Beyond the Wall: Near-Data Processing for Databases

Sam (Likun) Xi Oreoluwa Babarinsa Manos Athanassoulis Stratos Idreos
Harvard University
{*samxi*, *obabarinsa*, *manos*, *stratos*}@*seas.harvard.edu*

ABSTRACT

The continuous growth of main memory size allows modern data systems to process entire large scale datasets in memory. The increase in memory capacity, however, is not matched by proportional decrease in memory latency, causing a mismatch for in-memory processing. As a result, data movement through the memory hierarchy is now one of the main performance bottlenecks for main memory data systems. Database systems researchers have proposed several innovative solutions to minimize data movement and to make data access patterns hardware-aware. Nevertheless, all relevant rows and columns for a given query have to be moved through the memory hierarchy; hence, movement of large data sets is on the critical path.

In this paper, we present JAFAR, a Near-Data Processing (NDP) accelerator for pushing selects down to memory in modern column-stores. JAFAR implements the select operator and allows only qualifying data to travel up the memory hierarchy. Through a detailed simulation of JAFAR hardware we show that it has the potential to provide 9× improvement for selects in column-stores. In addition, we discuss both hardware and software challenges for using NDP in database systems as well as opportunities for further NDP accelerators to boost additional relational operators.

1. INTRODUCTION

The Development of In-Memory Data Systems. In recent years, the rapidly diminishing cost of main memory per gigabyte has 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 [4]. Big data applications like data analytics and transactional processing tend to be more memory-bound than CPU-bound. This trend, called the "memory wall" [56], is expected only to become worse as CPU performance improvement continues to surpass memory performance improvement.

DaMoN'15, June 1, 2015, Melbourne, Victoria, Australia.

http://dx.doi.org/10.1145/2771937.2771945

Database systems research has already been driven by "memory wall" since several decades ago [49]. In the late 1990s it led to the development of drastically alternative physical layouts and engines [3, 7] and as of early 2000s there was a massive transformation of modern systems to column-store designs that are highly tuned for modern hardware [1]. Furthermore, the database research community continues to focus on hardware-aware designs, i.e., algorithms that are cache-aware and fully utilize multi-cores, such as work on optimizing joins [5], B^+-trees [44], transactions on NUMA cores [40] and hardware optimized adaptive indexing [38, 39]. Past work, however, does not directly try to reduce data movement from memory to the CPUs. Instead, it focuses on utilizing existing hardware as best as possible; relevant data will still be pushed up the memory hierarchy, causing a significant cost as we move to bigger data sizes.

Near-Data Processing. Reducing data movement of even qualifying data can be achieved by moving the computation closer to the data itself, an approach known as *near-data processing* (NDP)[1]. NDP refers to creating specialized hardware located close to the data (i.e., on disk or memory), supporting a small set of computational capabilities, an idea already proposed in various forms decades ago [27, 51]. Nevertheless, NDP hardware has not yet seen widespread adoption in commercial products because, when it was originally proposed, Moore's Law and Dennard scaling enabled tremendous, continuous gains in CPU performance. Due to diminishing returns from technology scaling, however, there has been a recent resurgence in NDP research [4].

Historically, proposals for NDP systems have focused on general purpose computation [14, 20, 32, 37]. In recent years, however, there have been efforts in studying the performance and power improvement opportunities provided by specialized near data processing hardware accelerators that are designed for a limited set of functions and therefore can discard many sources of overhead associated with general purpose processing [12, 18, 30, 42].

NDP in Modern Database Systems. In this paper, we study the potential of near-data processing hardware ac-

[1]Past work has also used the terms "processing in memory" or "logic in memory". Here, we use the NDP terminology.

celerators for modern data systems and the associated side-effects for data system design. With an increasing number of database applications keeping all or hot data exclusively in large main memories, the memory wall becomes the primary bottleneck. Several database operators, such as selection, projection, and aggregation, produce strictly less than or the same amount of output as input, making them amenable to optimization of data movement. Executing these operators directly in memory and only transporting the necessary data (i.e., qualifying tuples, qualifying columns, or aggregates) through the memory subsystem leaves the CPU free to perform other tasks, reduces cache pollution, and alleviates memory bus pressure. On the other hand, NDP for operators that may produce larger results than their input, like joins, cannot always guarantee performance improvement. In this paper, we focus on studying the potential benefits and design space of NDP for select operators.

Select operators have improved significantly in recent years by using techniques such as working over compressed data, vectorization, and multicores, however, these only help if the system is not memory bound. On the contrary, designing NDP solutions for select operators allows us to avoid moving data completely.

It is also interesting to observe that the ability of NDP to reduce the total amount of data pushed to the CPU through the memory hierarchy is similar to the benefits achieved by columnar storage. However, columnar storage still requires entire relevant columns to be propagated through the memory hierarchy, whereas NDP selects can further filter the data by pushing up only the relevant tuples of the relevant columns.

Our Contribution. This paper makes the following contributions.

1. We present JAFAR, short for "Just a Filtering Accelerator on Relations", an accelerator embedded in a DRAM module that implements the select operator of a modern column-store (Section 2).
2. We show that JAFAR can provide up to 9× speedup for the select operator, demonstrating significant potential for NDP in database systems (Section 3).
3. We present a roadmap of research challenges and opportunities towards fully utilizing NDP in modern data systems (Section 4).

2. DESIGN OF JAFAR

In this section, we present the design and implementation of JAFAR. We first provide the necessary background by giving an overview of how DDR3 SDRAM operates.

2.1 Overview of DDR3 SDRAM

Figure 1(a) depicts a simplified diagram of a single DDR3 (dual-data rate) SDRAM (synchronous DRAM) chip on a DIMM (dual in-line memory module). A DIMM is composed of one or two *ranks*, which are collections of *separately packaged SDRAM chips*. Each chip is comprised of multiple independently addressable *banks*, where each bank is a collection of *arrays*. Data is interleaved across each array so that a memory request is serviced in parallel across all the arrays while keeping wire lengths short[2]. To access data, the memory controller decodes the physical memory address into row address strobe (RAS) and column address strobe (CAS) signals, which are then sent to DRAM. The RAS activates a row across all arrays of a bank by loading the stored bits into a row buffer, and the CAS selects one column of data from each array. Row activation requires precharging of the bitlines and sense amplification, so consecutive accesses to an active row are much faster than consecutive accesses to different rows.

In general, DRAM access latency is governed by four timing parameters: (1) CL: CAS latency, which is the minimum delay between successive CAS signals; (2) tRCD: row-to-column delay, which is the minimum delay between a RAS signal and the first CAS signal; (3) tRP: row precharge time, which is the time needed to close the current active row and precharge bitlines for the next row; (4) tRAS: active to precharge delay, which sets a minimum amount of time between accesses to two different rows.

DRAM array speed is limited by how quickly bitlines can be precharged, how quickly DRAM bitcells can discharge, and how quickly the sense amplifiers can operate. This total latency is on the order of tens of nanoseconds. In order to keep up with increasing CPU clock speeds and demand for data, the DRAM module is divided into two primary clock domains. The internal storage arrays are clocked around 100-200 MHz, while the externally-facing circuitry runs at a much faster clock.

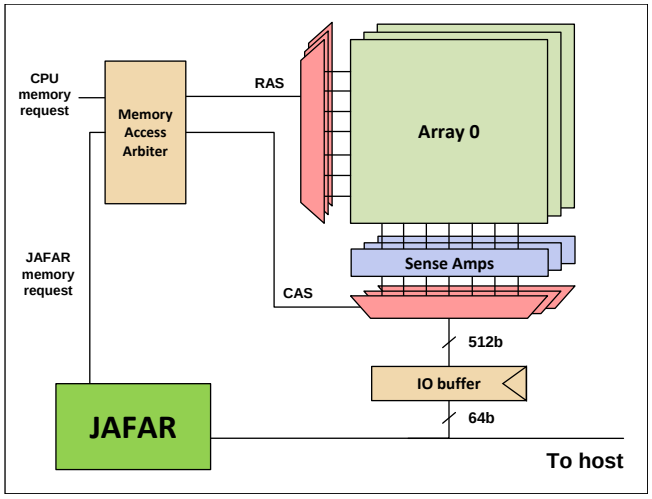
DDR3 is an $8n$-prefetch design, which means that a read request for a 64-bit data word can return up to 512 bits ($8\times$ what was requested), starting from the requested address. This design exploits spatial and temporal locality of typical memory access patterns and reduces the number of CAS signals that need to be sent. DRAM accesses are slow by nature, but the prefetching design amortizes this cost across many bits, greatly increasing the available memory bandwidth. The 512 bits are loaded into an internal IO buffer and then streamed out to the data bus 64 bits at a time on both the rising and falling edges of the clock (hence the term "dual data rate") over four data bus clock cycles. This means that the data bus clock domain must be four times faster than the internal array clock in order to be ready for the next 512 bits.

2.2 Architecture of JAFAR

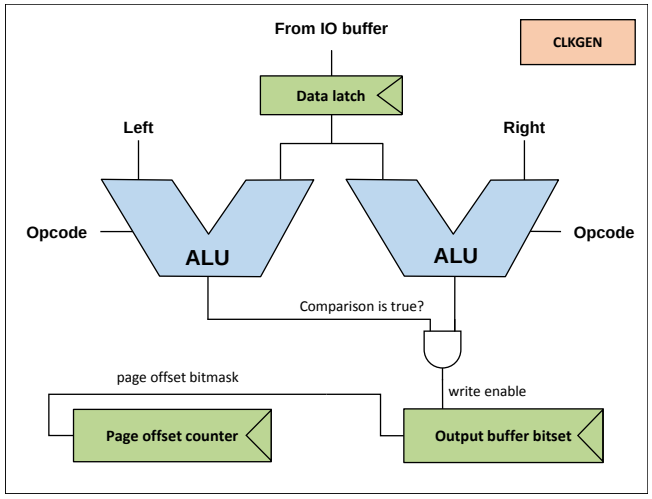
We now proceed to describe the design of JAFAR, which allows a data system to process the select operator of a modern column-store directly in memory.

Overall Operation. The architecture of JAFAR is shown in Figure 1(b). When a select operator is pushed to JAFAR by the database system, it starts filtering data directly in memory. Our current design supports the following predicates: $=$, $<$, $>$, $\leq$, and $\geq$ and works over integer data, but the design is easily extensible for supporting other data

[2]There exist alternative DRAM architectures; here, we present a common design for SDRAM.



(a) Integration of JAFAR with DDR3 SDRAM.



(b) JAFAR architecture.

Figure 1: JAFAR operates at the DRAM module level and receives input from the DRAM IO buffer. The leftmost figure shows how JAFAR obtains data and sends control signals (for simplicity, we show one bank out of one chip on the module).

types and operations (discussed in Section 4). At a high level, JAFAR requests data from DRAM in the same way that a CPU would, but the filtered data is not pushed back up the memory hierarchy until it is requested by the CPU after the operation completes.

JAFAR issues read requests and receives data directly from the IO buffer of the DRAM module as shown in Figure 1(a). For each 64 bit word received, an integer comparison is performed against the value of the tuple element corresponding to the query predicate. For range filters, two arithmetic logic units (ALUs) operate in parallel as shown in Figure 1(b). Integers are sufficient to capture most datatypes in modern data systems, but select accelerators like JAFAR can be extended to natively support floating-point and variable length datatypes as well. While executing the comparison, JAFAR tracks the current row offset 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 passed the filter. The output buffer holds n bits to represent the state of n filter operations. Every n cycles, the output buffer is fully filled and its contents are written back to DRAM at a pre-programmed location. The CPU controls the operation of JAFAR via memory-mapped accelerator control registers and is currently notified of JAFAR operation completion by polling a shared memory location (CPU utilization in a complete system can be improved by using hardware interrupts). To fit column-stores with a late materialization execution engine, JAFAR is designed to consume one complete column at a time.

Because the IO buffer operates on a dual-pumped clock, JAFAR receives two 64-bit data words per data bus clock cycle, one on each of the rising and falling edge. Rather than building ALUs and latches for a dual-pumped clock, JAFAR generates its own clock that is twice as fast as the data bus clock, thereby reducing hardware complexity. Looking at future opportunities, it is interesting to consider that current DDR3 SDRAM devices typically have CAS latencies of around 13ns [34]; in contrast, JAFAR operates at around 2GHz, or twice the data bus clock frequency (which is around 1GHz on DDR3). Each DRAM access retrieves up to eight 64-bit words, and JAFAR can process one per clock cycle (0.5ns) for a total of 4ns. As a result, JAFAR currently spends a total of 9 out of 13 nanoseconds waiting for data to arrive, which implies that there are opportunities to include more complex calculations, like hashing or aggregates, at virtually no additional latency.

```
int  errno = select_jafar(
     void*      col_data,
     int        range_low,
     int        range_high,
     uint8_t*   out_buf,
     size_t     num_input_rows,
     size_t*    num_output_rows);
```

Figure 2: JAFAR's API.

Programming JAFAR. We envision that JAFAR would be invoked using the API described in Figure 2. In the proposed API, `col_data` is a pointer to the start of a virtual memory page containing the column data and `range_low` and `range_high` define the inclusive bounds of the range filter. The output bitset is returned as a byte array in `out_buf`. The API is designed so that this function must be called for every page in the column, since JAFAR must rely on the CPU to provide memory translation services.

Handling Data Interleaving. Systems that have more than one DIMM installed must decide how the address space should be organized across them. The system can either choose to completely fill up one DIMM before moving on to the next, or it can interleave data across multiple DIMMs. Data interleaving is only possible if the DIMMs are symmetric (i.e. same capacity and latencies) and the memory controller supports multi-channel memory.

JAFAR can handle either case. The former case is straightforward because memory pages are contiguous, and so it does not require any change to JAFAR. The latter case has two solutions. Because interleaving is at the 64-bit granularity and JAFAR operates on 64-bit words, JAFAR can still perform its filtering operations as usual, but when it writes the output bitset back to main memory, it must only overwrite bits corresponding to rows it has operated on. Alternatively, the database storage engine can explicitly shuffle column data so that the physical layout is contiguous; this is an approach that has been taken by existing work [12].

Coordinating DRAM Access. Accesses to DRAM must obey a strict set of timing rules, so they must be carefully coordinated between the host CPU and JAFAR to prevent collisions and interference. Access to DRAM will be arbitrated by the query execution manager. If JAFAR has exclusive access to a DRAM rank that stores all the required input data and output buffers, its performance is extremely predictable; therefore, the query manager can grant "ownership" of a DRAM rank to JAFAR for a specified number of cycles, knowing that JAFAR will finish its allotted work in that amount of time.

The concept of DRAM rank ownership has been used in past work [12]. Here we discuss a possible way of passing ownership by appropriately setting DRAM mode registers. Mode registers are typically used to configure timing parameters, burst length, and other such parameters, but we believe that we can repurpose mode register 3 (MR3) for our use case. MR3 activates the multipurpose register (MPR), and when the MPR is enabled, the memory controller is only permitted to send read/write commands to the MPR, not to the DRAM chips. This effectively blocks the memory controller from issuing any ordinary reads and writes. Mode registers can be set via user-level code at runtime. This opens up many interesting questions about how to schedule DRAM ownership transfers in order to minimize the impact on the rest of the system. Verifying the viability of this design is ongoing work; more research on this problem will help deliver significant performance benefits with NDP.

Physical Implementation. JAFAR is designed as an external integrated circuit mounted on a DIMM. This means that JAFAR does not need to be fabricated using a DRAM logic process, which is well known to be unsuited for general purpose digital logic. It also means that JAFAR cannot affect the density or yields of DRAM chips, as the economics of the DRAM industry are extremely sensitive to these factors. This implementation strategy is similar to that used for fully-buffered memory, which implement the advanced memory buffer as an external chip [25].

3. EXPERIMENTAL RESULTS

In this section, we demonstrate that JAFAR has the potential to bring significant improvements, achieving up to 9× speedup over CPU-only execution. We first describe implementation details and experimental setup.

3.1 Experimental Setup

We simulate JAFAR using Aladdin [48], an accelerator modeling tool, and the surrounding system with gem5 [6].

gem5 simulator	Intel Xeon E7-4820 v2
One out-of-order CPU	Eight 2-way SMT cores
1GHz CPU	2GHz CPU
1 socket	4 socket server (32 phys. cores)
64kB L1, 128kB L2	256kB L1, 2MB L2, 16MB L3
2GB DRAM	1TB DDR3 SDRAM

Table 1: Specifications of our evaluation platforms. gem5 is used to model the performance of JAFAR in a single processor system in order to isolate the performance characteristics of JAFAR, while the Intel Xeon system is used to profile real database workloads.

Aladdin is an accelerator power and performance modeling tool that converts a C-style representation of the workload being accelerated into a dynamic data dependence graph which represents the structure and execution of the accelerator datapath itself. The dependence graph captures compute operations (e.g., add, subtract, compare), memory operations, and conditional statements, and Aladdin performs a variety of graph optimizations such as loop unrolling and pipelining. The resulting graph is then "executed cycle-by-cycle" by a breadth-first traversal that also takes into account constraints like memory bandwidth and available functional units.

The gem5 simulator is a widely used cycle-accurate CPU full system simulator. We integrated Aladdin into gem5's DRAM timing model, supported by gem5's out-of-order x86 core model. The integration enables the CPU to invoke JAFAR via system calls, and while JAFAR is executing, the CPU is free to do other work or wait until JAFAR is finished. In this analysis, our benchmarks do the latter as they focus on the NDP select operators, but in a full blown system, the CPU can perform other operations in parallel for the same or concurrent queries. The full specifications of the simulated system is given in the leftmost column of Table 1. This system was designed to be fairly simple in order to isolate the raw performance improvement possible with JAFAR.

To integrate JAFAR with a database system, we use an in-house prototype column-store that is capable of performing select-project-join queries using bulk processing and can invoke JAFAR to push down selections to the accelerator. To focus on the potential for improvement by pushing selects to the accelerator, we focus here on a query workload that consists of simple single column select queries, varying selectivity from 0% to 100%.

We experiment with 4 million rows in which all values are randomly generated integers uniformly distributed between 0 and 1 million. The columns are not sorted or indexed, so a complete scan of all rows is required. Although this dataset is admittedly very small for a database systems analysis, it is nonetheless an accurate sampling of the system's performance on a larger dataset because the workload is extremely regular with essentially no control flow. Sampling-based simulation is standard practice when using tools like CPU simulators because simulating the billions and trillions of instructions required for a dataset of billions of rows is prohibitively expensive. Furthermore, we find that 93% of the

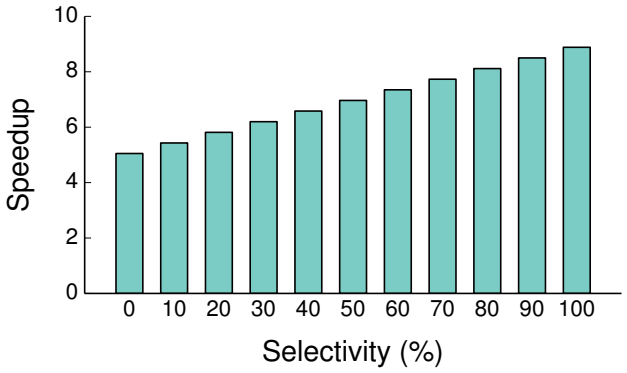


Figure 3: Simulated selection speedup obtained by JAFAR for a dataset of uniformly distributed random integers.

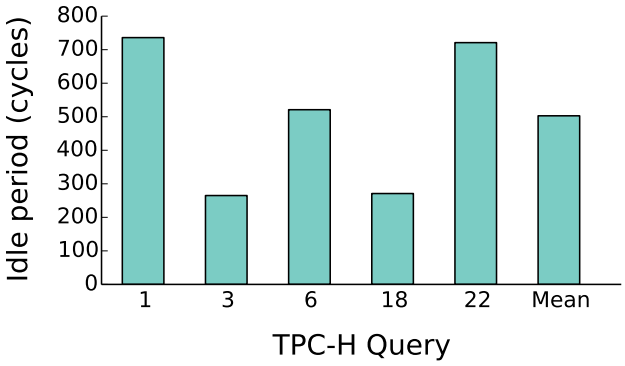


Figure 4: Memory controller idle time estimates for several TPC-H queries. Cycles refer to the memory bus clock.

total execution time is spent inside the accelerated region, so we are approaching the maximum attainable speedup for the single select operator under the current design. Four million rows is also larger than the total cache capacity of the simulated CPU, ensuring that the entire input set cannot fit into cache. Nevertheless, building the infrastructure to evaluate JAFAR over much bigger data sets which can illuminate further insights is currently in progress.

3.2 Speedup on Filtering Operations

We first evaluate the potential speedup on filtering operations of JAFAR over traditional CPU execution as a function of both the dataset size and the query selectivity. In Figure 3 we show the achieved speedup of JAFAR over CPU-only execution on the y-axis, while varying the selectivity on the x-axis. JAFAR provides significant speedup over CPU-only execution. The speedup gradually increases from 5× for 0% selectivity to 9× for 100% selectivity. In this experiment, there is no memory contention when JAFAR is running because the CPU is spin-waiting until it finishes.

The increased speedup with higher selectivity is attributed to a key difference between how JAFAR operates compared to a traditional CPU-based execution. CPU executes additional code to 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 irrespective of the query selectivity. Combining this with the additional instructions the CPU needs to execute to record results, leads in a linear increase in speedup for higher selectivity. In addition, it is worth noting that here we do not use predication for the software that run the selects in the CPU. Thus, JAFAR would materialize even bigger benefits for lower selectivity against a database system that uses predication for robustness, because while predication leads to more stable and better performance on average, for lower selectivity it has adverse impact. Essentially, JAFAR implements predication at the hardware level at zero cost.

3.3 Quantifying Memory 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 whenever JAFAR is running on a DRAM rank, the system cannot access that part of memory, and vice versa. Contention for memory has important implications for both JAFAR and system performance, so access to memory must be appropriately arbitrated by a scheduler. Exploring the large range of memory access schedulers is beyond the scope of this preliminary work. However, as a first step, it is interesting to examine JAFAR's potential performance when faced with memory contention in the *absence* of a scheduler.

Without a scheduling system, JAFAR can only run while the memory controller is idle or it would cause unexpected delays in CPU memory requests. To quantify the length of memory controller idle periods on a modern data system, we run several filter-heavy TPC-H queries on MonetDB [21] (version 11.19.7) on a high-end Intel Xeon server (described in detail in Table 1) and profile the system by sampling performance counters in the integrated memory controllers.

Calculating the distribution of idle period length with samples from performance counters is extremely difficult. The available performance counters provide the number of cycles the read queue of the memory controller is *busy* (RC_{busy}), and the number of cycles the write queue is *busy* (WC_{busy}), however, we are interested in the number of cycles the memory controller is idle, that is, both queues are *simultaneously empty* (MC_{empty}). We calculate the lower bound of MC_{empty}, during an experiment *total_cycles* long, by assuming zero overlap between the cycles that read or write requests are served: $MC_{empty} = total\_cycles - RC_{busy} - WC_{busy}$. Then, we estimate the mean idle period as the ratio between MC_{empty} and the total number of reads and writes: $mean\_idle\_period = \frac{MC_{empty}}{\#reads+\#writes}$. This is a pessimistic estimate, so we can expect the actual mean idle period to be higher.

Figure 4 shows on the y-axis the memory controller idle period in cycles, when several TPC-H queries are executed using MonetDB. Specifically, we show the idle period for Queries 1, 3, 6, 18, and 22, as well as the average idle time. The memory controller idle period ranges between 200 and 800 memory bus clock cycles, with an average of 500 cycles. DDR3's 8n-prefetch design means each memory request oc-

cupies at least four bus cycles (ignoring tCL, RAS/CAS, row activation, and precharging); this means that at most, JAFAR can process 500/4 = 125 32-byte data blocks, or a total of 4KB of data, per idle period.

To put this in context, there are commercial DDR3 chips whose banks store 8KB of data per row [34]; JAFAR would on average process half of a DRAM-activated row before an interruption. Interruptions are costly because if the address requested does not hit in the current active row, then that row must be flushed, the DRAM bitlines precharged, and a fresh row read from the arrays, incurring significant additional latency. Ideally, JAFAR could process at least an entire DRAM row uninterrupted.

This analysis shows that properly coordinating the on-chip memory controllers with any NDP units available is key to achieving good performance, which motivates additional work in memory access scheduling. Past work has shown that reordering DRAM reads and writes can provide large increases in memory bandwidth and overall system performance [35, 36, 45]. In this context, JAFAR is simply an additional agent of memory requests, but one that is highly sensitive to intervening requests. This discussion hints at interesting optimization problems for query plan scheduling that opens up suitable memory controller idle periods for accelerators like JAFAR to utilize.

4. OPPORTUNITIES AND CHALLENGES

JAFAR showcases that NDP for data systems has significant potential, however there are both more opportunities and challenges when designing near-data processing accelerators for data systems. Below, we discuss both new opportunities and challenges of employing NDP in data systems.

Aggregations. Aggregations such as sum, average, minimum, maximum, etc. require minimal additional hardware to support. For hash-based aggregations, common hash functions like SHA and MD5 can be provided *a priori* as fixed function hardware units, while custom hash functions could potentially be supported via reconfigurable logic. There is a large body of existing work on hardware accelerated SHA and MD5 [9, 10, 47], and reconfigurable logic has been extensively studied as well [15, 16, 19]. Due to hardware restrictions, there must be a limit to the number of hash buckets JAFAR can support, which suggests that a hierarchical aggregation approach will be required.

Projections. Project (or tuple reconstructions) operators are necessary in column-stores to fetch the qualifying values from one column based on a selection and a position list of another column. Projections are essential in column-store plans as every query plan has at least $N - 1$ project operators where N is the number of columns referenced in the query. Thus, creating NDP accelerators for projections or accelerators that combine filtering with projections may result in significant benefits.

Joins. A join operator may produce more tuples than its input and, thus, near-data processing may hurt performance. This raises challenging optimization problems regarding how to choose where to run a join and how to split computation across active computational units.

Sorting. Sorting is used widely in database query plans, such as sorting a position list after an index scan or in a order-based `group by`. Sorting algorithms suitable for hardware acceleration have been extensively studied in existing literature using GPUs [17, 46], FPGAs [29, 33], and ASICs [24]. JAFAR can easily incorporate a fixed function sort accelerator to support sorting. Because ASIC sorters are generally costly in terms of area, implementations are typically limited to sorting a small number of elements at a time. This does not prevent sorting larger datasets, using a divide-and-conquer approach.

Indexing and Compression. Indexing in column-stores [22, 23, 50] and working over compressed data [2, 31, 58] is used to reduce the amount of data that must be scanned. As NDP accelerators like JAFAR can perform extremely efficient scans, this raises the research question of whether NDP obviates the need for indexing and compression.

Data Types. JAFAR currently supports integer data only. While it can easily be extended to support additional fixed-length data types like floating-point numbers, support for variable-length data types such as strings is more difficult to adapt to JAFAR's architecture. Some past work in NDP for databases leaves filtering on strings as a task for the CPU to handle [53]. However, it is worth noting that many modern systems effectively handle string columns as integers using dictionary compression (e.g., to handle equality predicates).

NDP in Row-Stores and Hybrids. Near-data processing for row-stores or hybrids that store data as column-groups can be achieved by slightly altering the design of JAFAR to be able to apply in parallel different filtering operations to different attributes and record the result of the collective filter accordingly. Comparing near data processing in row-stores versus column-stores or hybrids with column-groups is a very interesting open topic and may affect the way we think these basic architectures relate to each other regarding their competitive advantages. Furthermore, NDP accelerators could also be used to support efficient projections on row stores directly in memory. For example, to do this, JAFAR would simply activate a row in DRAM and read the desired columns into internal buffers. When the internal buffers are full, JAFAR will dump the contents back to a pre-allocated memory location. This projection operation would thus not require moving data into the CPU caches and back. Removal of duplicate tuples will ultimately require support from the CPU since all intermediate results must be coalesced and examined as a whole.

Memory Management. Data systems do not manage how data in main memory is arranged in terms of which particular DRAM DIMM stores which region of data. Instead, the kernel assigns address ranges to specific DIMMs at boot, presenting the abstraction of main memory as a single source. In traditional data systems, this can create performance bottlenecks due to non-uniform memory accesses (NUMA), because accessing main memory can result in waiting for another chip to supply the requested data. Since JAFAR can only process data that is resident on its DIMM, the data system needs to know what data is located on which DIMM when invoking JAFAR. Therefore, prior to invoking JAFAR, the operating system must first pin the

memory pages JAFAR will access to specific DIMMs. Pinning guarantees that a virtual memory region is resident in RAM and is accomplished via the `mlock` and `munlock` system calls. To activate JAFAR, the data system invokes system calls that specify the virtual starting and ending address of the data region. The operating system can then translate this into the physical address and activate the appropriate JAFAR unit. Studying near-data processing APIs for database systems as well as adding support for more than one DIMM are essential future steps.

5. RELATED WORK

Building Hardware-Aware Data Systems. The data management systems community has been studying efficient ways to store and access data in order to match underlying hardware properties. Over the past fifteen years, there has been work on cache-conscious database algorithms [49], work on alternative layouts [3], as well as engines [7]. Column-stores represent a major transformation for database architectures and were essentially designed from the start to be hardware-aware [1]. More recent research has also focused on sharing the cost of reading data across multiple queries as one more effort to minimize the amount of data moved [13, 41, 43, 59]. Researchers have also proposed accelerating predicate handling and data decompression for in-memory scans with SIMD instructions [52].

Our work takes a different route, pushing computation to memory through a combination of new hardware and software designs. It allows to avoid moving even qualifying data up the memory hierarchy and highlights the research opportunity to rethink how we can utilize past experiences in system design with new hardware opportunities and co-design.

Database Accelerators. There has been a stream of novel work on hardware accelerators for database query processing in the past few years. Widx is an on-chip accelerator for database hash index look-ups, reducing their cost by decoupling key hashing from pointer chasing [28]. Widx uses a host core's TLB and L1 D-cache as its data source. HARP is a hardware accelerator for range partitioning that relies on a special stream buffer framework to retrieve data from main memory [54]. Q100 is a database processing unit (DPU) that implements a domain-specific instruction set targeting database operations [55]. Traditional query plans can be translated into DPU instructions and executed on Q100, which uses a stream buffering framework similar to that in HARP. Ibex is an FPGA-based storage engine for a row-store database embedded in the storage layer [53]. Following the near-data processing principle in a lower level of the overall memory/storage hierarchy than JAFAR, Ibex implements selection, projection, and aggregation, within the flash storage layer. All of these past efforts promise significant performance and power improvements over pure software implementations.

Past work on database accelerators is orthogonal to our work because all of them require movement of data from main memory to either CPU cache or special buffers, whereas our approach leverages NDP to greatly reduce data movement by processing data directly in memory. NDP presents an interesting set of new challenges as well as opportunities compared to off-chip or on-chip accelerator design, spanning both hardware design and database system design as discussed in the previous section. In addition, if NDP accelerators were to co-exist with other on-chip database accelerators, there could be important implications for database architectures that are open for exploration.

Database Machines. Database machines is another area relevant to our work with considerable research done in the 1970s and early 1980s [8]. The main motivation was to create hardware that is tailored to process database workloads efficiently. Database machines were not widely adopted because the custom storage media that these machines required were expensive to manufacture and scale with increasing disk density. Furthermore, technology scaling at the time enabled dramatic, steady improvements in CPU performance that provided sufficient processing power.

However, the situation today is different for several reasons. Diminishing returns from technology scaling motivate work for NDP and hardware acceleration. Also, NDP can be accomplished without compromising the cost per bit of storage by separating the DRAM logic process with the computational logic process [12, 26].

Near-Data Processing. NDP was originally proposed a few decades ago as a potential way to address the fact that CPUs were improving at a much faster rate than memory technologies. Several opportunities and challenges of NDP for general systems have been discussed in past work, such as the IRAM Project [37], the DIVA project [20], Terasys [14], and the Computational RAM project [11]. There has been a recent resurgence of interest in near-data processing due to both advancements in fabrication technologies as well as the development in-memory data-systems [4]. Zhang et al. [57] showed that high performance NDP can provide critical throughput compared to modern high-performance multi-core CPUs. Pugsley et al. [42] showed potential for NDP to enhance MapReduce. Our work extends this line of work towards a complete hardware-software co-design for NDP in modern data systems.

6. CONCLUSION

We present JAFAR, a near-data processing hardware accelerator embedded into DRAM modules that implements the select operator of a modern column-store. JAFAR scans and filters columns directly in memory without pushing the data up the memory hierarchy. We show that JAFAR can provide up to 9× speedup compared to moving the data to the CPU. In addition, we present a roadmap of research challenges and opportunities towards fully utilizing near-data processing for database systems.

Acknowledgments. This work is partially supported by the Swiss National Science Foundation and by the National Science Foundation under Grant No. IIS-1452595.

7. 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] D. J. Abadi, S. Madden, and M. Ferreira. Integrating compression and execution in column-oriented database systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 671–682, 2006.

[3] A. Ailamaki, D. J. DeWitt, M. D. Hill, and M. Skounakis. Weaving Relations for Cache Performance. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 169–180, 2001.

[4] 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.

[5] 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 Conference on Data Engineering (ICDE)*, pages 362–373, 2013.

[6] N. L. Binkert, B. M. Beckmann, G. Black, S. K. Reinhardt, A. G. Saidi, A. Basu, J. Hestness, D. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[7] P. Boncz, S. Manegold, and M. L. Kersten. Database architecture optimized for the new bottleneck: Memory access. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 54–65, 1999.

[8] H. Boral and D. J. DeWitt. Database Machines: An Idea Whose Time has Passed? A Critique of the Future of Database Machines. In H.-O. Leilich and M. Missikoff, editors, *Database Machines*, pages 166–187. 1983.

[9] R. Chaves, G. Kuzmanov, L. Sousa, and S. Vassiliadis. Cost-Efficient SHA Hardware Accelerators. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(8):999–1008, 2008.

[10] L. Dadda, M. Macchetti, and J. Owen. The Design of a High Speed ASIC Unit for the Hash Function SHA-256 (384, 512). In *Proceedings of the Design, Automation and Test in Europe Conference and Exposition (DATE)*, pages 70–75, 2004.

[11] D. G. Elliott, W. M. Snelgrove, and M. Stumm. Computational RAM A memory SIMD hybrid and its application to DSP. In *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC)*, pages 30.6.1–30.6.4, 1992.

[12] A. Farmahini-Farahani, J. H. Ahn, K. Morrow, and N. S. Kim. NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 283–295, 2015.

[13] G. Giannikis, G. Alonso, and D. Kossmann. SharedDB: killing one thousand queries with one stone. *Proceedings of the VLDB Endowment*, 5(6):526–537, 2012.

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

[15] S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, and R. R. Taylor. PipeRench: A Reconfigurable Architecture and Compiler. *IEEE Computer*, 33(4):70–77, 2000.

[16] V. Govindaraju, C.-H. Ho, T. Nowatzki, J. Chhugani, N. Satish, K. Sankaralingam, and C. Kim. DySER: Unifying Functionality and Parallelism Specialization for Energy-Efficient Computing. *IEEE Micro*, 32(5):38–51, 2012.

[17] A. Greßand G. Zachmann. GPU-ABiSort: optimal parallel sorting on stream architectures. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, 2006.

[18] Q. Guo, X. Guo, Y. Bai, and E. Ipek. A resistive TCAM accelerator for data-intensive computing. In *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 339–350, 2011.

[19] S. Gupta, S. Feng, A. Ansari, S. A. Mahlke, and D. I. August. Bundled execution of recurring traces for energy-efficient general purpose processing. In *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 12–23, 2011.

[20] M. Hall, P. Kogge, J. Koller, P. Diniz, J. Chame, J. Draper, J. LaCoss, J. Granacki, J. Brockman, A. Srivastava, W. Athas, V. Freeh, J. Shin, and J. Park. Mapping Irregular Applications to DIVA, a PIM-based Data-intensive Architecture. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, 1999.

[21] S. Idreos, F. Groffen, N. Nes, S. Manegold, K. S. Mullender, and M. L. Kersten. MonetDB: Two Decades of Research in Column-oriented Database Architectures. *IEEE Data Engineering Bulletin*, 35(1):40–45, 2012.

[22] S. Idreos, M. L. Kersten, and S. Manegold. Database Cracking. In *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*, 2007.

[23] S. Idreos, S. Manegold, H. Kuno, and G. Graefe. Merging What's Cracked, Cracking What's Merged: Adaptive Indexing in Main-Memory Column-Stores. *Proceedings of the VLDB Endowment*, 4(9):586–597, 2011.

[24] M. F. Ionescu. Optimizing Parallel Bitonic Sort. In *Proceedings of the International Parallel Processing Symposium (IPPS)*, pages 303–309, 1997.

[25] B. Jacob, S. Ng, and D. Wang. *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[26] J. Jeddeloh and B. Keeth. Hybrid memory cube new DRAM architecture increases density and performance. In *Proceedings of the Symposium on VLSI Technoogy (VLSIT)*, pages 87–88, 2012.

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

[28] Y. O. Koçberber, B. Grot, J. Picorel, B. Falsafi, K. T. Lim, and P. Ranganathan. Meet the walkers: accelerating index traversals for in-memory databases. In *Proceedings of the Annual IEEE/ACM*

International Symposium on Microarchitecture (MICRO), pages 468–479, 2013.

[29] D. Koch and J. Torresen. FPGASort: a high performance sorting architecture exploiting run-time reconfiguration on fpgas for large problem sorting. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, pages 45–54, 2011.

[30] S. Kumar, A. Shriraman, V. Srinivasan, D. Lin, and J. Phillips. SQRL: hardware accelerator for collecting software data structures. In *Proceedings of the International Conference on Parallel Architectures and Compilation (PACT)*, pages 475–476, 2014.

[31] J.-G. Lee, G. K. Attaluri, R. Barber, N. Chainani, O. Draese, F. Ho, S. Idreos, M.-S. Kim, S. Lightstone, G. M. Lohman, K. Morfonios, K. Murthy, I. Pandis, L. Qiao, V. Raman, V. K. Samy, R. Sidle, K. Stolze, and L. Zhang. Joins on Encoded and Partitioned Data. *Proceedings of the VLDB Endowment*, 7(13):1355–1366, 2014.

[32] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. J. Dally, and M. Horowitz. Smart Memories: a modular reconfigurable architecture. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*, pages 161–171, 2000.

[33] J. F. Martínez, R. Cumplido-Parra, and C. F. Uribe. An FPGA-based parallel sorting architecture for the Burrows Wheeler transform. In *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, 2005.

[34] Micron Technology. *1Gb: x4, x8, x16 DDR3 SDRAM*, 2006.

[35] O. Mutlu and T. Moscibroda. Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors. In *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 146–160, 2007.

[36] K. J. Nesbit, N. Aggarwal, J. Laudon, and J. E. Smith. Fair Queuing Memory Systems. In *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 208–222, 2006.

[37] 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.

[38] E. Petraki, S. Idreos, and S. Manegold. Holistic Indexing in Main-memory Column-stores. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2015.

[39] H. Pirk, E. Petraki, S. Idreos, S. Manegold, and M. L. Kersten. Database cracking: fancy scan, not poor man's sort! In *Proceedings of the International Workshop on Data Management on New Hardware (DAMON)*, pages 1–8, 2014.

[40] D. Porobic, E. Liarou, P. Tözün, and A. Ailamaki. ATraPos: Adaptive transaction processing on hardware Islands. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 688–699, 2014.

[41] I. Psaroudakis, M. Athanassoulis, and A. Ailamaki. Sharing data and work across concurrent analytical queries. *Proceedings of the VLDB Endowment*, 6(9):637–648, 2013.

[42] S. Pugsley, J. Jestes, R. Balasubramonian, V. Srinivasan, A. Buyuktosunoglu, A. Davis, and F. Li. Comparing Different Implementations of Near Data Computing with In-Memory MapReduce Workloads. *IEEE Micro*, 34(4):44–52, 2014.

[43] L. Qiao, V. Raman, F. Reiss, P. J. Haas, and G. M. Lohman. Main-memory scan sharing for multi-core CPUs. *Proceedings of the VLDB Endowment*, 1(1):610–621, 2008.

[44] J. Rao and K. A. Ross. Making B+- trees cache conscious in main memory. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 475–486, 2000.

[45] S. Rixner, W. J. Dally, U. J. Kapasi, P. R. Mattson, and J. D. Owens. Memory access scheduling. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*, pages 128–138, 2000.

[46] N. Satish, M. Harris, and M. Garland. Designing efficient sorting algorithms for manycore GPUs. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 1–10, 2009.

[47] A. Satoh and T. Inoue. ASIC-hardware-focused Comparison for Hash Functions MD5, RIPEMD-160, and SHS. *Integration, the VLSI Journal*, 40(1):3–10, 2007.

[48] 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 *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*, pages 97–108, 2014.

[49] A. Shatdal, C. Kant, and J. F. Naughton. Cache Conscious Algorithms for Relational Query Processing. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 510–521, 1994.

[50] L. Sidirourgos and M. L. Kersten. Column Imprints: A Secondary Index Structure. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 893–904, 2013.

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

[52] T. Willhalm, N. Popovici, Y. Boshmaf, H. Plattner, A. Zeier, and J. Schaffner. Simd-scan: Ultra fast in-memory table scan using on-chip vector processing units. *Proceedings of the VLDB Endowment*, 2(1):385–394, 2009.

[53] L. Woods, Z. István, and G. Alonso. Ibex - An Intelligent Storage Engine with Support for Advanced SQL Off-loading. *Proceedings of the VLDB Endowment*, 7(11):963–974, 2014.

[54] L. Wu, R. J. Barker, M. A. Kim, and K. A. Ross. Navigating Big Data with High-throughput, Energy-efficient Data Partitioning. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*, pages 249–260, 2013.

[55] L. Wu, A. Lottarini, T. K. Paine, M. A. Kim, and K. A. Ross. Q100: the architecture and design of a database processing unit. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*

(ASPLOS), pages 255–268, 2014.

[56] 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.

[57] D. P. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski. TOP-PIM: throughput-oriented programmable processing in memory. In *Proceedings of the International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, pages 85–98, 2014.

[58] M. Zukowski, S. Héman, N. Nes, and P. A. Boncz. Super-Scalar RAM-CPU Cache Compression. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, page 59, 2006.

[59] M. Zukowski, S. Héman, N. J. Nes, and P. Boncz. Cooperative Scans: Dynamic Bandwidth Sharing in a DBMS. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 723–734, 2007.