# Database Processing-in-Memroy: A Vision*

Tiago R. Kepe[1,2][0000−0001−9744−0034], Eduardo C. Almeida[1][0000−0001−5666−0815], Marco A. Z. Alves[1][2222−−3333−4444−5555], and Jorge A. Meira[3][2222−−3333−4444−5555]

[1] Federal University of Paraná, Curitiba, PR, Brazil
{trkepe,mazalves,eduardo}@inf.ufpr.br
[2] Federal Institute of Paraná, Curitiba, PR, Brazil
[3] University of Luxembourg
jorge.meira@uni.lu

**Abstract.** The recent trend of Processing-in-Memory (PIM) promises to tackle the memory and energy wall problems lurking in the data movement around the memory hierarchy, like in data analysis applications. In this paper, we present our vision on how database systems can embrace PIM in query processing. We share with the community an empirical analysis of the pros/cons of PIM in three main query operators to discuss our vision. We also present promising results of our ongoing work to build a PIM-aware query scheduler that improved query execution in almost 3× and reduced energy consumption in at least 25%. We complete our discussion with challenges and opportunities to foster research impulses in the co-design of Database-PIM.

**Keywords:** Processing-in-Memory · Query Processing · Query Scheduler. · Energy Efficiency

## 1 Introduction

PIM is a hardware architecture with simple processing units attached to the memory chip to efficiently use the internal memory bandwidth. PIM was originally thought at the end of the 60's [23, 18, 28]. Over the years, PIM followed the progress of memory technology with processing components installed in magnetic disks to run particular database algorithms [9] that later evolved to Smart Disks [19, 1] with embedded logical components. Intelligent RAM [25] also tried to add logical units inside the DRAM to support specific computations. Unfortunately, commercial products did not adopt those approaches due to limitations of the hardware technology and the continuous growth in CPU performance complied to the Moores Law and Dennard scaling. In the 2010s flash disks also tried to add internal functional units to run database applications [29, 10, 7]. However, they missed a general programming interface and suffered from the low abstraction level when handling hardware errors.

* This work was partially supported by the Serrapilheira Institute (grant number Serra-1709-16621).

With the increasing growth in CPU performance, the memory access became the bottleneck for many applications, a problem known as the "memory wall" [4]. In the last years, the assumptions of Moore and Dennard came to an end due to hardware limitations and the emergence of multi-core, but the memory wall remains an open issue as well the current "energy wall" [11, 33]. In data-centric systems, both walls are critical as huge amounts of data move around the memory hierarchy from disks, main memory, and caches to the CPU.

**Why PIM Now and Again?** Recently, PIM architectures came back to the spotlight due to the introduction of Through-Silicon Vias (TSVs) that made 3D integration technologies feasible: the integration of DRAM dies and logic cells in the same chip area. Furthermore, co-processing mechanisms have been proposed for GPU, CPU and PIM [26, 20] and commercial GPUs already embed the emerging 3D-stacked memories, such as the Hybrid Memory Cube (HMC) [16] and the High Bandwidth Memory (HBM) [21]. These smart memories invert the traditional data processing approach: they move computation to where the data resides.

There is no surprise that PIM poses as an attractive approach to reduce the data movement in data-centric systems. Therefore, in our vision, it is time to discuss *how Database Management System (DBMS) can embrace PIM in query processing.* The major benefits to be explored in the Database-PIM co-design are the drastic reduction in energy consumption and the internal high memory bandwidth due to the high levels of data access parallelism and the on-chip processing. PIM has been used to accelerate isolated database operators: select [27, 31] and join [24]. But, our recent work demonstrated [20, 30] the trade-offs of PIM in query processing. The behavior of each query operator depends on the dataset characteristics and the system cache settings. This position paper aims to disclose the key requirements and insights for Database-PIM. Our main contributions are:

**I. PIM-aware Query Processing.** We make a case for designing a PIM-aware Query Processing engine that coordinates intra-query parallelism between CPU and PIM. We reinforce the idea of moving computation around instead of just deciding over the data movement, discussing: *When is it cheaper to process a record in main memory rather than moving around the memory hierarchy?*

**II. PIM-aware Scheduler.** We introduce a design of PIM-aware Scheduler [20] at operator granularity to interleave intra-query processing between CPU and PIM. We discuss the research question: *How the DBMS should coordinate intra-query execution between CPU and the memory processor to exploit the potential gains from each device?*

**III. Findings and Results.** We share our findings about the impact of PIM on traditional query processing. We discuss the energy-saving and drastic reduction in response time that goes more than 1 order of magnitude for certain query operators.

**IV. Challenges and Opportunities.** We share a list of challenges and opportunities for the co-design and integration of a Database-PIM.

Next, we introduce an overview of current PIM architectures (Section 2) and the impact of PIM on traditional query processing (Section 3). We then describe our current work for Database-PIM (Section 4) and present challenges and opportunities (Section 5). We conclude with a discussion of related work and our findings (Sections 6 and 7).

## 2 PIM Architecture

Emerging 3D-stacked memories increase memory bandwidth by integrating one logic die with a stack of four or eight DDR-3 dies bonded by the TSV. Typically, the DDR-3 module consists of memory rows of 8-KB, which means that the memory controller must activate an entire 8-KB DRAM row to access any memory portion, even smaller portions than 8-KB, which limits the memory bandwidth and increases energy consumption. Therefore, the 3D-stacked memories split the 8-KB DRAM rows into small rows of 256-bytes. They organize the stack of DRAM dies vertically, where underlying memory banks of different dies are interconnected by the TSV to the logic die at the base, forming a vertical partition called as a "vault" (see Figure 1). Current 3D-stacked memories have 32 interdependent vaults to access and process data in parallel leveraging the internal high bandwidth memory up to 320 GB/s. The logic layer of 3D-stacked memories supports arithmetic, logical and bitwise atomic instructions up to 16 bytes size.

To fully benefit from the 256-bytes vault's row buffer, a recent work [2] extended the logic layer to operate with vectorized instructions of 256-bytes, enabling *Single Instruction, Multiple Data* (SIMD) with PIM. Figure 1 shows the described PIM architecture with SIMD support to the right side and the traditional von Neumann architecture to the left: the processor core and a detached cache hierarchy. At the top of Figure 1, the processor dispatches PIM instructions directly to the PIM device bypassing the cache hierarchy. At the end of the instructions, the PIM device only returns the instructions status to the CPU to continue the pipeline. This is a main advantage compared to current DBMS, such as Netezza and Exasol, that also filter data in hardware before passing to the CPU. They have to deal with packing qualifying tuples into condensed pages to avoid unnecessary bufferpool pollution, which is expensive and error prone. On the other hand, current memory protocols support all the idiosyncrasies of PIM instructions, such as cache coherence, Error-Correcting Code Memory (ECC) and Direct Memory Access (DMA). The execution flow works at instruction-granularity as the traditional CPU processing (e.g., AVX/SSE): programmers insert intrinsics PIM instructions into the code, like Intel Intrinsics, and the compiler flags them as special memory PIM instructions.

## 3 Query Processing with PIM

The first step towards a PIM-aware query processing is understanding *the impact of PIM on traditional query processing*. We discuss the impact on the execution

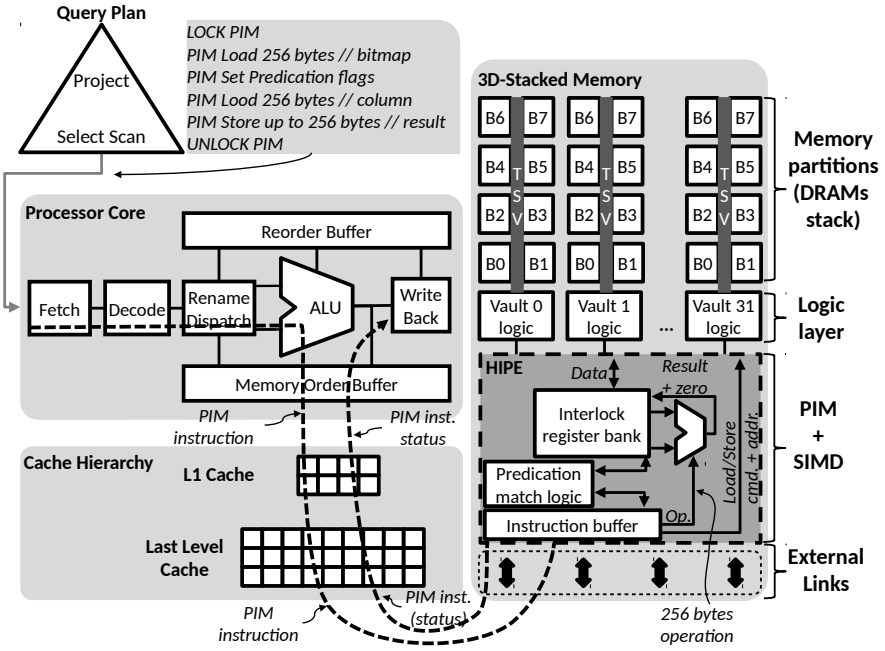



Fig. 1: A query datapath movement in a traditional von Neumann architecture plus a modern 3D-stacked memory with PIM and SIMD support.

time and energy consumption of the operators: projection, selection and join when switching the processing unit from the CPU to PIM. Since current PIM hardware do not yet implement all the extensions depicted in Figure 1, we performed the analysis through the SiNUCA cycle-accurate simulator [3] with the same parameters used by related work [31]. The energy estimations consider the DRAM parameters for state-of-the-art PIM devices [16]. The SiNUCA is extensively adopted by scientific articles in computer architecture [31, 27], HPCC [3] and database [30, 20].

We evaluate those operators using the 1 GB TPC-H database workload because their input data set fit into the caches (the best scenario for the CPU processing). It is therefore possible to investigate the data reuse behavior of such operators. The CPU implementations of the query operators use SIMD instructions of 64-bytes (i.e., Intel AVX-512) and the PIM versions use SIMD instructions of 256-bytes (see [30, 20] for details). The results presented here guide us to exploit our Database-PIM co-design.

**Projection**: Figure 2 shows that the PIM-aware projection operator reduced the execution time by more than one order of magnitude, 60× for projection and 23× for projection-path. We also observe reductions in energy consumption around 36× for both projection primitives. **Selection**: The PIM-aware selection operator reduces the energy consumption by around 98% and reduces the execution time by 76× compared to the best CPU scenario.

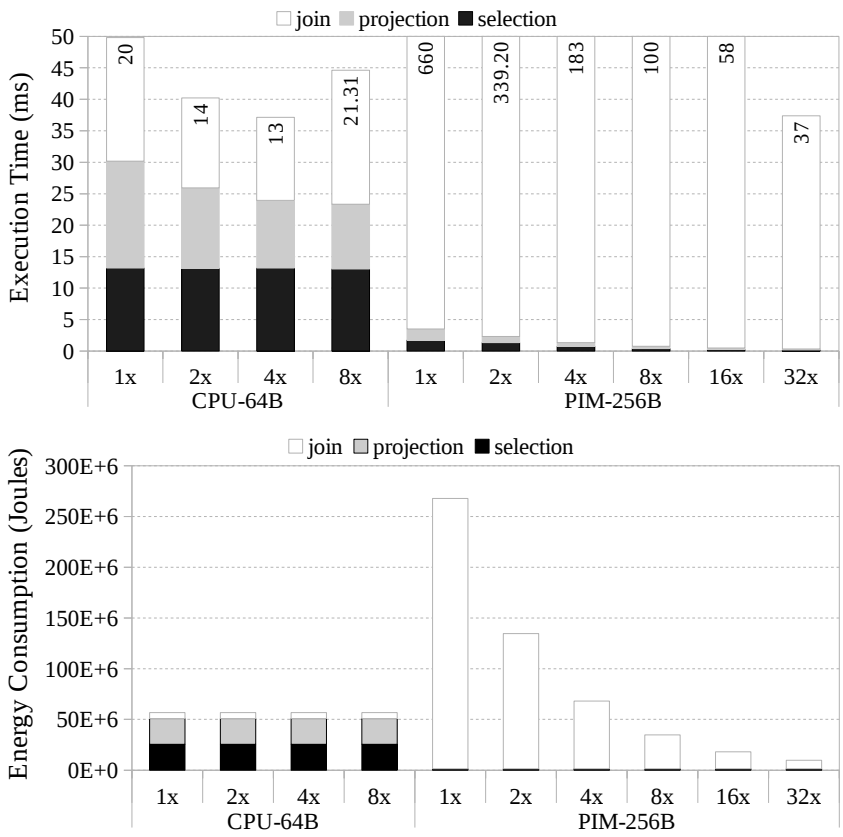



Fig. 2: The execution time and energy consumption breakdown of the TPC-H Query 03 on the operators: selection, projection and join in the CPU-64B and PIM-256B with different unrolling depths.

The streaming behavior of the projection and selection operators results in low data reuse and less amount of off-chip data transfers when running into PIM. In a conventional CPU processing, the column under processing is streamed from main memory across the memory hierarchy to the CPU registers, then the CPU tests column chunks against the selection and projection predicates. However, all the column chunks are "dead" on arrival in the cache hierarchy, because the operators do not reuse any chunk during the execution. On the other hand, the counterpart PIM operators send the computation from CPU to the PIM, both operators access all the 32 vaults in parallel and evaluate the columns with 32× PIM+SIMD instructions of 256-bytes. These results endorse the feasibility of both operators for PIM. However, small enough tables may fit into the cache, and the PIM device might lead to performance loss. Here, the DBMS must take a careful decision between CPU and PIM.

**Join**: In our experiments, we implemented the tree main join algorithms, but the Hash Join and Sort-Merge Join algorithms generate random memory accesses that inhibit the internal data access parallelism and the high bandwidth. To give an intuition of the problems of join in PIM, we analyze the execution of the Nested Loop Join (NLJ). The NLJ-PIM unrolls the inner loop up to 32× to exploit the data access parallelism of the PIM device. Every inner loop iteration causes compulsory *load* and *store* instructions, i.e., the inner column is re-accessed every time in the inner loop. Such an effect is more evident in our tests because the inner column fits into the data caches that have lower memory latency than the DDR3 dies. Thus, the CPU processing outperforms the execution of the NLJ-PIM, as depicted in Figure 2. The best CPU processing is 2.8× faster than PIM. Even the energy consumption, a major benefit of PIM, is 70% worst in the best PIM execution. However, the NLJ-PIM becomes appealing as long as the inner column does not fit into the LLC inhibiting data reuse. In this point, PIM shows an improvement of 1.38× against the CPU. Here, we observe opportunities to tackle random memory access of other join algorithms.

## 4 PIM-Aware Query Processing

Although the emerging PIM architectures stand as high performance memory technology removing part of the memory wall, the impact of PIM on query processing arises the issue: *Not every (instance of) database operator benefits from PIM*. The choice of the target architecture to process a query operator is not trivial. Indeed, we observed that query operators with high data reuse benefit from the caching mechanism and thus the CPU processing becomes appealing, such as for the NLJ. On the other hand, operators that perform data streaming (e.g., projections and selections) are best fit for PIM [27]. Other operators, such as aggregation, may have low or medium data reuse and partial streaming behavior, making it unclear where is the best fit device to process.

### 4.1 PIM-Aware Scheduler

Our investigation focus on how to interleave intra-query execution between the CPU and PIM. To the best of our knowledge, this is the first effort in that direction. Existing solutions usually direct sole data-intensive operators for PIM [24, 30, 31]. However, they neglect the potential of CPU processing boosted by caching mechanism for workloads with high temporal and spatial data locality. Thus, our insight is that *a database system requires a PIM-aware scheduler for intra-query processing*.

The DBMS scheduler is a critical performance component in query processing: it orchestrates the execution of physical primitives from the query plans. Thus, we design two scheduling strategies for a PIM-aware scheduler: static profile based scheduling and dynamic profile based scheduling. Inspired by related work, we discuss two potential strategies to receive as input the optimal plan generated by the query optimizer and coordinate the intra-query execution between PIM and the traditional CPU.

**Static Scheduling.** The static scheduling uses a classification model based on operator profiles to decide which architecture to process a given operator. Different from related work on operator scheduling for GPU, this strategy does not require a calibration step [5] nor heuristics [17]. We refer to a partial execution of the TPC-H Query 03 to evaluate our scheduling:

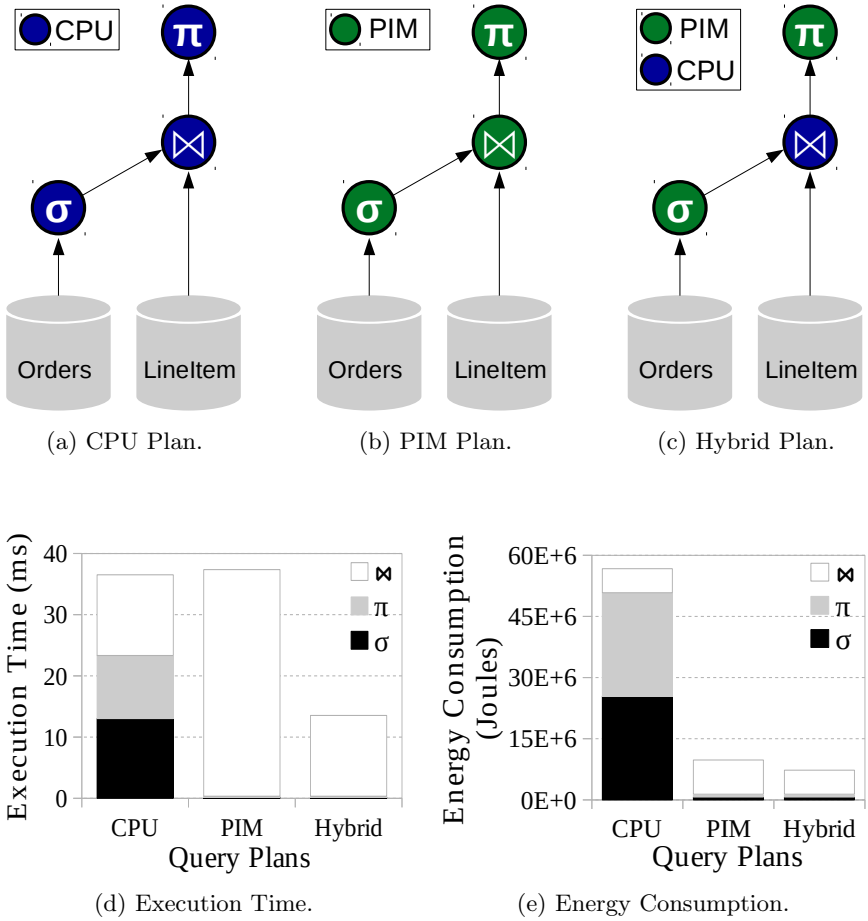
```
SELECT o_custkey FROM Orders, Lineitem
WHERE o_orderdate < '1995-03-15' AND
      l_orderkey = o_orderkey
```

Figure 3 depicts three execution plans from the optimal plan to answer that query: two hardware-specific query plans, i.e., either CPU or PIM, and a hybrid query plan based on the static profile scheduling. In the hybrid query plan, the selection and projection operators run into the PIM hardware, while the join operator executes in the CPU.

Figure 3d shows that the hybrid query plan outperforms the hardware-specific query plans in almost 3× exploiting the advantages of both architectures: the operators with high data reuse run in CPU, and the operators with data streaming run into PIM. Also, the hybrid plan reduces the energy consumption in at least 25%.

**Dynamic Scheduling.** The dynamic scheduling is our ongoing work to coordinate, automatically, the intra-query execution between CPU and PIM. We envision the dynamic scheduler regarding two criteria: execution time and energy consumption.

We present a prototype of the dynamic scheduling in Figure 4: (1) Initially, it assumes a target architecture based on the PIM-aware algorithms from each operator of the query plan; (2) The scheduler requests the observed operator profiles to the DB Profile that returns the stored profiles generated according to some criteria on database load monitoring (e.g., energy consumption); (3) The scheduler chooses the most prominent profile and (4) decides the target architecture and algorithm. (5) After the execution, the scheduler stores the observed workload information based on the chosen criteria to learning base. At its heart, the learning model decides the target architecture for an incoming operator from a query plan. The decision model shall select the most prominent algorithm based on the defined criteria from a set of profile operators. A profile operator consists of an execution algorithm (e.g., NLJ), the target architecture (e.g., CPU, PIM), the measured criteria (e.g., energy consumption) and the features about the dataset (e.g., size, distribution, cardinality, and type). We bind the workload information $W$ to any profile, such as data reuse and selectivity, and use in statistical methods to interpolate future profiles. The $W$ is useful to measure the quality of the chosen profile. For instance, the PIM-aware scheduler measures the distance between the new $W$ and the chosen profile (e.g., Dynamic time warping). We call this distance as "Energy ramp" when using the energy criteria. Also, the $W$ might be used to insert, remove or update profiles.




(a) CPU Plan. (b) PIM Plan. (c) Hybrid Plan.

(d) Execution Time. (e) Energy Consumption.

Fig. 3: Three query execution plans: a sole CPU (**a**) and PIM (**b**) plans, a hybrid plan (**c**) with CPU/PIM and the respective execution time (**d**) and energy consumption (**e**).

Figure 5 exemplifies the energy consumption of expected and observed profiles for the NLJ. As the data sets do not fit into the caches (L1, L2, and LLC), then the data reuse distance increases, which leads to increasing data movement across cache levels and more energy consumption. When the dynamic scheduling detects an increasing energy ramp, a scheduling decision is made to switch the processing unit of the running operator based on the trend profile. In the example, we use the least squares method to set the energy trend profile. If the energy trend goes up and data do not fit in cache, then the operator is rescheduled to the PIM device. In our ongoing work, we also study the tradeoffs of restarting the running operator from scratch after rescheduling(e.g., cache invalidation).

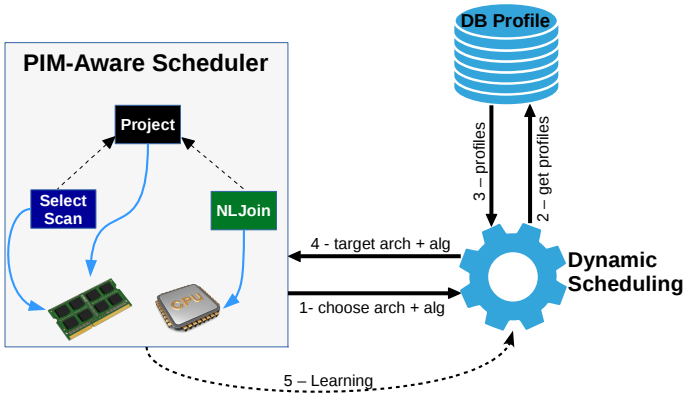



Fig. 4: Architecture overview of the PIM-aware Scheduler with dynamic scheduling.

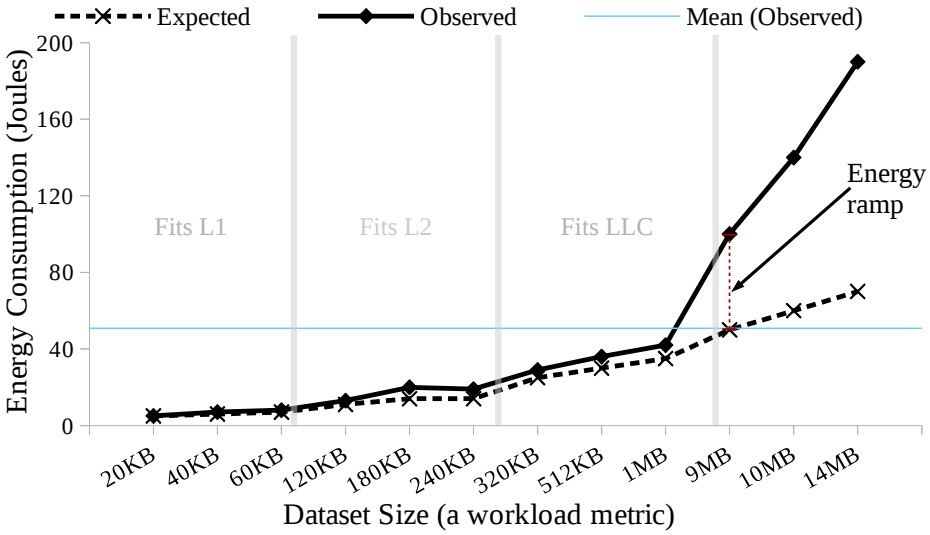



Fig. 5: The dynamic scheduler monitors the energy consumption (Joules) during the data movement (Bytes) of the NLJ for Query 03.

## 5 Challenges & Opportunities

Hybrid query processing and optimization require a holistic view of a query optimizer to exploit heterogeneous co-processing. Next, we identify a list of challenges for the Database-PIM co-design.

**Simultaneous Co-processing**: In heterogeneous processing environments, the optimizer needs to identify opportunities of co-processing to avoid idle devices or inefficient power-consumption (e.g., the power wall problem [33]). Be-

sides, the simultaneous CPU and PIM processing may add hard-to-predict concurrency into the main memory.

**Query Plan Optimization**: The search space to optimize query plans is already a problem for traditional query optimizer. The addition of hybrid query plans increases the complexity to generate efficient candidate query plans. A PIM-aware query optimizer needs to take into account hardware-specific features to choose the appropriate running device, such as limited processing power and reduced data movement for PIM, and fast processing with caching mechanisms for superscalar CPUs.

**Multi-Objective Optimization**: Due to the peculiarities of heterogeneous processing devices, the query plan optimizer must focus on multi-objective optimization (e.g., min energy s.t. runtime constraint), which is challenging to linearize.

**Transactions**: Intrinsically, arithmetic and logical PIM update instructions are atomic [14]. This opens research opportunity for near-data transactions and Hybrid Transactional/Analytical Processing (HTAP). The current PIM ISA supports compare-and-swap instruction to evaluate values. Therefore, PIM update instructions can be synchronized in-memory without wasting cache-check time or extra memory bandwidth. The opportunity is to reduce the overhead of locking and latching, which correspond to 30% of the instructions in OLTP [13].

**Programming Interface:** The programming interface of PIM architectures is still premature to implement database operations. A preliminary effort is the Intrinsics-PIM [8] that provides Intel-like intrinsic functions to simulate PIM instructions. However, the Intrinsics-PIM only provides functions for basic PIM instructions, and it does not yet support complex components (e.g., large registers) and instructions (e.g., lock and unlock) of other PIM architectures (HIVE [2], RVU [27] or HIPE [31]).

**DBMS Adoption:** We envision that DBMSes should invoke PIM instructions at the operator code base, similarly to the SSE and AVX approaches. We also consider code optimization to provide intrinsic functions for PIM ISA.

## 6 Related Work

To our knowledge, we are the first work to investigate query scheduling between CPU and modern PIM architectures aiming energy-saving.

**Flash Disks**. Recently, the attention moved to flash disks to accelerate [12] and save energy [22] of scan and join operators. However, there are two main problems in these works: (1) They rely on complex and database dedicated hardware that may reduce the general use for the hardware. Smart SSDs [10] use an embedded ARM processor into the SSD with a firmware for communication to evaluate the execution of database operators. Intelligent SSDs [7] add a reconfigurable stream processor to reach high processing performance with energy savings. (2) They are application-driven without a general interface to abstract hardware features. Active Flash [29] offloads particular functions of scientific

workloads to run into the SSDs. The Samsung Smart SSD prototype is originally designed to process the intersection of lists [32].

**PIM As Query Accelerator**. Recent works use PIM devices as isolated accelerators to boost query operators, such as selection [27, 31], projection [30], and join [24]. However, this one-sided approach neglects the potential of CPU-PIM co-processing with caching and energy-saving benefits.

**Scheduling On Emerging Hardwares**. Current intra-query scheduling focused on co-processing between GPU and CPU to improve execution time based on runtime learning model [5] and operator cost model [17]. In the Intel Xeon Phi co-processor [6] was also tested a similar scheduling idea. These approaches tackle compute-intensive applications but neglect the potential of PIM to run data-intensive applications.

**Kernel Scheduling on PIM-Assisted GPU**. Related work in GPU architectures proposed scheduling techniques with PIM devices installed as GPU main memory. GPU applications are split into independent GPU-kernels and interleave the processing of each kernel between the GPU cores and the PIM device [26, 15]. Although GPUs are highly parallel devices to boost processing power, data still need to be transferred around the memory hierarchy before moving to the GPU-PIM device.

## 7 Conclusion & Future Work

This paper presented our Database-PIM co-design vision to exploit the unprecedented memory bandwidth with on-chip processing delivered by modern PIM hardware. We discussed promising results of interleaving the parallel execution of intra-query processing between PIM and CPU. With our static scheduler, the hybrid query plans outperformed hardware-specific plans in almost 3× and reduced energy consumption about 25%. Our ongoing work focuses on the dynamic scheduling strategy to create and update operator profiles and reschedule operators on-the-fly between the CPU and PIM devices. Finally, we share with the community a list of challenges and opportunities opened by our vision in the co-design of DBMS and PIM.

## References

1. Acharya, A., Uysal, M., Saltz, J.H.: Active disks: Programming model, algorithms and evaluation. In: Bhandarkar, D., Agarwal, A. (eds.) ASPLOS-VIII Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, California, USA, October 3-7, 1998. pp. 81–91. ACM Press (1998). https://doi.org/10.1145/291069.291026
2. Alves, M.A.Z., Diener, M., Santos, P.C., Carro, L.: Large vector extensions inside the HMC. In: Fanucci, L., Teich, J. (eds.) 2016 Design, Automation & Test in Europe Conference & Exhibition, DATE 2016, Dresden, Germany, March 14-18, 2016. pp. 1249–1254. IEEE (2016), http://ieeexplore.ieee.org/document/7459502/

3. Alves, M.A.Z., Villavieja, C., Diener, M., Moreira, F.B., Navaux, P.O.A.: Sinuca: A validated micro-architecture simulator. In: 17th IEEE International Conference on High Performance Computing and Communications, HPCC 2015, 7th IEEE International Symposium on Cyberspace Safety and Security, CSS 2015, and 12th IEEE International Conference on Embedded Software and Systems, ICESS 2015, New York, NY, USA, August 24-26, 2015. pp. 605–610. IEEE (2015). https://doi.org/10.1109/HPCC-CSS-ICESS.2015.166
4. Boncz, P.A., Kersten, M.L., Manegold, S.: Breaking the memory wall in monetdb. Commun. ACM **51**(12), 77–85 (2008). https://doi.org/10.1145/1409360.1409380
5. Breß, S., Mohammad, S., Schallehn, E.: Self-tuning distribution of db-operations on hybrid CPU/GPU platforms. In: Schmitt, I., Saretz, S., Zierenberg, M. (eds.) Proceedings of the 24th GI-Workshop "Grundlagen von Datenbanken 2012", Lübbenau, Germany, May 29 - June 01, 2012. CEUR Workshop Proceedings, vol. 850, pp. 89–94. CEUR-WS.org (2012), http://ceur-ws.org/Vol-850/paper_bress.pdf
6. Cheng, X., He, B., Lu, M., Lau, C.T.: Many-core needs fine-grained scheduling: {A} case study of query processing on intel xeon phi processors. J. Parallel Distrib. Comput. **120**, 395–404 (2018). https://doi.org/10.1016/j.jpdc.2017.09.005
7. Cho, S., Park, C., Oh, H., Kim, S., Yi, Y., Ganger, G.R.: Active disk meets flash: a case for intelligent ssds. In: Malony, A.D., Nemirovsky, M., Midkiff, S.P. (eds.) International Conference on Supercomputing, ICS'13, Eugene, OR, USA - June 10 - 14, 2013. pp. 91–102. ACM (2013). https://doi.org/10.1145/2464996.2465003
8. Cordeiro, A.S., Kepe, T.R., Tome, D.G., Almeida, E.C., Alves, M.A.Z.: Intrinsics-HMC: An automatic trace generator for simulations of processing-in-memory instructions. In: XVIII Brazilian Symposium on High Performance Computing Systems, WSCAD, Campinas, SP, Brazil (2017)
9. DeWitt, D.J., Hawthorn, P.B.: A performance evaluation of data base machine architectures (invited paper). In: Very Large Data Bases, 7th International Conference, September 9-11, 1981, Cannes, France, Proceedings. pp. 199–214. IEEE Computer Society (1981)
10. Do, J., Kee, Y., Patel, J.M., Park, C., Park, K., DeWitt, D.J.: Query processing on smart ssds: opportunities and challenges. In: Ross, K.A., Srivastava, D., Papadias, D. (eds.) Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013. pp. 1221–1230. ACM (2013). https://doi.org/10.1145/2463676.2465295
11. Esmaeilzadeh, H., Blem, E.R., Amant, R.S., Sankaralingam, K., Burger, D.: Dark silicon and the end of multicore scaling. IEEE Micro **32**(3), 122–134 (2012). https://doi.org/10.1109/MM.2012.17
12. Graefe, G., Harizopoulos, S., Kuno, H.A., Shah, M.A., Tsirogiannis, D., Wiener, J.L.: Designing database operators for flash-enabled memory hierarchies. IEEE Data Eng. Bull. **33**(4), 21–27 (2010), http://sites.computer.org/debull/A10dec/hp-paper.pdf
13. Harizopoulos, S., Abadi, D.J., Madden, S., Stonebraker, M.: OLTP through the looking glass, and what we found there. In: Wang, J.T. (ed.) Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008. pp. 981–992. ACM (2008). https://doi.org/10.1145/1376616.1376713
14. HMC Consortium: Hybrid Memory Cube Specification 2.1 (June 2015), http://www.hybridmemorycube.org/, hMC-30G-VSR PHY

15. Hsieh, K., Ebrahimi, E., Kim, G., Chatterjee, N., O'Connor, M., Vijaykumar, N., Mutlu, O., Keckler, S.W.: Transparent offloading and mapping (TOM): enabling programmer-transparent near-data processing in GPU systems. In: 43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016. pp. 204–216. IEEE Computer Society (2016). https://doi.org/10.1109/ISCA.2016.27
16. Jeddeloh, J., Keeth, B.: Hybrid memory cube new DRAM architecture increases density and performance. In: Symposium on VLSI Technology (VLSIT). IEEE (2012). https://doi.org/10.1109/VLSIT.2012.6242474
17. Karnagel, T., Habich, D., Schlegel, B., Lehner, W.: Heterogeneity-aware operator placement in column-store DBMS. Datenbank-Spektrum **14**(3), 211–221 (2014). https://doi.org/10.1007/s13222-014-0167-9
18. Kautz, W.H.: Cellular logic-in-memory arrays. IEEE Trans. Computers **18**(8), 719–727 (1969). https://doi.org/10.1109/T-C.1969.222754
19. Keeton, K., Patterson, D.A., Hellerstein, J.M.: A case for intelligent disks (idisks). SIGMOD Record **27**(3), 42–52 (1998). https://doi.org/10.1145/290593.290602
20. Kepe, T.R.: Dynamic database operator scheduling for processing-in-memory. In: Roy, S.B., da Silva, A.S. (eds.) Proceedings of the VLDB 2018 PhD Workshop co-located with the 44th International Conference on Very Large Databases (VLDB 2018), Rio de Janeiro, Brasil, Aug 27-31, 2018. CEUR Workshop Proceedings, vol. 2175. CEUR-WS.org (2018), http://ceur-ws.org/Vol-2175/paper07.pdf
21. Kim, J., Kim, Y.: HBM: memory solution for bandwidth-hungry processors. In: 2014 IEEE Hot Chips 26 Symposium (HCS), Cupertino, CA, USA, August 10-12, 2014. pp. 1–24. IEEE (2014). https://doi.org/10.1109/HOTCHIPS.2014.7478812
22. Kim, S., Oh, H., Park, C., Cho, S., Lee, S.: Fast, energy efficient scan inside flash memory. In: Bordawekar, R., Lang, C.A. (eds.) International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures - ADMS 2011, Seattle, WA, USA, September 2, 2011. pp. 36–43 (2011), http://www.adms-conf.org/p36-KIM.pdf
23. Minnick, R.C., Goldberg, J., et al.: Cellular arrays for logic and storage. Tech. rep., Stanford Research Inst Menlo Park Calif (Apr 1966)
24. Mirzadeh, N., Kocberber, O., Falsafi, B., Grot, B.: Sort vs. hash join revisited for near-memory execution. In: ASBD@ISCA (2015)
25. Patterson, D.A., Anderson, T.E., Cardwell, N., Fromm, R., Keeton, K., Kozyrakis, C.E., Thomas, R., Yelick, K.A.: A case for intelligent RAM. IEEE Micro **17**(2), 34–44 (1997). https://doi.org/10.1109/40.592312
26. Pattnaik, A., Tang, X., Jog, A., Kayiran, O., Mishra, A.K., Kandemir, M.T., Mutlu, O., Das, C.R.: Scheduling techniques for GPU architectures with processing-in-memory capabilities. In: Zaks, A., Mendelson, B., Rauchwerger, L., Hwu, W.W. (eds.) Proceedings of the 2016 International Conference on Parallel Architectures and Compilation, PACT 2016, Haifa, Israel, September 11-15, 2016. pp. 31–44. ACM (2016). https://doi.org/10.1145/2967938.2967940
27. Santos, P.C., Oliveira, G.F., Tome, D.G., Alves, M.A.Z., de Almeida, E.C., Carro, L.: Operand size reconfiguration for big data processing in memory. In: Atienza, D., Natale, G.D. (eds.) Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, March 27-31, 2017. pp. 710–715. IEEE (2017). https://doi.org/10.23919/DATE.2017.7927081
28. Stone, H.S.: A logic-in-memory computer. IEEE Trans. Computers **19**(1), 73–78 (1970). https://doi.org/10.1109/TC.1970.5008902

29. Tiwari, D., Boboila, S., Vazhkudai, S.S., Kim, Y., Ma, X., Desnoyers, P., Solihin, Y.: Active flash: towards energy-efficient, in-situ data analytics on extreme-scale machines. In: Smith, K.A., Zhou, Y. (eds.) Proceedings of the 11th USENIX conference on File and Storage Technologies, FAST 2013, San Jose, CA, USA, February 12-15, 2013. pp. 119–132. USENIX (2013), https://www.usenix.org/conference/fast13/technical-sessions/presentation/tiwari
30. Tome, D.G., Kepe, T.R., Alves, M.A.Z., de Almeida, E.C.: Near-data filters: Taking another brick from the memory wall. In: Bordawekar, R., Lahiri, T. (eds.) International Workshop on Accelerating Analytics and Data Management Systems Using Modern Processor and Storage Architectures, ADMS@VLDB 2018, Rio de Janeiro, Brazil, August 27, 2018. pp. 42–50 (2018), http://www.adms-conf.org/2018-camera-ready/01_adms_camera_ready.pdf
31. Tome, D.G., Santos, P.C., Carro, L., de Almeida, E.C., Alves, M.A.Z.: HIPE: HMC instruction predication extension applied on database processing. In: 2018 Design, Automation & Test in Europe Conference & Exhibition, DATE 2018, Dresden, Germany, March 19-23, 2018. pp. 261–264. IEEE (2018). https://doi.org/10.23919/DATE.2018.8342015
32. Wang, J., Park, D., Kee, Y., Papakonstantinou, Y., Swanson, S.: SSD in-storage computing for list intersection. In: Proceedings of the 12th International Workshop on Data Management on New Hardware, DaMoN 2016, San Francisco, CA, USA, June 27, 2016. pp. 4:1–4:7. ACM (2016). https://doi.org/10.1145/2933349.2933353
33. Wang, L., Skadron, K.: Implications of the power wall: Dim cores and reconfigurable logic. vol. 33, pp. 40–48 (2013). https://doi.org/10.1109/MM.2013.74