Processing-In-Memory of Data Filter On Compressed Data

Tiago R. Kepe[1,2], **Francis B. Moreira**[1], **Marco A. Z. Alves**[1]

[1]Departament of Informatics – Federal University of Paraná (UFPR)

[2]Departament of Informatics – Federal Institute of Paraná (IFPR)

{trkepe,fbm,mazalves}@inf.ufpr.br

Abstract. *The data filter is essential in data-centric applications, but it requires moving large data sets to the processing units. One approach to tackle such a hassle is data compression by using lightweight compression methods such as dictionary encoding. In this paper, we exploit the idea of Processing-In-Memory (PIM) data filters directly over compressed data. The initial experiments show noticeable speed-ups of over 2x against the AVX512 architecture.*

1. Introduction

The data filter operation is ubiquitous in data-centric applications from database systems to specialized algorithms like machine learning, genome analysis, and data science. However, it requires moving large data sets to the processing units to compute smaller filtered subsets. This data movement is time and energy-consuming, further impairing today's big data applications as the amount of data grows exponentially. Some solutions, especially in the database field, use data compression not only for storage reduction but also to improve the operation latency [Abadi et al. 2006]. Some approaches apply filters directly on compressed data [Li and Patel 2013, Feng et al. 2015] but still have two residual drawbacks: **1**) they demand data transfer through the memory hierarchy; **2**) they require a costly and non-vectorized operation to extract a bitmap from the filter's result.

In this paper, we engage in (1) Processing-In-Memory (PIM) architectures to deal with the data movement issues and (2) a lightweight compression method suitable for filtering compressed data and enabling intra-bank and inter-bank parallelism through in-memory vectorized instructions. Our goal is to evaluate the performance of the equality filter on compressed data with PIM against AVX512 and reveal research opportunities.

2. Data Filter On Compressed Data with PIM

PIM architecture. We use the Vector-In-Memory Architecture (VIMA) [Cordeiro and et al. 2021] as a target PIM architecture because of its native vectorized capabilities and intrinsic instructions. VIMA can operate over 8 KB vector registers performing arithmetic on integers, floating-point, and bitwise instructions.

Data Compression. Many applications apply data compression to alleviate storage requirements. Compression can also aid in vectorized architectures (e.g., Intel-AVX512) to boost data processing parallelism [Feng et al. 2015]. As the decompression step is costly, lightweight compression methods are required to enable data filtering on compressed data. A widespread method is the dictionary encoding with value/code order-preserving such that $code_i < code_j$ for every $value_i < value_j$ from a data set, which preserves data semantics allowing to perform arithmetic and bitwise operations on compressed data.

The dictionary maps distinct values from a data set to unique compressed codes. For instance, Brazil has twenty-seven states, which we can represent with 5 bits. Each state is associated with a code by preserving the order of the values: $AC \rightarrow 00000, AL \rightarrow 00001, ..., TO \rightarrow 11011$. We store these codes in a byte slice that is suitable for SIMD data-parallelism [Feng et al. 2015]. Then we apply a padding of 1's and keep only the most significant bit in 0, e.g., $AC \rightarrow \mathbf{011}00000$. After, we generate a proper bitmask (BM) for every code by setting the more significant bit in 1, e.g., $BM_{AC} \rightarrow \mathbf{1}1100000$.

With this dictionary structure, we can filter all people that reside in the Acre state by extracting the bitmask (e.g., 11100000). We replicate as many copies as possible in a vector register and apply a series of proper arithmetic and bitwise SIMD instructions to filter the compressed codes that match the Acre state code, generating a resulting bitmap.

We evaluated such equality filter with the OrCS[1] in-house simulator and the data set from the TPC-H[2] benchmark with factor of 1 GB. We filtered a compressed column using two different approaches: 1) We transfer the target data set to the CPU to process it using Intel-AVX512 instructions and 64 B register vectors; 2) We ran the same experiment with VIMA using vector registers of 8 KB to assess the performance of computing filters within the memory chip. Figure 1 present a high speed-up of over 2x using the PIM capabilities of VIMA.

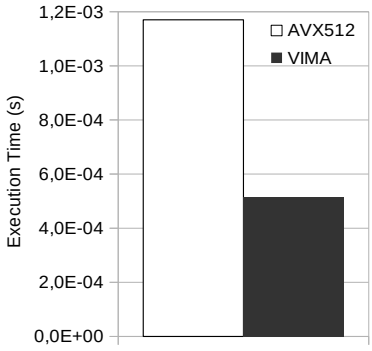

Figure 1. Evaluation of equality filter on compressed data in AVX512 and VIMA.

3. Conclusions

Data filter on compressed data with PIM is feasible and promising. Our preliminary result (over 2x speed-up) has shown the advantage PIM can bring to applications that rely on data filtering. Besides equality filters, other kinds of data filters still need to be evaluated, such as inequality filters, range filters, and chains of filters to assess data reuse.

References

Abadi, D. J., Madden, S., and Ferreira, M. (2006). Integrating compression and execution in column-oriented database systems. SIGMOD, pages 671–682.

Cordeiro, A. S. and et al. (2021). Machine learning migration for efficient near-data processing. In *Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, pages 212–219.

Feng, Z., Lo, E., Kao, B., and Xu, W. (2015). Byteslice: Pushing the envelop of main memory data processing with a new storage layout. SIGMOD, pages 31–46.

Li, Y. and Patel, J. M. (2013). Bitweaving: fast scans for main memory data processing. SIGMOD, pages 289–300.

[1]https://github.com/mazalves/OrCS

[2]http://www.tpc.org/tpch/