JAFAR: Near-Data Processing for Databases

Oreoluwa Babarinsa
Harvard University
obabarinsa@seas.harvard.edu

Stratos Idreos
Harvard University
stratos@seas.harvard.edu

ABSTRACT

As main-memory sizes have grown, data systems have been able to process entire large-scale data-sets in memory. However, because memory speeds have been not been keeping pace with CPU speeds, the cost of moving data into CPU caches has begun to dominate certain operations within in-memory data systems. Recent advances in hardware architectures point to near memory computation capabilities becoming possible soon. This allows us to rethink how database systems process queries and how they split computation across the various computational units. In this paper, we present JAFAR, a near data processing accelerator for pushing selects down to memory. Through a detailed simulation of JAFAR hardware we show it has the potential to provide up to 900% improvement for select operations in modern column-stores.

1. INTRODUCTION

The Development of In-memory Data Systems. Rapidly diminishing costs of main memory per gigabyte have led to the development of in-memory data systems, significantly increasing throughput and performance compared to disk-based systems. However, while disk accesses are no longer the performance bottleneck for in-memory data systems, the cost of moving data from main memory across the memory buses and into CPU caches is still significant [2]. Big data applications like data analytics tend to be more memory bound than CPU-bound. This trend, called "memory wall" [11], will only become worse as CPU performance improvement continues to surpass memory performance improvement.

Furthermore, while there have been fruitful efforts in tuning database operators to be cache-aware and efficient in multi-core settings [3], ultimately, data still must be moved from main memory to the CPUs, incurring a significant cost. There is extensive literature on optimizing the memory behavior of DBMS workloads, such as improving cache hit rates with multi-core optimized joins [3]. Past work, however, does not directly try to reduce movement of relevant data from memory to the CPUs.

Near Data Processing. Reducing data movement can be achieved by moving the computation closer to the data itself by having specialized hardware that performs computation close to the data. This is known as *near-data processing* (NDP). In the hardware architecture community, NDP has been proposed in various forms in the past [7, 10]. However, NDP hardware did not see widespread adoption, when it was originally proposed due to strong past gains in CPU performance. However, due to diminishing returns from technology scaling, there has been a recent resurgence in NDP research [2]. Historically, proposals for NDP systems have emphasized design for general purpose computation [5, 6, 8].

Our Contribution. In this paper, we study NDP for columnar data systems [1]. We present JAFAR, short for "Just a Filtering Accelerator on Relations". JAFAR is a hardware accelerator embedded in a DRAM module that implements the select operator of a modern column-store. According to our experiments, JAFAR is capable of achieving up to 9× speedup on the selection operator.

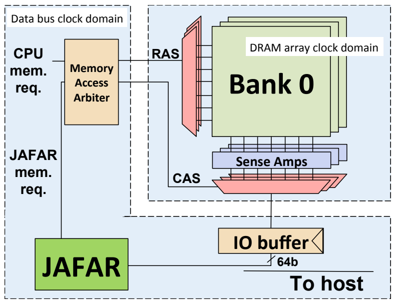
2. JAFAR

JAFAR allows for processing database filtering operations directly in memory. When a select operator is pushed to JAFAR by the database system, JAFAR starts filtering data directly in memory. The key idea is that JAFAR requests data from memory as normal but it filters data without pushing data further up in the memory hierarchy. To achieve that JAFAR receives input directly from the IO buffer of the DRAM module, as shown in Figure 1a. JAFAR targets integer input data, covering a spectrum of use cases. Modern data systems typically store columns of data employing dictionary compression, allowing a variety of data types to be represented by integers or fixed-width data.

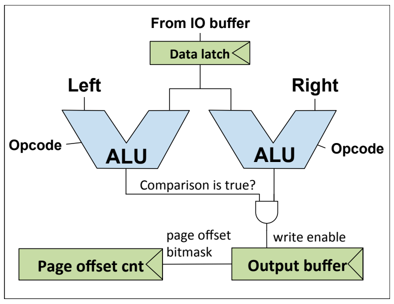
The internals of JAFAR are depicted in Figure 1b. For each 64 bit word that it receives, an integer comparison is performed against the value of the element corresponding to the query predicate. For range filters, two arithmetic logic units (ALUs) operate in parallel. JAFAR tracks the offset of the current row in the page. If the result of the filter is true, then the offset is converted into a bitmask and written into an output buffer, which is a bitset indicating which rows pass the filter. The output buffer thus holds n bits to represent the state of n filter operations. Every n cycles, the output buffer is filled, the bitset is written back to main memory at a pre-programmed location. The CPU controls JAFAR via memory-mapped accelerator control registers and is notified on completion by polling a shared memory location.

We design and simulate JAFAR using an accelerator modeling tool called Aladdin [9] and the surrounding system with the gem5 simulator [4]. Aladdin is a power and performance modeling tool that converts a C-style representation of the workload being accelerated into a data dependence graph, which represents the structure and execution of the accelerator datapath itself.

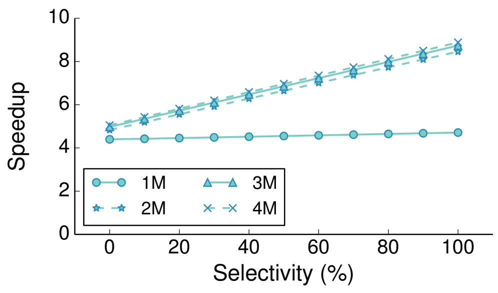
SIGMOD '15 May 31 - June 04, 2015, Melbourne, VIC, Australia
ACM 978-1-4503-2758-9/15/05
http://dx.doi.org/10.1145/2723372.2764942.



(a) DDR3 DRAM module



(b) JAFAR architecture



(c) Speedup increases with selectivity

Figure 1: JAFAR design and evaluation

3. EXPERIMENTAL ANALYSIS

Experimental Setup. Using the gem5 simulator we simulate a machine with a single 1 GHZ CPU with 64KB L1, 128KB L2 cache, and 2GB of DRAM. We experiment using a synthetic dataset consisting of integers between 0 and 10^6, randomly generated from a uniform distribution. We evaluate JAFAR using scans and selections of varying selectivity over different data sizes. Figure 1c shows on the y-axis the speedup achieved when running a scan followed by a selection using JAFAR when compared with a CPU-only execution. On the x-axis we vary the selectivity from 10% to 100%. The four different lines correspond to different data sizes. We calculate speed-up by comparing the run-time of a query on our simulated machine with and without JAFAR handling scans.

Impact of Dataset Size. We first discuss the impact of dataset size on JAFAR (the different lines in Figure 1c). JAFAR speedup is small for small datasets (< 1M rows), but increases to a maximum of 9× for large datasets. This is because on large datasets, most of the program execution is spent in the accelerated selection operation. For smaller datasets, the amount of time in the accelerated part of the code is reduced, thus leading to a smaller speedup. In this experiment, there is no memory contention when JAFAR is running because the CPU is spin-waiting until JAFAR finishes.

Impact of Selectivity. In addition to the data size, selectivity plays an important role in the achieved speed-up due to JAFAR, especially for data sizes larger than 1M (note the top three lines in in Figure 1c). We observe that JAFAR performance increases with query selectivity. This behavior is attributed to a key difference between how JAFAR operates compared to a traditional CPU-based execution. CPU executes additional code on a record when a row passes the filter. On the other hand, JAFAR always writes the contents of the output buffer back to main memory each time the buffer is full, without delaying the filtering operation. Hence, JAFAR has constant execution time regardless of the query selectivity. Combining that with the increasing number of commands the CPU needs to execute for increasing selectivity, results to the observed linear increase in speedup for higher selectivity.

Contention. JAFAR provides considerable speedup on filtering operations, increasing both with data size and query selectivity. However, so far we did not consider memory contention effects, which are important because JAFAR was designed as a drop-in NDP accelerator which can can only operate while the memory controller is idle. To quantify this effect, we measure the idle periods when using a modern data system. We use several filter-heavy TPC-H queries running on MonetDB on a NUMA machine with 4 sockets, each equipped with an Intel Xeon E7-4820 v2 Ivy processor running at 2.0GHz with 16MB of L3 cache. We find that the memory controller idle periods range between 200 and 800 memory bus clock cycles, with an average of 500 cycles. Thus, on average, JAFAR can process $500/4 = 125$ 32-byte data blocks, or a total of 4KB of data, per idle period. Due to space restrictions we do include this analysis here. More details can be found in a subsequent publication [12].

4. SUMMARY & FUTURE WORK

We present JAFAR, an NDP hardware accelerator embedded into DRAM that provides up to 9× speedup on filtering operations in modern column-stores. A more complete version of this study is now available [12], where we analyze the performance gains when running analytical queries and quantify memory contention.

Our future research plans include the design of further operators, as well as studying NDP opportunities for row-stores and hybrid systems. Another important topic is to identify what are the minimum changes required at the host system and software stack as well as study possible optimizations based on data types.

Acknowledgments. The authors would like to thank Sam Xi and Manos Athanassoulis, member of Harvard DASlab, for their help during this project.

5. REFERENCES

[1] D. J. Abadi, P. Boncz, S. Harizopoulos, S. Idreos, and S. Madden. The Design and Implementation of Modern Column-Oriented Database Systems. *Foundations and Trends in Databases*, 5(3):197–280, 2013.

[2] R. Balasubramonian, J. Chang, T. Manning, J. H. Moreno, R. Murphy, R. Nair, and S. Swanson. Near-Data Processing: Insights from a MICRO-46 Workshop. *IEEE Micro*, 34(4):36–42, 2014.

[3] C. Balkesen, J. Teubner, G. Alonso, and M. T. Ozsu. Main-memory hash joins on multi-core CPUs: Tuning to the underlying hardware. In *Proceedings of the IEEE International Conf. on Data Engineering (ICDE)*, pages 362–373, 2013.

[4] N. L. Binkert et al. The gem5 simulator. *SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[5] M. Gokhale, W. Holmes, and K. Iobst. Processing in Memory: The Terasys Massively Parallel PIM Array. *IEEE Computer*, 28(4):23–31, 1995.

[6] M. Hall et al. Mapping Irregular Applications to DIVA, a PIM-based Data-intensive Architecture. In *Proceedings of the ACM/IEEE Conf. on Supercomputing*, 1999.

[7] W. H. Kautz. Cellular Logic-in-Memory Arrays. *IEEE Transactions on Computers*, 18(8):719–727, 1969.

[8] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick. A Case for Intelligent RAM. *IEEE Micro*, 17(2):34–44, 1997.

[9] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks. Aladdin: A Pre-RTL, Power-performance Accelerator Simulator Enabling Large Design Space Exploration of Customized Architectures. In *Proc. of the Annual International Symposium on Computer Architecture (ISCA)*, pages 97–108, 2014.

[10] H. S. Stone. A Logic-in-Memory Computer. *IEEE Transactions on Computers*, 19(1):73–78, 1970.

[11] W. A. Wulf and S. A. McKee. Hitting the Memory Wall: Implications of the Obvious. *ACM SIGARCH Computer Architecture News*, 23(1):20–24, 1995.

[12] S. Xi, O. Babarinsa, M. Athanassoulis, and S. Idreos. Beyond the Wall: Near-Data Processing for Databases. In *Proceedings of the International Workshop on Data Management on New Hardware (DAMON)*, 2015.