-chosen DMA transfer sizes improves DPU utilization and performance.

DPU utilization remains unchanged, substantiating DPUs compute-boundness (Fig. 7.B).

Insights: Tasklet-parallelism efficiently interleaves data-transfer times and processing with other tasklets. Tasklets can interleave data movement and processing exceptionally well for compute-bound operations. Yet, they also help utilizing the available bandwidth more efficiently in data-intensive operations. However, there is a *tradeoff* between the number of tasklets and available WRAM. An overly high number of tasklets reduces the amount of available WRAM for each tasklet, as tasklets operate on their own portion of the fast but limited WRAM (64KB), which itself is shared amongst all tasklets.

Experiment 5: Projection and Aggregation. To further evaluate the capabilities of PIM, we increased the data and compute intensity by introducing an additional aggregation on top of a scan-and-select operation. We vary the intensity through the number of aggregation attributes, thereby increasing the projectivity. With 50% selectivity, we execute the aggregation on 64 DPUs with 11 tasklets. Additionally, the DMA transfers are configured to be 32B in size. Tasklet-parallelism should allow bandwidth-bound DPUs to interleave additional computing effort without significantly impacting the performance. Compute-bound operations, however, should efficiently interleave increased data movements. Our results are shown in Fig. 8.

In NSM, we cache tuples in WRAM before continuing with the predicate evaluation and aggregation. In the current setting, NSM can transfer and cache all required attributes through a single DMA transfer, which does not increase the data intensity for larger projections. Furthermore, the tasklets efficiently interleave the additional compute cost introduced by the aggregation, yielding better DPU utilization and robust performance in larger projections (Fig. 8.B). In PAX, the number of attributes to aggregate does influence the data intensity. We cache portions of the required mini-pages in advance in WRAM. However, the tasklets can effectively interleave the increased data movements, visible as steady IPC in Fig. 8.B. Yet, the aggregation performance of PAX degrades with increased projection (Fig. 8.A). PAX is already compute-bound, and the additional aggregation-costs slow down the execution further.

Insight: Interleaving data transfers and computation with tasklet-parallelism enables efficient data processing in PIM. However, balancing compute and bandwidth per operation is a challenging research question that influences the optimal design of operations. To optimize PIM applications, profiling will be essential, which UPMEM supports with host-side and DPU profiling tools.

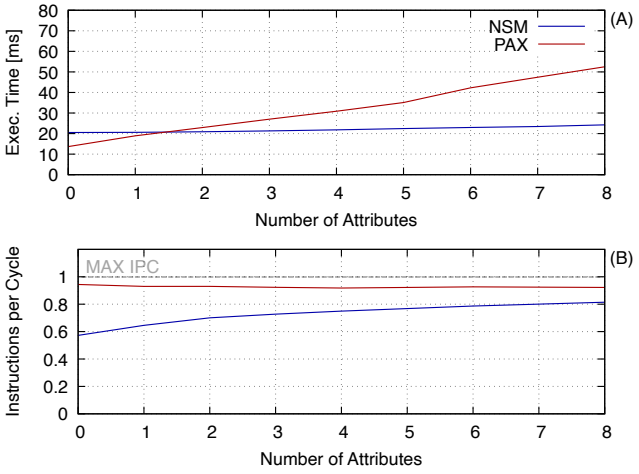

Figure 8: Performance and DPU utilization of aggregations in NSM and PAX with increasing projection.

Experiment 6: Materialization. Effective result-set handling and materialization strategies are critical to fully leveraging the advantages of PIM systems. Choosing a suitable strategy requires careful consideration of workload characteristics, query pipelining, memory capacity, compute/bandwidth trade-offs, data access patterns, and PIM-CPU transfer costs. The amount of memory available per DPU is limited and should be taken into account. Early materialization strategies can quickly consume available memory for temporary or intermediate results, which need to be reserved explicitly in advance. Late materialization can reduce memory usage by materializing only the tuple locations/positions. In our setup, we chose to create a positional bit-vector per page (one bit per tuple) and set individual bits depending on the predicate evaluation. To demonstrate the compute/bandwidth trade-off between PAX and NSM during early and late materialization, we execute a scan-and-select operation with varying selectivities (Fig. 9.A). Initially, we opt 64 DPUs to reduce the invocation overhead. However, we increase the number of DPUs, while maintaining a selectivity of 50% (Fig. 9.B), to investigate the impact on scalability. Both experiments are performed on SF 1 with 11 Tasklets per DPU and a DMA transfer size of 64B.

The additional compute cost of creating a bit vector can be efficiently interleaved by tasklet parallelization in NSM. PAX, already being compute-bound, performs worse compared to no materialization, resulting in a degraded performance of 21,7% at 10% selectivity and up to 35,9% at 100% selectivity. Both utilize WRAM to cache the positional bit-vector and reduce slow MRAM transfers. The bit-vector is flushed using a single DMA transfer once a page is fully processed.

Our full materialization approach uses pre-allocated and reserved MRAM memory to store the materialized tuples, reducing the number of pages that can be stored overall in PIM. The NSM page layout stores data in rows, making it easy to materialize records by copying the entire tuple to MRAM. To optimize performance, each tasklet allocates 64B in WRAM to read and cache an entire tuple and, if necessary, write the entire tuple in one DMA transfer. However, writing cached tuples from WRAM to MRAM for materialization increases the data movement and negatively impacts bandwidth-bound NSM, leading to decreased performance.

PAX's column-based storage requires reading from multiple locations to materialize a tuple, which increases compute and data movement costs. To mitigate this, we partially cache mini-pages in WRAM. Furthermore, PAX requires format parsers to identify attribute types and sizes for constructing a materialized record, adding to the computation cost. However, even with optimizations, a full materialization in PAX still results in significantly worse performance than NSM.

Insight: Despite the performance differences between PAX and NSM and the materialization strategies presented, scaling is linear with an increasing number of DPUs (Fig. 9.B). Again, the invocation cost has a noticeable impact on performance beyond 512 DPUs.

Experiment 7: Kernel deployment and invocation. One of the key decisions in designing database operations as PIM/DPU kernels is the choice between using large generalized kernels or small specialized kernels. Generalized kernels refer to the use of a single DPU binary than can handle a wide range of instructions and operations, while specialized kernels involve tailoring the binary to a specific task or operation. This trade-off has significant implications on the limited instruction memory (IRAM), kernel loading times, and kernel invocation, which are crucial factors in PIM systems.

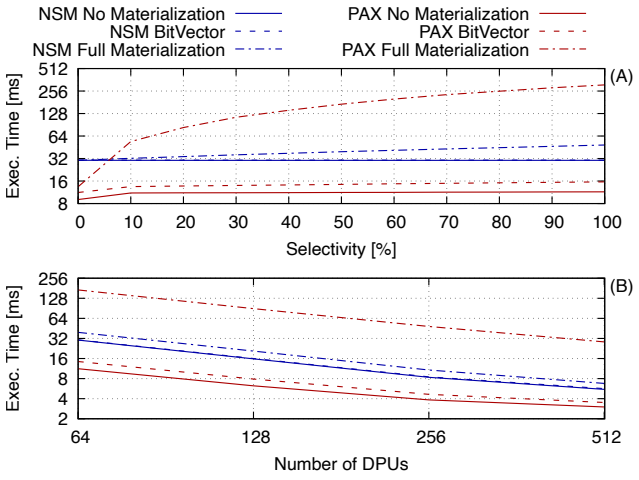

Figure 9: Impact of materialization strategies on PIM/DPU performance and scalability. Late materialization (positional bit-vector) reduces the cost of materialization. It incurs a compute overhead that is interleaved with transfers in NSM, but visible in PAX, which is compute bound.

Limited instruction memory capacity poses a challenge in PIM architectures, as it restricts the amount of code that can be stored and executed within a kernel. Large generalized kernels offer the advantage of versatility, as they can execute multiple task without the need for binary swapping. This approach simplifies the programming model and reduces the overhead associated with kernel management. However, they introduce a more complex invocation process. Since the kernel needs to handle various operations, additional parameters or context information are required for proper execution. This increases the data transfers during kernel invocation. On the other hand, specialized binaries archive higher code density and are tailored to specific tasks or applications, resulting in reduced instruction memory requirements. The invocation process is simplified, as the system only needs to provide the necessary inputs, without the need for additional configuration parameters.

Another factor to consider are the binary loading times, which incur additional overhead during loading or swapping. Large generalized kernels require a one-time loading process, reducing the frequency of binary transfers. This can be advantageous in scenarios where swapping is impractical or time-consuming, especially in situations where latency is a critical concern. Specialized kernels, on the other hand, may require more frequent binary loading, as each task may have its own dedicated kernel. However, the smaller size of specialized binaries reduces the loading time.

To show the trade-off between large generalized and specialized kernels in terms of binary loading times and the invocation cost, we measured the transfer time of the DPU kernel binary and the invocation parameters with an increasing number of allocated DPUs (Fig. 10.A). Moreover, we compare the scan-and-select kernel

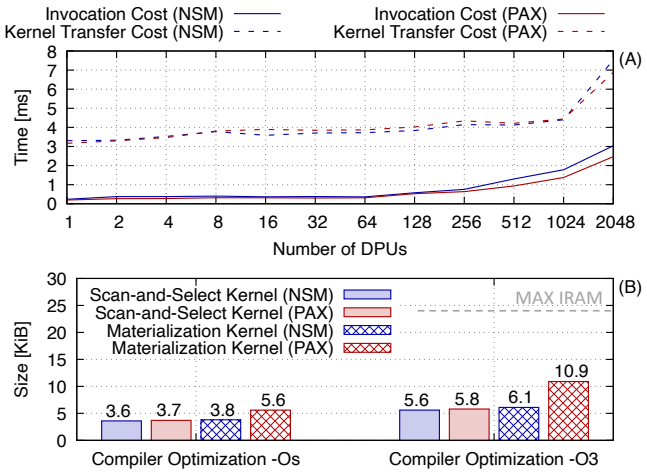

Figure 10: Trade-off between large generalized and specialized kernels in terms of loading times and the invocation cost. (A) Transfer costs of invocation parameters and kernel varying the number of DPUs. (B) IRAM utilization of kernels.

against the larger and more complex materialization kernel to show the instruction memory (IRAM) utilization (Fig. 10.B).

The cost of transferring invocation parameters in UPMEMs PIM architecture is remarkably low and ranges from 200us for a single DPU up to 3ms for 2048 DPUs. Binary transfer time ranges from 3.2ms for 1 DPU up to 7.5ms for 2048 DPUs, indicating that kernel swapping can be performed within reasonable duration. Compiler size optimization results in compact binaries with reduced instruction memory requirements, with 3.8KiB for NSM and 5.6KiB for PAX, for our largest kernel supporting materialization. The provision of 24KiB of IRAM in each DPU offers ample space for implementing more complex as well as more generalized kernels.
Insight: Our findings highlight the importance of adaptive kernel selection for PIM DBMS. By dynamically choosing between larger generalized kernels and specialized kernels based on the characteristics of the workload, a balance between versatility and efficiency is achievable. With pimDB, we plan to explore and leverage this flexibility to adapt to varying computational requirements and achieve optimal performance in a range of scenarios.

5 LESSONS LEARNED

In the present work, we provide an initial investigation of PIM for database use on real hardware. UPMEM is one of the first real systems available in practice, and one that is feature-rich. In general, there are many open questions regarding the hardware architecture of PIM systems. How will PIM-capable memories be embedded into the memory hierarchy? Will they potentially co-exist with passive memories building a multi-tier main-memories or is it realistic to assume PIM-only memory? How will PIM-memories scale and what interconnect will they have, especially with view of novel cache-coherent interconnects such as CXL and protocols like CXL.mem?

(1) In this initial study, we observe novel aspects of PIM. On the one hand, we observe the *massive parallelism* that such systems bring along. On the other hand, we observe that PIM systems bring different and potentially heterogeneous hardware that can be *dynamically configured* potentially for each invocation. We expect that both properties will have a significant impact on data structure designs. For example, consider the impact of transfer units. This paper shows that PIM DMA sizes are dynamically configurable (instead of 64B cacheline-sized) and that this impacts the PIM operation, the parallelism and materialization strategies.
(2) The massive PIM parallelism (DPU and tasklet) allows DBMS operations running in PIM settings to outperform traditional CPU-based settings for well-parallelizable and size-reducing operations on partitioned datasets. Additionally, large parallelizable workloads benefit from PIMs higher parallelism, as multiple operations are performed simultaneously on different parts of the data, avoiding CPU resource contention.
(3) PIM systems demonstrate efficient scalability to accommodate evergrowing datasets while maintaining constant performance for parallelizable and size-reducing operations.
(4) Data placement and mitigating I/O wait times are essential for fully utilizing the PIM/DPU- and tasklet-parallelism. Given the relatively weak DPU compute capabilities relative to the high in-situ bandwidth, we observe a PIM-specific compute/bandwidth tradeoff. However, balancing compute and bandwidth per operation is a challenging research question that influences the optimal design of operations.
(5) With configurable cache-like WRAM, PIM offers more flexibility than CPU-based cache-aware processing, allowing workload-tailored configurations of memory and DMA transfer units. We expect PIM's *dynamic configuration* to influence the design of new PIM-tailored data structures and operations and the transition to invocation-based configurations.
(6) *Tasklet-parallelism* efficiently interleaves data-transfer times and processing with other tasklets. However, there is a tradeoff between the number of tasklets and available WRAM.
(7) UPMEM's PIM architecture is well-suited for different execution modes, given the efficient kernel swapping and invocation performance as well as ample instruction memory.

6 CONCLUSIONS

In this paper, we introduce pimDB, a PIM-capable main-memory DBMS. We compare processor-centric to PIM-style basic data processing operations. PIM offers very early advantages over the host-only executions. PIM does not only scale well with the number of DPUs, it also allows configuring the DRAM transfer sizes and WRAM usage per invocation, thus optimizing for different data structures and operation types. PIM invocations may execute in a shared-noting manner across the DPUs and therefore their efficiency depends on how well they parallelize.

ACKNOWLEDGMENTS

The authors wish to thank the anonymous reviewers for the insightful comments, which significantly improved the quality of the paper. We are grateful to UPMEM for fruitful cooperation, valuable input, and access to the UPMEM hardware. This work has been partially supported by *DFG Grant neoDBMS – 419942270.*

REFERENCES

[1] Anastassia Ailamaki, David J. DeWitt, Mark D. Hill, and Marios Skounakis. 2001. Weaving relations for cache performance. *VLDB 2001 - Proc. 27th Int. Conf. Very Large Data Bases* (2001), 169–180.

[2] Cisco Corp. [n. d.]. Cisco Global Cloud Index: Forecast and Methodology, 2016-21.

[3] Fabrice Devaux. 2019. The true Processing In Memory accelerator. In *IEEE Hot Chips*.

[4] Jeff Draper, Chang Woo Kang, Ihn Kim, Gokhan Daglikoca, Jacqueline Chame, Mary Hall, Craig Steele, Tim Barrett, Jeff LaCoss, John Granacki, Jaewook Shin, and Chun Chen. 2002. The architecture of the DIVA processing-in-memory chip. In *Proc. 16th Int. Conf. Supercomput. - ICS '02*. ACM Press, New York, New York, USA, 14.

[5] Duncan Elliott et al. 1999. Computational RAM: Implementing Processors in Memory. *IEEE Des. Test 1999* (1999).

[6] Maya Gokhale, Bill Holmes, and Ken Iobst. 1995. Processing in memory: the Terasys massively parallel PIM array. *Computer (Long. Beach. Calif)*. 28, 4 (apr 1995), 23–31.

[7] 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). https://doi.org/10.1109/ACCESS.2022.3174101

[8] Tiago R. Kepe, Eduardo C. de Almeida, and Marco A. Z. Alves. 2019. Database Processing-in-Memory: An Experimental Study. *VLDB Endowment* 13, 3 (2019). https://doi.org/10.14778/3368289.3368298

[9] Donghun Lee, Jinin So, MINSEON AHN, Jong-Geon Lee, Jungmin Kim, Jeonghyeon Cho, Oliver Rebholz, 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 *Data Management on New Hardware* (Philadelphia, PA, USA) *(DaMoN'22)*. Article 2, 9 pages. https://doi.org/10.1145/3533737.3535093

[10] D. Lomet. 2018. Cost/Performance in Modern Data Stores. In *Proc. DAMON'18*.

[11] K. Mai et al. 2000. Smart Memories: a modular reconfigurable architecture. In *Proc. SCA*.

[12] Onur Mutlu. 2018. Processing data where it makes sense in modern computing systems. In *MECO'18*.

[13] T. Neumann et al. 2020. Umbra: A Disk-Based System with In-Memory Performance. In *CIDR*.

[14] M. Oskin et al. 1998. Active Pages: a computation model for intelligent memory. In *Proc. ISCA*.

[15] David Patterson, Thomas Anderson, and et al. 1997. A Case for Intelligent RAM. *IEEE Micro* 17, 2 (1997), 11 pages.

[16] SAMSUNG. [n. d.]. Memory redesigned to advance AI. https://semiconductor.samsung.com/insights/technology/pim/.

[17] D. Shaw et al. 1981. The NON-VON Database Machine: A Brief Overview. *IEEE DE.Bull.* 4, 2 (1981), 41–52.

[18] Harold S. Stone. 1970. A Logic-in-Memory Computer. *IEEE Trans. Comput.* 19, 1 (Jan. 1970), 73–78.

[19] Alexander Szalay and Jim Gray. 2006. 2020 Computing: Science in an exponential world. *Nature* (2006).

[20] Jens Teubner and Louis Woods. 2013. *Data Processing on FPGAs*. Morgan & Claypool Publishers.

[21] TPC-H. 2011. dbgen. https://github.com/electrum/tpch-dbgen.

[22] UPMEM. [n. d.]. UPMEM User Manual. Version 2023.1.0. https://sdk.upmem.com/2023.1.0/.

[23] H.-S. Philip Wong. 2019. What Will the Next Node Offer Us?. In *2019 IEEE Hot Chips*.

[24] Yi Kang, Wei Huang, Seung-Moon Yoo, Diana Keen, Zhenzhou Ge, Vinh Lam, Pratap Pattnaik, and J. Torrellas. 1999. FlexRAM: toward an advanced intelligent memory system. In *Proc. 1999 IEEE Int. Conf. Comput. Des. VLSI Comput. Process. (Cat. No.99CB37040)*. IEEE Comput. Soc, 192–201.