Dynamic Database Operator Scheduling for Processing-in-Memory

Tiago Rodrigo Kepe[†,‡]
Supervised by Eduardo Cunha de Almeida[†]
and Marco Antonio Zanata Alves[†]
[†]Federal University of Paraná and
[‡]Federal Institute of Paraná, Brazil.
trkepe@inf.ufpr.br

ABSTRACT

The emerging architectures for Processing-in-Memory (PIM) present new challenges to database systems: one of them is how to schedule intra-query execution with the x86 processor to be executed inside the memory. In this paper, we show that inefficient scheduling for PIM degrades performance and increases energy consumption, but the proper PIM operators can reach significant improvement in more than one order of magnitude. Therefore, we bring out our vision of *a PIM-aware scheduler for intra-query processing.*

1. INTRODUCTION

The new Processing-in-Memory or Processor-in-Memory (PIM) architectures have emerged as a solution to tackle the problem of data movement between memory and processor, although research in database systems is still premature to carry them out. An open issue is how to coordinate intra-query execution between x86 and underlying memory processor to exploit the potential gain from each device.

The choice of the target architecture for processing impacts directly on query execution because mis-scheduling of database operators can degrade query performance and increase energy consumption. For example, operators with high data reuse benefit from caching mechanism, and thus the x86 processing becomes appealing. On the other hand, operators with data streaming behavior are more suitable for PIM [12]. However, recent work focus on boosting database operators in PIM architectures with a single operator perspective, such as select [12, 13, 14] or join [10, 9]. Such one-sided approach ignores the benefits of cache data reuse.

Furthermore, current query processing scheduling on emerging hardware have focused on multi-core architectures to reduce resource requirements [5], balancing resource utilization across sockets [11] and the proper allocation of cores [4]. Other solutions adapt queries in many-core architectures using fine-grained scheduling of database operators to improve resources usage on the Intel Xeon Phi processor [3, 2]. Those scheduling solutions deal with idiosyncrasies of the multi-core and many-core architectures to run compute-intensive applications. But, PIM architectures, such as Hybrid Memory Cube (HMC) [6], have other peculiarities to accomplish efficient data-intensive applications.

In this thesis, we focus on investigating how to interleave intra-query execution between the x86 and processing-in-memory. As far as we know, this is the first effort in that direction. Existing solutions usually direct data-intensive operators for PIM, however, they neglect the potential of x86 processing using caches for workloads with high temporal and spatial data locality. Thus, our vision is that *a database system requires a PIM-aware scheduler for intra-query processing.* During a Query Execution Plan (QEP) the database scheduler shall be capable of dispatching operators to suitable devices: PIM and x86. In addition, the database system needs efficient implementations of operators for PIM to benefit from the high bandwidth and the parallelism of the underlying hardware.

2. IMPACT OF DATABASE OPERATORS

Initially, we investigate the set of database operators on analytic workloads, like TPC-H, because they generate massive data movement throughout the memory hierarchy [12]. We run the 100 GB TPC-H in the column-oriented database system MonetDB [7], because of its mature research in database kernels for contemporary memory hierarchy. For each TPC-H query, we added the TRACE statement modifier of MonetDB that records broad information of every database operator, including the execution time. We group the most time-consuming operators into four categories: project, select, join, and the remaining ones grouped into the category "others". We perform the experiments on a Intel quad-core i7-5500U@2.40GHz with RAM of 16 GB (DDR3L 1333/1600) and L3 cache size of 4MB running OpenSuse Leap 42.3 on Linux kernel version 4.4.76-1-default. Figure 1 shows the execution time split among those operators for each query and the last bar to the right aggregates the results of the whole TPC-H. The most time-consuming operator is the project (around 56%), followed by select (21%) and join (15%), these operators impact more than 90% in the execution time of TPC-H because DBMSes push them down into the QEP to filter massive volume of data for subsequent operators.

Proceedings of the VLDB 2018 PhD Workshop, August 27, 2018. Rio de Janeiro, Brazil.

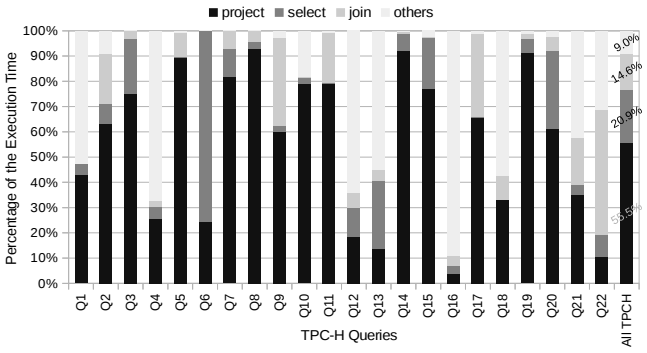


Figure 1: Top time-consuming database operators in MonetDB [7] running the 100 GB TPC-H benchmark.

3. DATABASE OPERATORS INSIDE HMC

In this section, we devise efficient implementations of such operators inside the HMC evaluating the execution time and energy consumption. We used the SiNUCA cycle-accurate in-house simulator [1] to evaluate the execution time with same parameters used by related work [13]. The energy estimations consider the DRAM values for HMC [8]. We evaluate those operators using the 1 G TPC-H because, in that instance, the input data sets fit into the cache, i.e., the best scenario for x86 processing. Based on this analyze is possible to profile the candidate operators for PIM or x86.

3.1 Project Operator

Surprisingly, TPC-H spends around 56% of the execution time and memory footprint with projections. In a further analysis, we investigate the TPC-H Query 03[1] to scrutinize the reasons for our findings. Figure 2 presents a partial execution of the TPC-H Query 03 in a top-down view and shows the interaction among select-project-join operators. The operators select and join generate filters from input columns, such as an array of matching row-ids and bitmaps, and the project operator uses these filters to project other columns in the QEP. Therefore, the impressive impact of projections on TPC-H with MonetDB is because they take the burden to materialize intermediate and final results of other operators. As shown in the diagram of Figure 2, MonetDB performs projections through two primitives: *algebra.projection* and *algebra.projectionpath*. The first projects a column using as input a set of filtered row-ids (or bitmap) generated by either a select or join. The latter receives as input two sets of filtered row-ids (or bitmaps) from the select and join operators, and combining them to project a column.

We avail the projection operator in the HMC [6] to analyze the processing-in-memory performance. The HMC 2.0 is a 3D die-stacked device comprised of four or eight DRAM dies and one logic die. Internally, the HMC is organized as 32 independent vaults. With the support of HIPE [13], a PIM processor with predicated instructions under the HMC, we run HIPE-Projection in the flagged projections of Figure 2. Figure 3 depicts both projection primitives in C and assembly-like codes (HIPE-Projection).

Both primitives traverse bitmap vector(s) to filter a projection column. Inside the HMC, HIPE-Projection performs one load of 256-bytes of the input bitmap(s) and, in case of

[1]We chose Query 03 as it gathers the operators that we are interested in: project, select, join and group-by/aggregation.

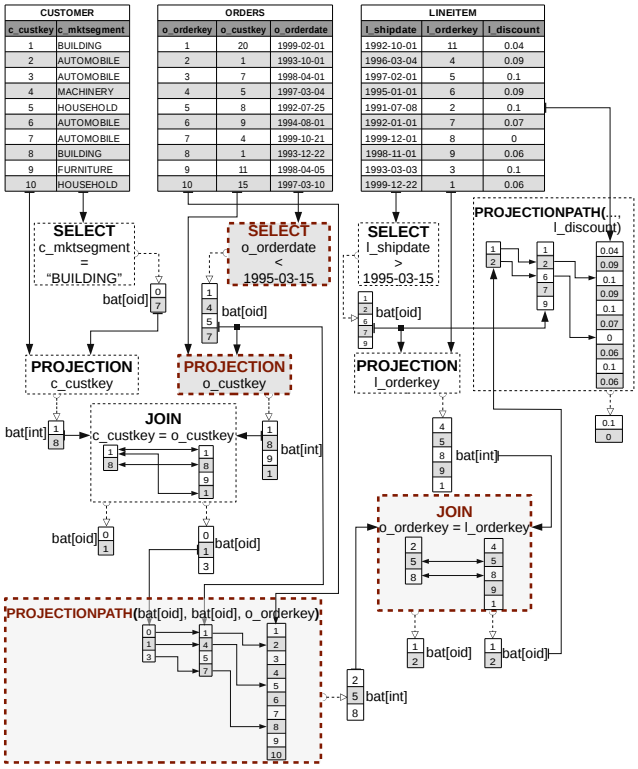


Figure 2: Top-down diagram of a partial execution of the TPC-H Query 03.

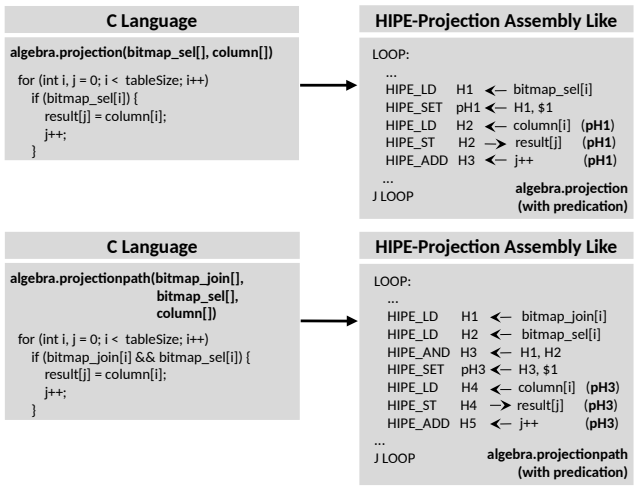
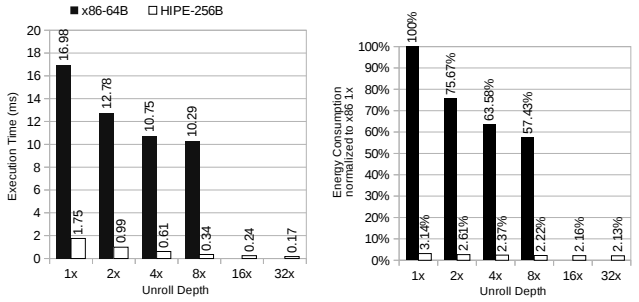


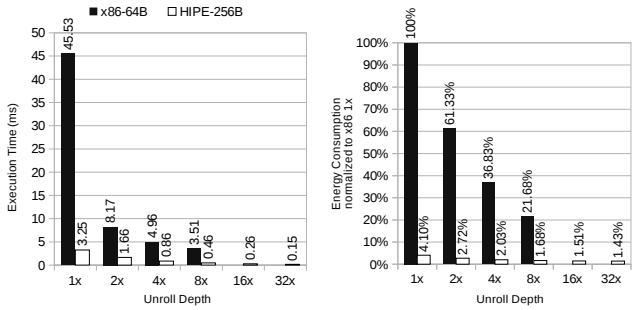
Figure 3: C and HIPE-Projection codes for the two projection primitives.

matched entries, it executes up to 32 parallel loads of 256-bytes of the projection column and stores the values into the result vector. Those HIPE-load and HIPE-store instructions are on-chip memory operations, i.e., the load gets data from DRAM dies to HIPE registers within the HMC, while the store does the inverse. Therefore, the HIPE-Projection benefits from the low on-chip memory latency and it uses the maximum degree of parallelism of the HMC vaults.

Figure 4 shows that HIPE-Projection reduced the execution time by more than one order of magnitude with the



(a) Projection: execution time and energy consumption.



(b) Projectionpath: execution time and energy consumption.

Figure 4: Evaluating of execution time and energy consumption of the HIPE-Projection varying loop unrolling depth in the column-at-a-time engine.

loop unrolling technique of 32x against the best case of the x86 processor with the unroll depth of 8x[2]. The energy consumption of the total DRAM accesses normalized by the x86 execution with unroll depth of 1x is present on figure 4. We observe that the HIPE-Projection reduces the energy consumption around 55% and 20% on average for both projection primitives, respectively. Those results are due to the streaming behavior of projections that causes low data reuse and less amount of off-chip data transfers during the copy of data (materialization). Hence, the results endorse the feasibility of the projection operator for PIM.

3.2 Select Operator

Considering that select scan is the second most time-consuming operator (see Figure 1) responsible for almost 21% of the TPC-H execution time. Traditionally it moves data around the memory hierarchy up to the processor to validate filter conditions on database columns. Similar to the projection operator, we avail the select inside HMC with the implementation of the HIPE-Selection, as depicted in Figure 5. HIPE-Selection traverses a database column testing each value against a constant, which causes compulsory loads of 32x 256-bytes of the select column and, in case of matched entries, it executes parallel stores in the result bitmap vector. These memory accesses are on-chip operations. Therefore, HIPE-Selection reaches the maximum degree of parallelism of the HMC and also exploits the low on-chip memory latency.

[2]The 8x is the deepest unroll generally implemented by compilers due to the reduced number of general purpose registers.

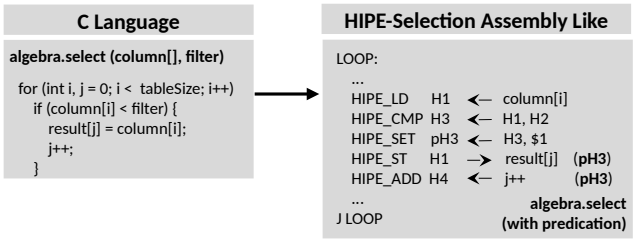
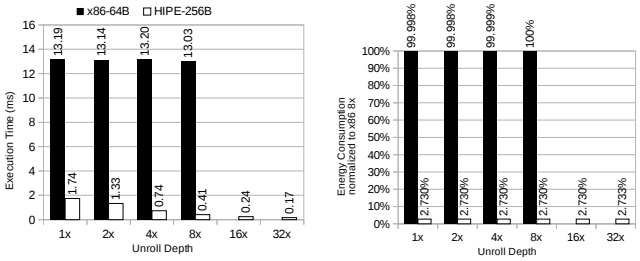


Figure 5: C and HIPE-Selection codes for select operator.



(a) Execution time.

(b) Energy consumption.

Figure 6: Evaluating of execution time and energy consumption of the HIPE-Selection varying loop unrolling depth in the column-at-a-time engine.

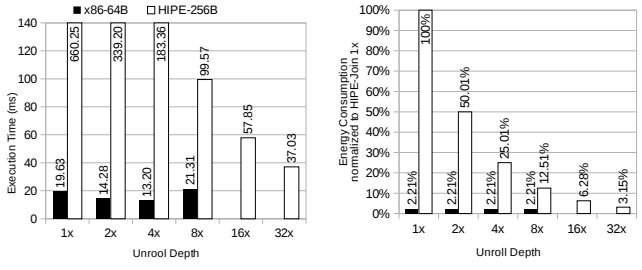
We run HIPE-Selection in the flagged selection of Figure 2 for performance and energy consumption analysis. Figure 6a shows that HIPE-Selection unrolled 32x reduces the execution time by 76x compared to the best x86 scenario with the unroll depth of 8x. Figure 6b brings the total DRAM energy consumption normalized by the x86 execution with unroll depth of 8x. HIPE-Selection reduces the energy consumption by around 98% of any unroll version. In HIPE-Selection most data transfers take place on-chip, but the selection in the x86 causes off-chip data transfers for the entire column and the result vector (i.e., data movement throughout the cache memory hierarchy). These results corroborate for the feasibility of the select operator for PIM.

3.3 Join Operator

The join operator has been studied for years, with different algorithms to exploit the potential of the x86 processor. Most of the algorithms belong to the classes of hash join and sort-merge join. However, such algorithms generate random memory accesses that inhibit the potential of HMC (high bandwidth and parallelism over contiguous data), and indeed the HMC is better exploited for streaming applications [12]. Thus, we implemented the Nested Loop Join (NLJ) due to its streaming behavior that benefits from the HMC parallelism. The NLJ traverses the join columns in two loops: the outer and the inner. In HIPE-Join[3], the inner loop is unrolled up to 32x to exploit the data access parallelism of the HMC. But, different from other PIM operators, the HIPE-Join reaches poor performance for the execution time in the flagged join of Figure 2, as shown in Figure 7a. Even the energy consumption is worst in the best HIPE-Join: 3% with the unroll depth of 32x shown in Figure 7b, compared to any version of the x86. Although the

[3]We suppressed the HIPE-Join codes due to space restriction.

NLJ streams the join columns, it enforces *data reuse* when repeatedly traversing the inner loop. Furthermore, the join columns fit into the cache hierarchy and thus only the first compulsory misses occur when loading the columns in the cache. Thereafter, there is no LLC cache misses until the end of the execution while the inner column is reused N times. By contrast, in the HIPE-Join every inner loop iteration cause compulsory LOAD and STORE (in case of matched join values) instructions. The loop unrolling technique reduces such impact due to the high internal bandwidth of the HMC, as depicted in Figure 7. However, it is not enough to improve performance nor energy consumption when using PIM.



(a) Execution time. (b) Energy consumption.

Figure 7: Evaluating of execution time and energy consumption of the HIPE-Join varying loop unrolling depth in the column-at-a-time engine.

4. PIM-AWARE SCHEDULER

Database system environments with PIM demand careful decision by the QEP to interleave intra-query execution with the x86 processor, but the choice of the best target architecture depends on intra-operator data reuse, which fluctuates according to the operator behavior, the cache settings, and data characteristics. This motivates the design of a PIM-aware scheduler: our future work on dynamic database operator scheduling for emerging PIM architectures.

The scheduler is a critical performance component of a DBMS: it orchestrates the execution of physical primitives of a QEP. Thus, we envision two scheduling strategies for a PIM-aware scheduler: static scheduling and dynamic scheduling. These strategies receive as input the optimal plan generated by the query optimizer, coordinating intra-query execution between PIM and the x86.

Static scheduling: before query execution, using a classification model based on operator profile to decide in which architecture to process each operator.

Dynamic scheduling: during query execution, beginning with the static scheduling and modifying on-the-fly the operator implementation as the scheduler detects abnormal data reuse.

5. CONCLUSIONS AND FUTURE WORK

Although PIM architectures feature as high performance memory by delivering unprecedented bandwidth, data access parallelism and computational power, not every application benefits from PIM capabilities. Thus, database systems must accurately interleave intra-query processing between PIM and x86. In this direction, we introduce our vision of a PIM-aware scheduler for intra-query processing. We first investigate the most impacting database operators in analytic workloads and develop adjusted PIM operators: HIPE-[*Projection*, *Selection* and *Join*]. The first two proved efficient for the execution time and energy consumption. However, the latter presented another perspective that is the intra-operator data reuse is a heavy factor for scheduling decisions. Our ongoing work is the static scheduling strategy based on operator profiles. Our future work is the dynamic scheduling strategy to reschedule operators on-the-fly.

6. ACKNOWLEDGMENTS

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

7. REFERENCES

[1] M. A. Z. Alves, C. Villavieja, M. Diener, and t al. Sinuca: A validated micro-architecture simulator. *HPCC*, 2015.

[2] X. Cheng, B. He, M. Lu, and C. T. Lau. Many-core needs fine-grained scheduling: A case study of query processing on intel xeon phi processors. *Journal of Parallel and Distributed Computing*, 2017.

[3] X. Cheng, B. He, M. Lu, C. T. Lau, H. P. Huynh, and R. S. M. Goh. Efficient query processing on many-core architectures: A case study with intel xeon phi processor. In *SIGMOD*, pages 2081–2084, 2016.

[4] S. Dominico, E. C. de Almeida, J. A. Meira, and M. A. Z. Alves. An elastic multi-core allocation mechanism for database systems. In *ICDE*, 2018.

[5] J. Giceva, G. Alonso, T. Roscoe, and T. Harris. Deployment of query plans on multicores. *PVLDB*, 8(3):233–244, 2014.

[6] HMC Consortium. *Hybrid Memory Cube Specification 2.1*, June 2015. HMC-30G-VSR PHY.

[7] S. Idreos et al. Monetdb: Two decades of research in column-oriented database architectures. *IEEE Data Eng. Bull.*, pages 40–45, 2012.

[8] J. Jeddeloh and B. Keeth. Hybrid memory cube new dram architecture increases density and performance. In *VLSI*, 2012.

[9] Y. O. Koçberber et al. Meet the walkers: accelerating index traversals for in-memory databases. In *MICRO-46*, pages 468–479, 2013.

[10] N. S. Mirzadeh, O. Kocberber, B. Falsafi, and B. Grot. Sort vs. hash join revisited for near-memory execution. In *ASBD@ISCA*, 2015.

[11] I. Psaroudakis, T. Scheuer, N. May, A. Sellami, and A. Ailamaki. Adaptive numa-aware data placement and task scheduling for analytical workloads in main-memory column-stores. *PVLDB*, pages 37–48, 2016.

[12] P. C. Santos, G. F. Oliveira, D. G. Tomé, M. A. Z. Alves, E. C. de Almeida, and L. Carro. Operand size reconfiguration for big data processing in memory. In *DATE*, pages 710–715, 2017.

[13] D. G. Tomé, P. C. Santo, L. Carro, E. C. Almeida, and M. A. Z. Alves. Hipe: Hmc instruction predication extension applied on database processing. In *DATE*, pages 261–264, 2018.

[14] S. L. Xi, O. Babarinsa, M. Athanassoulis, and S. Idreos. Beyond the wall: Near-data processing for databases. In *DaMoN*, pages 2:1–2:10, 2015.