Hardware Data Re-organization Engine for Real-Time Systems

Shahin Roozkhosh*, Denis Hoornaert†, Renato Mancuso* and, Manos Athanassoulis*
*Boston University †Technische Universität München
*{shahin, rmancuso, mathan}@bu.edu, †denis.hoornaert@tum.de

***Abstract*—Access patterns and cache utilization play a key role in the analyzability of data-intensive applications. In this demo, we re-examine our previous research on software-hardware co-design to push data transformation closer to memory from a real-time perspective. Deployed in modern CPU+FPGA systems, our design enables efficient and cache-friendly access to large data by only moving relevant bytes from the target memory. This (1) compresses the cache footprint and (2) reorganizes complex memory access patterns into sequential and predictable patterns.**

***Index Terms*—Memory Semantic, Data re-organization**

I. Introduction

One of the key bottlenecks in modern computing is moving data through the memory hierarchy to processing elements. The corresponding predictability issues are particularly problematic in real-time systems, especially when large-footprint applications exhibiting complex memory access patterns are considered. Multi-level caches have been introduced to hide the latency of memory fetches. They are effective in optimizing performance when data accesses are characterized by *spatial* and *temporal* locality.

Unfortunately, achieving spatiotemporal locality in large-footprint applications is challenging. Motivated by this, our recently published work [1] investigates the role of hardware-aided on-the-fly data reshaping for a specific class of large-footprint applications, i.e., database systems.

Relational databases typically store in-memory relations (tables) employing a row-oriented layout—offering good locality for transactional processing—or a column-oriented layout, with good locality for analytical processing. New applications, however, blend analytical and transactional processing. Therefore, no single optimal layout exists. At the same time, switching between them introduces costly bookkeeping and data duplication overheads [2].

The same challenge also appears in real-time workloads such as image processing and neural-network-based applications, where accessing tensor data often results in complex strides that break locality. Moreover, the mismatch between the size of cache lines and data items (e.g., integers, double) results in unwanted data being transferred from main memory. As the size of the accessed data grows, moving data through the memory hierarchy becomes a fundamental bottleneck. The higher the pressure exerted on the bottleneck, the more unpredictability worsens.

While the main focus of our previous work is to optimize the average-case performance of relational databases, our intuition suggests that on-the-fly data reorganization can also bring significant benefits in terms of predictability for two main reasons: (1) reduced inter-process cache line eviction—thanks to cache footprint compression and (2) conversion of complex access patterns into sequential accesses—from the cache and prefetcher perspective. Overall, enforcing access patterns with high locality is increasingly more challenging in data-intensive applications in both real-time and relational systems. In all these applications, the processing is performed by streaming over a set of data items that are (1) orders of magnitude larger than the typical size of CPU caches; (2) often sparsely stored in memory; and (3) accessed with hard-to-predict, input-dependent patterns that are not optimized for the linear organization of data in DRAM. In addition, often, the computation performed on each data item is minimal. Thus, hiding the cost of data movement via deep pipelines and instruction-level reordering becomes ineffective.

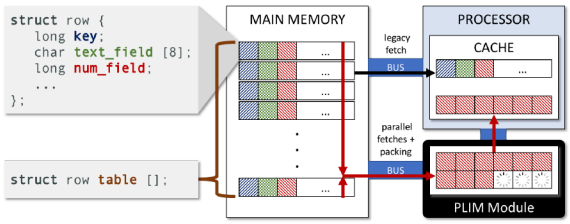
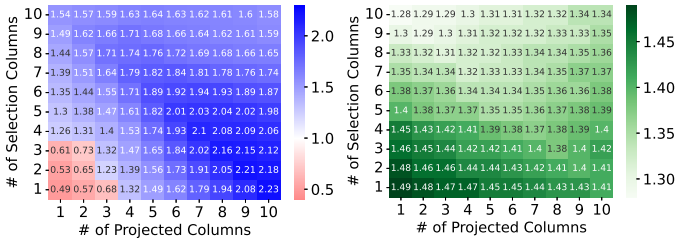


Fig. 1: On-the-fly data transformation enhancing data locality.

In our demo, we will review the implications of on-the-fly data reorganization in CPU+FPGA systems. Next, we will provide a walk-through of our Relational Memory Engine (RME), capabilities, and deployment procedure on a real hardware platform. Finally, we will showcase the live acquisition of measurements that highlight the benefits of data reorganization from the standpoint of performance predictability.

II. Data Re-organization Engine

The RME is a hardware module located between the last-level cache (LLC) and memory. Cache refills on a (configurable set of) variables go through the FPGA where RME resides to capture CPU-memory accesses on-the-fly. Upon the first capture, it initiates a set of transfers from main memory to carve out only the desired bytes and into an internal buffer where data locality is maximized (Figure 1). Once ready, a cache line of packed useful data is available to the CPU as if it existed in the main memory.



(a) Speedup - RME vs Columnar (b) Speedup - RME vs Row

Fig. 2: Heat map of the RME speed-up against columnar 2a and row 2b store for varying projected and selected columns.

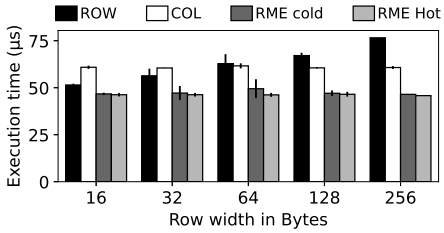


Fig. 3: RME enables deterministic accesses latency.

We implemented and deployed RME on commercially available Systems-on-Chips (SoCs) integrating an on-chip FPGA and a traditional multi-core processor (e.g., Intel HARPv2, Xilinx UltraScale+). By employing commercially available CPU+FPGA SoCs, we create an immediately-usable complete prototype capable of running realistic applications. Our design is based on the Programmable Logic In the Middle (PLIM) [3] approach and can be employed to achieve greater control over memory traffic by instantiating custom logic as an intermediary between processors and main memory.

III. Data-reshape for real-times systems

RME creates a re-organized alias of the target memory based on a software-provided configuration. RME achieves the timeliness requirements of real-time systems by accessing only the desired subset of data items in main memory on behalf of the processing units before sending fully compressed cache lines to the LLC. This mechanism effectively filters out all undesired elements that would otherwise pollute the cache, enabling high data locality in upstream caching layers.

Motivated by real-time applicability, first, we experimentally demonstrate that RME offers efficient native accesses to any matrix column or column group, outperforming direct row-wise and direct columnar accesses. To perform a fair comparison, we implement RME, the row-store (ROW), and the column-store (COL) approach in the same memory. The default size of each row is 64 bytes, and the column width is 4 bytes. Each experiment was repeated 30 times, and we reported averages and standard deviations. We run two sets of experiments for RME: hot (when the targeted data is ready in the internal) and cold (otherwise).

We design a synthetic benchmark (Listing 1) to test the behavior of our engine under representative memory access patterns. Consider the following operation: Given a matrix M, it reads over the columns subset based on a different selection predicate. Here, $COL_{p_1}, ..., COL_{p_i}$ are projection columns and $COL_{s_1}, ..., COL_{s_j}$ are selection columns.

Listing 1: Synthetic Matrix Operation

```
READ COL_{p_1}, ..., COL_{p_i} FROM M WHERE COL_{s_1}, ..., COL_{s_j} > k ;
```

A. Latency Showcase

Figures 2a and 2b show the speedup of RME compared to the in-memory row-store and column-store. In the x- and y-axis we vary the number of projection (i) and selection (j) columns. Figure 2a shows that when the number of involved columns is small (≤ 4), column-store dominates over RME (colored red). However, as the number of columns increases due to the tuple materialization cost, the diminished prefetching columnar access performance falls behind. In fact, RME can be up to $2.23\times$ faster than columnar access (bottom rightmost cell). Figure 2b further highlights that RME **always** outperforms in-memory row access by being $1.3-1.5\times$ faster.

B. Predictability Showcase

We continue our experimentation with the benchmark above where $i = 1, j = 1, COL_i \neq COL_j$, focusing on the comparison between RME, direct row-wise (ROW), and direct columnar access (COL). We access 4 byte-wide columns while varying the row size. Figure 3 shows the absolute latency.

We note from this figure that even without having the projected column in the Reshape Buffer in FPGA (RME *cold*), *RME has faster execution* than both ROW and COL in all experiments. The reason is that (1) RME better exploits the internal memory bandwidth to fetch only the desired data items at bus-width granularity, and (2) the CPU caches are not polluted with unwanted fields.

RME's latency remains virtually the same as it accesses only the relevant data. However, answering the query via direct access of the row-oriented data leads to poor cache utilization as larger rows lead to higher cache pollution. Conversely, RME exhibits *stable and predictable performance regardless of the row size*. Thus, RME allows predicting and exploiting data reuse across processing phases.

C. Real-Time Evaluation

RME outperforms the row-store layout because, by definition, it accesses fewer data. On the other hand, queries that access fewer columns can be more efficiently evaluated from a columnar layout. However, when the number of projected columns is high enough (more than four in our setup), RME outperforms the columnar layout. Further, the RME implementation used in this setup runs at only 1/3 of the maximum FPGA frequency. Operating at a higher frequency may reduce memory access time and increase the benefits of RME.

IV. Conclusion

We **depart from the traditional view of memory as a flat array of bytes**. We reshape the data via near-memory computation before moving it to the CPU, resulting in improvement of both performance and determinism of memory accesses.

ACKNOWLEDGMENT

The material presented in this paper is based upon work supported by the National Science Foundation (NSF) under grant number CCF-2008799. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the NSF. Denis Hoornaert was supported by the Chair for Cyber-Physical Systems in Production Engineering at TUM and the Alexander von Humboldt Foundation.

REFERENCES

[1] S. Roozkhosh, D. Hoornaert, J. H. Mun, T. I. Papon, A. Sanaullah, U. Drepper, R. Mancuso, and M. Athanassoulis, "Relational memory: Native in-memory accesses on rows and columns," in *Proceedings 26th International Conference on Extending Database Technology, EDBT 2023, Ioannina, Greece, March 28-31, 2023*. [Online]. Available: https://doi.org/10.48786/edbt.2023.06

[2] R. Appuswamy, M. Karpathiotakis, D. Porobic, and A. Ailamaki, "The Case For Heterogeneous HTAP," in *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*, 2017. [Online]. Available: http://cidrdb.org/cidr2017/papers/p21-appuswamy-cidr17.pdf

[3] S. Roozkhosh and R. Mancuso, "The potential of programmable logic in the middle: cache bleaching," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2020.