Editors: Volodymyr Kindratenko, University of Illinois, kindr@ncsa.uiuc.edu | Pedro Trancoso, Univ. of Cyprus, pedro@cs.ucy.ac.cy

Hardware Acceleration for Query Processing: Leveraging FPGAs, CPUs, and Memory

Oriol Arcas-Abella | Barcelona Supercomputing Center and Universitat Politècnica de Catalunya
Adrià Armejach | Barcelona Supercomputing Center
Timothy Hayes | Barcelona Supercomputing Center and Universitat Politècnica de Catalunya
Gorker Alp Malazgirt | Bogazici University, Istanbul
Oscar Palomar and Behzad Salami | Barcelona Supercoming Center and Universitat Politècnica de Catalunya
Nehir Sonmez | Barcelona Supercomputing Center

Database management systems (DBMSs) have become an indispensable tool for industry, government, and academia, forming a significant component of modern datacenters and handling various tasks such as online analytical processing, data mining, e-commerce, and scientific analysis. The rate at which new data is being produced grows every year: in 2000, researchers estimated 800,000 petabytes of data were stored collectively in the world, and in 2020, they estimate the number will jump to 35 zettabytes (1 zettabyte is 1000^7 bytes, or 1 trillion Gbytes). Given this exponential growth, there's pressure on software and hardware developers to create datacenters that can cope with increasing data storage requirements. Here, we look at the organization of a modern relational DBMS and propose optimizations for storage access, memory, and CPU.

Our proposals represent distinct techniques, but we envision a unified system in which these optimizations can complement one another and work holistically. Figure 1 shows a high-level overview of our conceived platform. Reconfigurable architectures such as field-programmable gate arrays (FPGAs) are good candidates for hardware accelerators; our

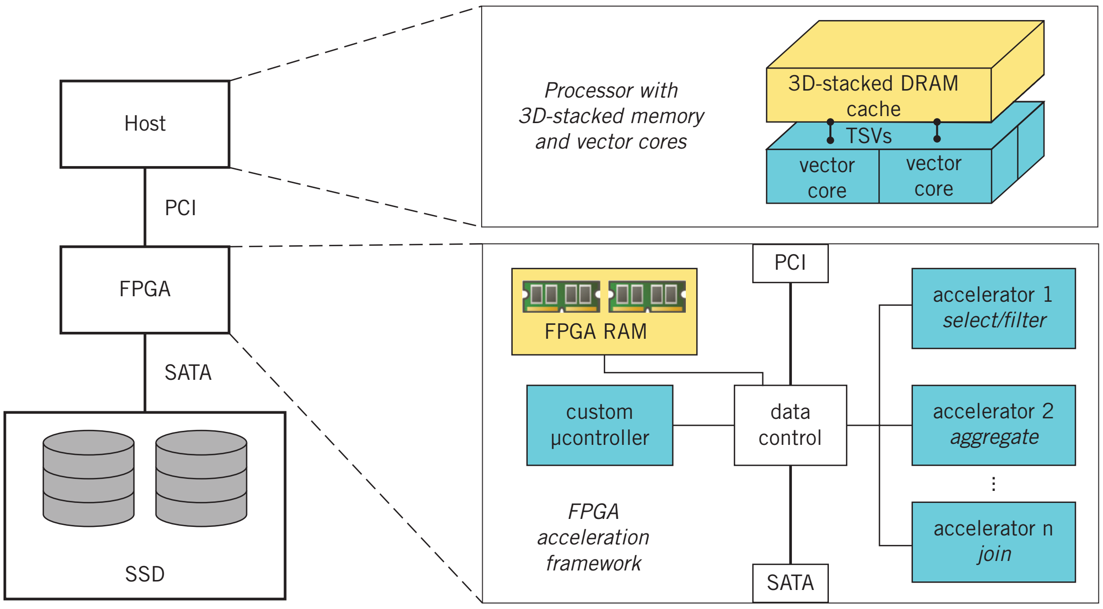


Figure 1. High-level system overview. The host contains the CPU with vector extensions and 3D-stacked DRAM. Its duty is to run the database management system (DBMS) software and selectively offload tasks to the acceleration framework. This framework has a high-bandwidth channel with the disk system and routes relevant data directly to the accelerator units or bypasses the request to the host.

objective is to bring this specialized computation closer to storage devices to reduce the high overheads associated with data movement. FPGAs not only provide massive fine-grain parallelism, but they're also resilient to irregular computation and storage access patterns. In the CPU, we investigate the capability of single-instruction, multiple-data (SIMD) multimedia instruction extensions to accelerate *sort*, an important database operator with a significant processing overhead. We find that current SIMD instruction set architectures (ISAs) lack the semantics to efficiently capture much of the available data-level parallelism (DLP). Based on this observation, we propose new SIMD instructions to capture irregular DLP and apply this to a novel sorting algorithm. In the memory hierarchy, we focus on 3D stacked DRAM caches, which allows for the integration of large amounts of memory with the processor die, increasing memory bandwidth and lowering access latency for the stacked DRAM. We also investigate schemes to improve data placement in such caches to improve performance.

FPGA-Based Acceleration Framework

At the Barcelona Supercomputing Center, we're developing an FPGA-based acceleration framework that contains a dynamic dataflow-like network between the disk system and various accelerators, implementing database primitives. Building on this framework, we're investigating the potential of running entire complex DBMS queries precompiled on the board as well as the applicability of using high-level synthesis languages to allow novice programmers to create their own custom queries to execute directly on the framework.

Near Data Processing

As the amount of data to manage grows, there's an increasing strain on DBMS software to meet its throughput and latency requirements. Data movement is a point of concern for modern datacenters, and excessive movement can degrade both throughput and latency in addition to increasing average energy cost per query. Near data processing aims to bring the computation closer to where the data resides so that more operations can be completed, avoiding nonessential data movement. Although this isn't the first study to process queries directly from disk using FPGA technology,[1] our focus is on designing state-of-the-art accelerators inside such an infrastructure. In this article, we specifically focus on our hash join engine.

Figure 1 shows our FPGA acceleration framework, which is self-contained on a VC709 FPGA

Further performance improvements are possible if the framework provides acceleration for user-specific queries that require customized processing not easily expressible through default operators.

development board with a Virtex-7 FPGA and 8 Gbytes of RAM. The acceleration framework is in close proximity to a pair of 250-Gbyte solid state drives (SSDs). The host accesses the acceleration framework through a software API. In a typical SQL DBMS, a query execution planner does the job of establishing an ordered set of steps to access and process data. We adapted this planner to appropriately use our infrastructure and to send and receive data and commands through a fast PCI link to communicate with the acceleration framework's custom microcontroller. Depending on the query plan that the DBMS establishes, the host machine effectively programs the microcontroller to utilize the accelerator modules. This specialized microcontroller calculates the regions of the disk affected and instructs the appropriate accelerators to retrieve and process the data. The results can be stored back to the disk or sent to the host.

Query-Processing Accelerators

We use Xilinx Vivado high-level synthesis (HLS) to perform the selection/filtering operations. For this purpose, we designed a compute engine that can process arbitrary-length data types in parallel. It takes rows as inputs and a condition such as the BETWEEN operator and then applies a selection/filtering operation on the desired columns as they're read, filtering out unwanted data from further processing, thus reducing the size of the input set.[2]

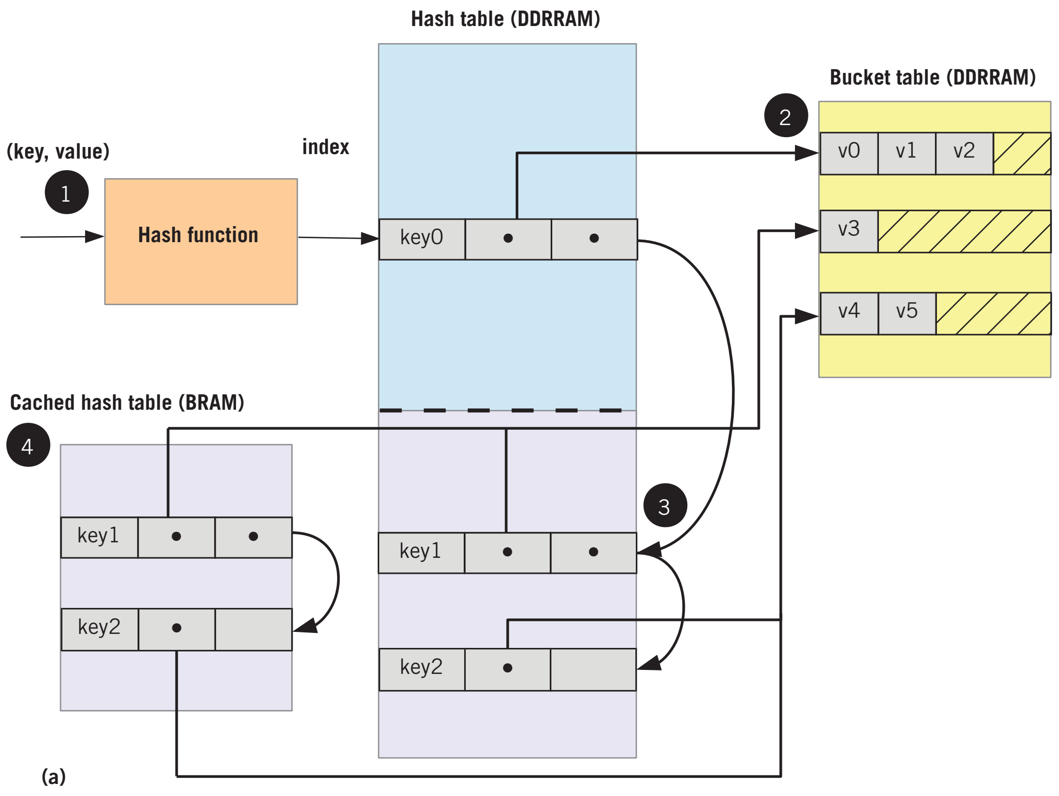
To support join acceleration, we propose a novel architecture of hash join, one of the most commonly used and time-consuming DBMS operations.[3] Hash collisions—that is, multiple distinct keys resolving to the same location in the hash table—are a critical issue that can be detrimental to good performance. Pointer chasing is a common method for resolving collisions, but in DRAM based engines, the memory wall can significantly undermine performance. To overcome the memory wall, one feasible solution is to use FPGAs' low-latency local block RAMs (BRAMs). However, due to their small size, there's no guarantee that they can store an entire hash table. Therefore, we introduce a novel DRAM-based hash join engine that transforms BRAMs into low-latency cache. The entries of the main hash table are cached in the BRAM using a direct mapped methodology. Figure 2 shows the engine's block diagram.

We developed the fully pipelined version of the proposed hash join architecture using BlueSpec SystemVerilog HLS and performed experiments on a selection of TPC-H queries. Depending on the BRAM hit rate, experimental results show an up to 2.8× improvement in the number of cycles over an FPGA baseline without BRAM and up to 82.3× over the hash join engine of PostgreSQL running on RAMDisk. Figure 2b shows detailed experimental results for queries q03, q12, q13, and q14 of TPC-H running three dataset sizes (1 Gbyte, 10 Gbytes, and 100 Gbytes).[3]

Even though we developed accelerators for fundamental database primitives, further performance improvements are possible if the framework provides acceleration for user-specific queries that require customized processing not easily expressible through default operators. As the optimal accelerator would be very much workload dependent, the implementation would depend on the DBMS user, who's typically not an expert in FPGA development. We studied four HLS tools that could help nonexpert users accelerate common database operations.[4] Choosing the right design languages and tools can greatly affect final hardware performance. Our results show that high-level hardware description languages can produce accelerators with similar performance to manually designed hardware. In our study, we use four accelerators for database operations; programming experience varies greatly from one design paradigm to the other. Advanced HLS tools, which can generate hardware from pure C code, are still in the initial stages, although they look highly promising as a solution for nonexpert users to generate high-quality designs with little to no effort.

Novel SIMD Instructions for Sorting

FPGA-based acceleration can be used to offload many tasks from the main processing unit, but the CPU still plays a crucial role. Modern CPUs can yield very high performance if the algorithm is carefully tuned and thoroughly leverages architectural features. As a case study, we focus on sorting and vector SIMD extension, one of the most



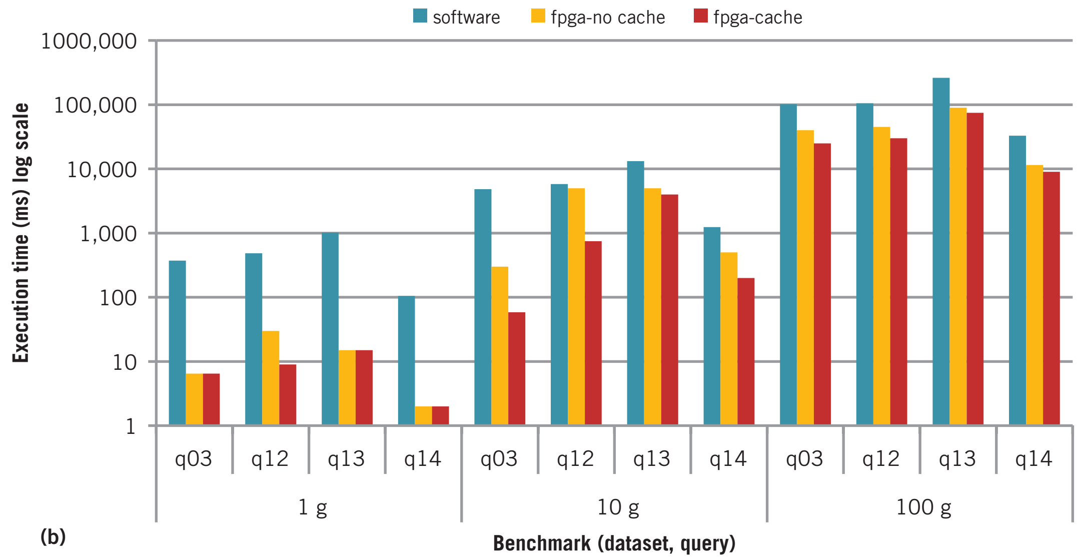


Figure 2. Hash join engine: (a) overview showing the (1) compute hash index using a pipelined hash function, (2) grouping values of each individual key into the bucket table, (3) collisions resolved with a pointer chasing method, and (4) block RAM (BRAM) working as a cache of the main hash table; (b) execution time of cache-employed version compared to the baseline and software.

efficient and powerful features available in commodity processors and essential to achieving high performance on the CPU.

Sorting is a cornerstone of relational databases that's frequently used in the construction of composite functions. Given this operation's importance, it's

Previous attempts at vectorizing radix sort with a SIMD ISA have been only semisuccessful.

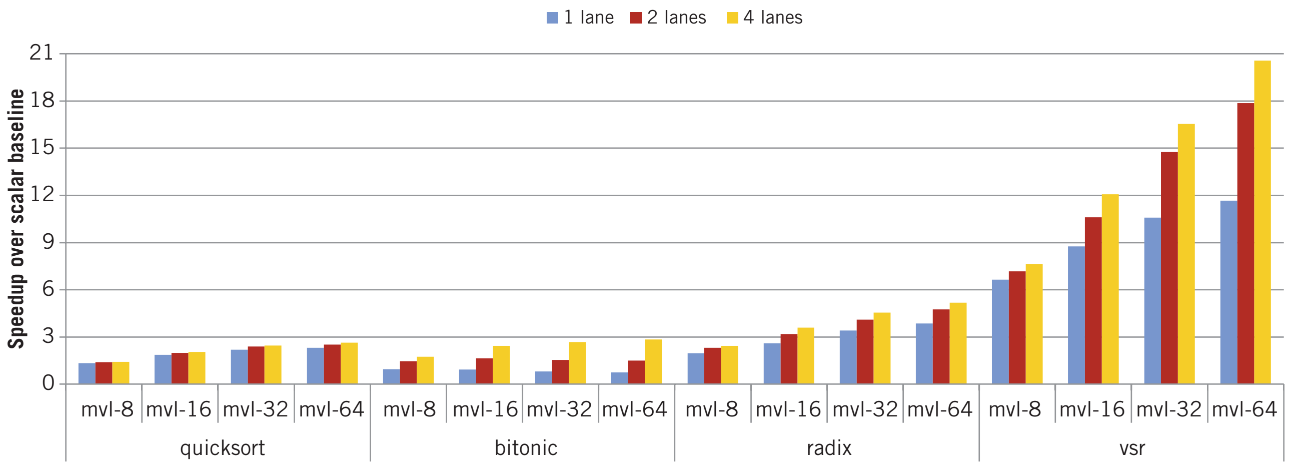


Figure 3. Speedup over scalar algorithm for vectorized versions of quicksort, bitonic mergesort, radix sort, and our novel VSR algorithm while varying the maximum vector length (mvl) and vector lanes. VSR sort exhibits good scalability and achieves maximum speedups up to 20.6× over a scalar baseline and on average performs 3.4× better over a standard SIMD radix sort when run on the same hardware configuration.

vital to have a high-performance algorithm. By leveraging SIMD extensions in modern microprocessors, sorting could potentially take advantage of explicit DLP and provide more bandwidth between functional units and memory hierarchy. SIMD extensions have become ubiquitous in modern microprocessors, but current extensions lack essential features, which in turn inhibits scalability and speed when vectorizing sorting algorithms. We anticipate that SIMD extensions will grow both in width and functionality in future generations, and our contribution could help guide this evolution in a clear and positive way.

For relational databases, radix sort is arguably the best choice of sorting algorithm. Primary and foreign keys have cardinalities and radixes well below their respective data type's limit. Previous attempts at vectorizing radix sort with a SIMD ISA have been only semisuccessful. The reasons are numerous but boil down to two fundamental problems: radix sort has strict stability requirements, meaning it's a serial algorithm in nature, and actions within standard SIMD operations should be independent of one another. This complicates the process of updating monolithic bookkeeping structures, such as histograms, as there are often conflicting gather-modify-scatter operations. To circumvent this irregular DLP, significant transformations must be made to radix sort to vectorize it—for example, the input needs to be read incrementally using an inefficient strided memory access pattern so that individual elements of a vector register operate on contiguous partitions, and bookkeeping structures must be replicated for every element in a vector register to avoid update conflicts.

Based on these observations, we propose VSR sort,[5] a novel way of vectorizing radix sort without inefficient transformations. We introduce two new instructions into our SIMD ISA. The *vector prior instances* (VPI) instruction lets us load the input and store the output using an efficient unit-stride (contiguous) memory access pattern without breaking radix sort's strict stability requirements. In essence, it corrects problematic input values that have a serial dependency and ensures they end up in adjacent locations of the output. *Vector last unique* (VLU) can be used to update nonreplicated bookkeeping structures while avoiding gather-modify-scatter conflicts. Using nonreplicated bookkeeping structures lets us grow them considerably and consequently process a larger subset of bits per pass, thereby reducing the algorithm's runtime. The full semantics and implementation of these instructions appear elsewhere.[5]

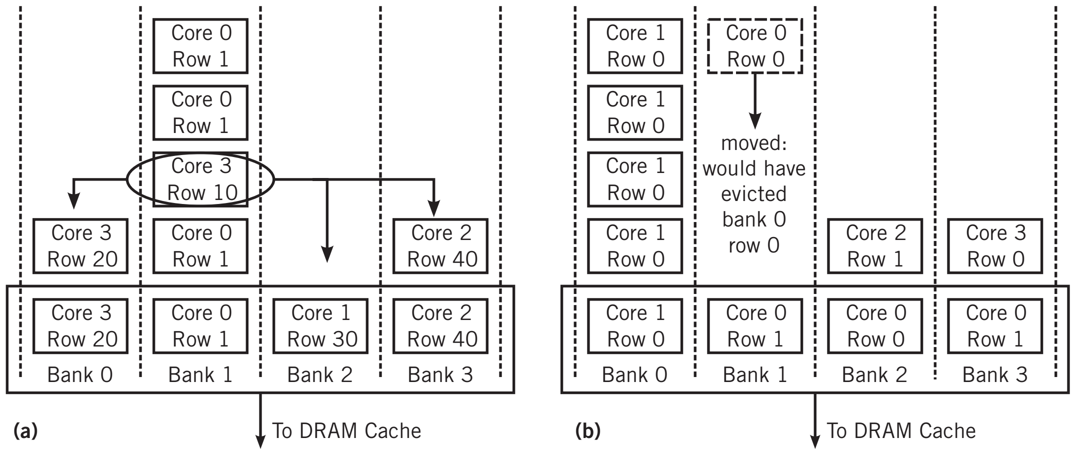


Figure 4. Performance pathologies present in DRAM caches: (a) row-buffer interference and (b) utility interference.

Using a custom simulation framework, we can perform significant sensitivity experiments on VSR sort and three diverse SIMD sorting algorithms suitable for relational databases: quicksort, bitonic mergesort, and radix sort. We take into account our predictions of width and functionality in future microprocessor generations. Figure 3 shows a comparison of each algorithm. As a metric, we used speedup over a scalar baseline, changing two critical parameters: the maximum vector length (mvl), which is the number of elements in a SIMD register, and lanes, which refers to the number of redundant functional units available to a single SIMD instruction. A configuration with a single lane resembles a purely pipelined functional unit implementation, like that of the CRAY-1. We find that all the prior work suffers from bottlenecks and scalability problems. VSR sort exhibits good scalability and achieves maximum speedups up to 20.6× over a scalar baseline and on average performs 3.4× better over a standard SIMD radix sort when run on the same hardware configuration. We feel that VSR sort is an excellent candidate to implement sort primitives in a relational database.

Leveraging Emerging Memory Technologies

Due to the ever increasing computational throughput of chip multiprocessors, off-chip memory has become a performance-limiting factor due to limited pin count scalability. Current off-chip memory bandwidth capabilities aren't sufficient to meet the demands of modern servers running data-demanding workloads such as those employing relational databases.[6] However, die-stacking technology enables tightly coupled integration of DRAM with the processor die, providing hundreds of megabytes to several gigabytes of storage that can be accessed at an unprecedented bandwidth with latencies that are significantly lower than those needed to access off-chip memory. Unfortunately, the amount of storage that this technology offers still isn't sufficient to cope with memory capacity requirements, which are often one or two orders of magnitude higher.[7] For this reason, researchers have been exploring the use of stacked DRAM as a large last-level cache.

We've analyzed current DRAM cache proposals and identified different performance pathologies that arise due to interleaved accesses from different cores at the DRAM banks. Figure 4 shows different queued requests for each bank, and for each request, the requesting core and the targeted row (page) within the DRAM bank where the data is known to be present.

Figure 4a illustrates a case of row-buffer interference where core$_0$ performs several accesses over the same DRAM row (page) on bank$_1$, for which the access latency is low due to row-buffer hits. However, there's a request to a different row from core$_3$ that needs to be served, destroying this row-buffer locality. We want to avoid any interference from other cores that could prematurely close the row-buffer just to service a single request, having to open again the previous row to finish reading it.

Figure 4b illustrates a case of utility interference. In the example, bank$_0$ contains a page accessed by core$_1$ with high spatial locality, from which many blocks are useful (that is, demanded). The rest of the cores access pages with low

Banks with rows that present low spatial locality are likely to generate more evictions, but these evicted pages are cheaper to fetch back and have a lower impact on the overall cache hit ratio.

spatial locality, suggesting a lower number of useful blocks. If the incoming request in $bank_1$, which was a cache miss, had been allocated in $bank_0$, then the page with high locality and block usage would have been evicted even if there were better replacement candidates in other banks. We want to protect cores that would experience significant performance improvements from interfering cores that might hinder these improvements by thrashing the cache, evicting highly reused blocks, and undermining cache utility.

In the context of database workloads, memory bandwidth and access latency are critical to achieving good performance. Moreover, in such workloads, abundant spatial locality is present in a significant number of common primitives, such as sequentially reading from a table. While a DRAM cache has the potential to be a great performance booster for these kinds of workloads, the performance pathologies we described can degrade potential improvements. As Figure 4 suggests, these pathologies can be mitigated with better data placement within the cache. Our proposal aims at providing this functionality to improve DRAM cache performance.[8]

In particular, our proposal identifies cores experiencing high spatial locality and dynamically modifies the DRAM cache replacement policy to allocate their pages to a subset of banks. This data placement policy reduces row-buffer interference by avoiding the closure of row-buffers just to service a single access in the common case: high spatial locality implies high row-buffer locality. Similarly, allocations for cores with poor spatial locality are placed on another subset of the banks, where row-buffer locality is less penalized because it's already low. In addition, utility interference is also reduced because rows with a high block demand won't be replaced by rows with low demand. Banks with rows that present low spatial locality are likely to generate more evictions, but these evicted pages are cheaper to fetch back and have a lower impact on the overall cache hit ratio.

We evaluate our proposal using an architectural simulator that models a chip multiprocessor (CMP) with eight out-of-order cores and a three-level cache hierarchy, which closely resembles a server grade Sandy Bridge Intel processor. To model the stacked and off-chip DRAM memory, we integrated a detailed main memory simulator and configured our DRAM cache to have a size of 2 Gbytes and four memory channels. As a representative data analytics workload, we use a set of queries (q9, q13, q15, and q16) from the TPC-H benchmark running on a modern column-store database engine with a dataset that exceeds 100 Gbytes. Our proposal yields a 24.5 percent performance improvement over a system without stacked DRAM and outperforms a state-of-the-art DRAM cache proposal[9] by 10 percent.

In this work, we proposed hardware improvements for DBMSs that use emerging technologies: FPGAs, future SIMD support, and die-stacking DRAM. We anticipate that applying these redesigns to all levels of the system will result in datacenters that can provide a manyfold increase in throughput. One of our goals is to bridge the gap between these specialized technologies and the nonexpert end user.

Although our work currently comprises several disparate ideas, we ultimately want to bring these together in a unified system and allow them to cooperate harmoniously. Vector SIMD extensions are a good fit for certain key algorithms, but performance is typically limited by the available memory bandwidth. A die-stacked DRAM cache provides abundant memory bandwidth that can unlock additional performance for vector extensions while providing the additional benefits described here. Although these two technologies work well in tandem, data movement between storage systems and processing units is a major bottleneck that will be exacerbated as the amount of data to be processed grows. To minimize data movement, we propose using FPGA technology for near data processing and to tackle certain operations that map well to these devices, reducing data movement to computational resources. ■

Acknowledgments

The research leading to these results has received funding from the EU's Seventh Framework Programme

(FP7/2007- 2013) under the AXLE project (grant 318633). This is a collaborative project where the authors have contributed equally. They have been listed in alphabetical order by surname. Gorker Alp Malazgirt undertook this work with a Severo Ochoa grant at the Barcelona Supercomputing Center.

References

1. T.C. Scofield et al., "XtremeData dbX: An FPGA-Based Data Warehouse Appliance," *Computing in Science & Eng.*, vol. 12, no. 4, 2010, pp. 66–73.
2. G.A. Malazgirt et al., "Accelerating Complete Decision Support Queries Through High-Level Synthesis Technology," *Proc. ACM/SIGDA Int'l Symp. Field-Programmable Gate Arrays*, 2015, p. 277.
3. B. Salami, Behzad et al., "HATCH: Hash Table Caching in Hardware for Efficient Relational Join on FPGA," *Proc. Ann. Int'l Symp. Field-Programmable Custom Computing Machines* (FCCM), 2015, p. 163.
4. O. Arcas-Abella et al., "An Empirical Evaluation of High-Level Synthesis Languages and Tools for Database Acceleration," *Proc. Int'l Conf. Field Programmable Logic and Applications* (FPL), 2014, pp. 1–8.
5. T. Hayes et al., "VSR Sort: A Novel Vectorised Sorting Algorithm & Architecture Extensions for Future Microprocessors," *Proc. IEEE 21st Int'l Symp. High Performance Computer Architecture* (HPCA), 2015, pp. 26–38.
6. B.M. Rogers et al., "Scaling the Bandwidth Wall: Challenges in and Avenues for CMP Scaling," *Proc. 36th Ann. Int'l Symp. Computer Architecture* (ISCA), 2009, pp. 371–382.
7. M. Ferdman et al., "Clearing the Clouds: A Study of Emerging Scale-Out Workloads on Modern Hardware," *Proc. 17th Int'l Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS), 2012, pp. 37–48.
8. A. Armejach et al., "Tidy Cache: Improving Data Placement in Die-stacked DRAM Caches," to appear in *Proc. Int'l Symp. Computer Architecture and High Performance Computing* (SBAC-PAD), 2015.
9. D. Jevdjic et al., "Die-Stacked DRAM Caches for Servers," *Proc. 40th Ann. Int'l Symp. Computer Architecture* (ISCA), 2013, pp. 404–415.

Oriol Arcas-Abella is a PhD candidate in computer architecture at the Universitat Politècnica de Catalunya, where he obtained his BS and MS in computer science. His research at the Barcelona Supercomputing Center is focused on soft microarchitecture prototyping on reconfigurable devices, including hardware debugging and verification techniques and high-level design. Contact him at oriol.arcas@bsc.es.

Adrià Armejach is a postdoctoral researcher at the Barcelona Supercomputing Center. His interests include computer architecture, parallel computing, memory systems, and performance evaluation. Armejach received a PhD in computer architecture from the Universitat Politècnica de Catalunya. Contact him at adria.armejach@bsc.es.

Timothy Hayes is a third-year PhD candidate in the field of computer architecture at the Barcelona Supercomputing Center. His research interests include SIMD models of computation, workload characterization, and high-performance microarchitecture development. Contact him at timothy.hayes@bsc.es.

Gorker Alp Malazgirt is a PhD candidate in computer engineering at Bogazici University, Istanbul. His research interests are computer architectures, reconfigurable computing, and metaheuristics. Malazgirt received an MSc in system-on-chip design from Lund University, Sweden. Contact him at alp.malazgirt@boun.edu.tr.

Oscar Palomar is part of the parallel paradigms group at the Barcelona Supercomputing Center. His research interests involve low-power vector architectures and energy minimization. Palomar received a PhD in computer architecture from the Universitat Politècnica de Catalunya. Contact him at oscar.palomar@bsc.es.

Behzad Salami is pursuing a PhD in computer architecture at the Universitat Politècnica de Catalunya while doing research at the Barcelona Supercomputing Center. His research interests include reconfigurable computing, big data processing, and high-performance computing. Contact him at behzad.salami@bsc.es.

Nehir Sonmez is a postdoctoral researcher in the parallel paradigms research group at the Barcelona Supercomputing Center. His research interests include reconfigurable computing, multicore computer architecture, and database acceleration for big data and transactional memory. Sonmez received a PhD in computer engineering from the Universitat Politècnica de Catalunya. Contact him at nehir.sonmez@bsc.es.