Short paper

Optimizing Data Movement with Near-Memory Acceleration of In-memory DBMS

Donghun Lee[†], Minseon Ahn[†], Jungmin Kim[†], Kangwoo Choi[†], Oliver Rebholz[‡]

Andrew Chang[§], Jongmin Gim[§], Jaemin Jung[§], Vincent Pham[§], Krishna Malladi[§], Yang Seok Ki[§]

[†] SAP Labs Korea [‡] SAP SE
{dong.hun.lee, minseon.ahn, jungmin.kim, kangwoo.choi, oliver.rebholz}@sap.com
[§] Samsung Semiconductor Inc.
{andrew.c1, gim.jongmin, j.jaemin, tung1.pham, k.tej, yangseok.ki}@samsung.com

ABSTRACT

Despite the increase of memory capacity and CPU computing power, memory performance remains the bottleneck of in-memory DBMS due to ever-increasing data volumes and application demands. Since the scale of data workload has outpaced traditional CPU caches and memory bandwidth, it is essential to optimize data movement from memory to computing units. In this work, we present a near-memory Database Accelerator (DBA) framework that offloads data-intensive database operations via or to a near-memory computation engine. DBA's system architecture includes a DBA software module/driver and memory module with DBA engine. We build a Proof-of-Concept (PoC) of DBA using FPGAs with attached DIMMs, and then conduct an experimental evaluation.

1 INTRODUCTION

Low cost and high capacity of DRAM accelerated the market of in-memory database management systems (IMDBMS). The latest IMDBMS architecture capable of running both Online Transactional Processing (OLTP) and Online Analytical Processing (OLAP) applications in a single system removes the data redundancy and provides higher performance and efficiency with lower total cost ownership (TCO) [9]. However, with ever-increasing data volumes and application demands, memory performance becomes the main performance bottleneck of IMDBMSs. Our study with OLTP/OLAP applications shows that performance can be bound by expensive data-intensive operations like table scan and aggregation of OLAP workloads. These data-intensive operations have very little data reuse for further computation but consume more than 50% of CPU resources and almost all memory bandwidth in many cases. The other mission critical workloads suffer from cache conflicts (or cache thrashing) and memory bandwidth bottleneck. Therefore, it is essential to optimize data movement from memory to computing units.

The best way to optimize this data movement in IMDBMS would be to process these data-intensive operations within memory devices. Instead of transferring the whole data to computing units, forwarding the filtered results to the next processing step could minimize the overhead. Near-storage computing [3, 4] tries to accelerate the data-intensive operations by minimizing the data transfer overhead from storage to processing nodes or CPU. However, this research does not deliver byte addressability and much lower latency necessary for IMDBMS. Previous work to accelerate database operations using FPGA [7, 8, 10] and GPGPU [5, 6] shows an order of magnitude performance gain in compute-intensive operations. However, these approaches show a smaller gain in data-intensive operations because of the data movement overhead [1]. Even Hybrid CPU-FPGA approaches [7, 10] require data movement from host memory to accelerator computing units which has a high memory bandwidth overhead.

Processing-In-Memory (PIM) approaches like UPMEM [2] advance the concept of near-memory computing but are still in early stage. Furthermore, the data needs to be reformatted to utilize the processing units, thus the existing data structure cannot be reused directly.

In this paper, we propose near-memory database accelerator (DBA) to offload data-intensive operations of IMDBMS to memory devices. By placing simple arithmetic units near DRAM within memory devices like DIMMs, we 1) save CPU cycles for data-intensive operations, 2) avoid cache thrashing among threads, and 3) reduce the host memory bottleneck. We implement our proof-of-concept (PoC) system using FPGAs with attached DIMMs. Its DBA kernel is designed to perform parallel comparisons in a SIMD manner fully utilizing internal memory bandwidth. Our evaluation shows that near-memory DBA has more than 2 times performance improvement in OLTP workloads when offloading the data-intensive operations. Finally, we discuss the obstacles to embody the approach in real memory devices.

2 BACKGROUND

2.1 Motivational example

Figure 1 shows the performance degradation of OLTP workload by the interference from the scan workloads on OLAP data in the server having 4 sockets and 72 physical cores. The two workloads managed by two separate processes access the different sets of data but compete with each other for limited hardware resources like CPU, cache and memory bandwidth. As the number of scan threads increases, the CPU resources allocated for OLTP workloads are reduced, thus the throughput of OLTP workloads decreases.

It is quite common to apply SIMD instructions to data-intensive operations like scan within a DBMS [11, 12], as SIMD performs the same operation on multiple data points simultaneously exploiting data level parallelism. We observe that the OLTP workloads show a larger performance degradation, when the scan operation is implemented with SIMD commands like AVX2 or AVX 512 because of much higher memory bandwidth usage. As

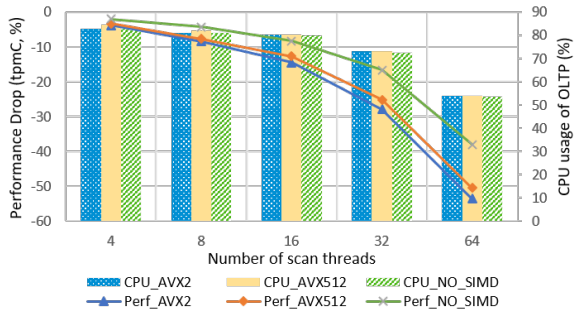

Figure 1: OLTP throughput by number of concurrent scans

Table 1: Memory bandwidth usage (%) by scan workloads

#Threads	AVX2	AVX512	No_SIMD
4	11.3	6.0	0.8
8	21.8	12.3	1.3
16	41.3	23.0	2.8
32	73.8	46.3	5.5
64	94.8	85.3	11.3

shown in Table 1, 64 scan threads consume almost all memory bandwidth of 4 sockets with SIMD, only 12% of memory bandwidth is consumed without SIMD. Interestingly, there is no difference in CPU usage between SIMD and NO-SIMD, but the OLTP throughput shows a larger performance degradation with SIMD. In Figure 1, with 64 scan threads, the CPU usage by OLTP decreased by 30% but the OLTP throughput decreases by about 40% without SIMD and more than 50% with SIMD. This supports our claim that the larger memory bandwidth usage by data-intensive workloads degrades OLTP performance more.

2.2 Scan operation in In-memory DBMS

Recent IMDBMSs are designed to support both OLTP and OLAP workloads and keep the data in the columnar storage for fast read accesses of the tables, storing the majority of data of each column in the read optimized main storage, and maintaining the separate delta storage for optimized writes [9]. The delta storage is periodically merged to the main storage [8]. To reduce the memory footprint (or TCO), the main storage uses dictionary encoding where the distinct values are stored in the dictionary and the individual values are replaced with the corresponding value IDs of the dictionary separately with the bit-packed compression [9]. A scan in IMDBMS reads this value ID array with filter conditions. In this work, the two common scan operations - Range search (having from/to filter conditions) and Inlist search (having a list of filtered values) are offloaded to DBA as they are simple and common data-intensive operations that often consume relatively high CPU usage (5-10% by itself). They include the decompression of value ID (integer) array and return row IDs satisfying the predicates. Offloading only low-level data access operators in the query plans reduces the effort to integrate them with the existing query optimizer.

3 DBA ARCHITECTURE

This section discusses the architecture and design of our proposed near-memory Database Accelerator (DBA). Figure 2 describes the system architecture of DBA. The objective of this work is 1) to demonstrate the offloading feasibility within current ecosystem, 2) to provide a framework to measure the stand-alone DBA engine performance, and more importantly, 3) to study the system impact of our proposal. To remove the unnecessary data movement, database operations are performed by DBA in the device memory where the source data is stored. After DBA completes the operations, the result output is written back to the device memory in FPGA. The host has access to the device memory via memory mapped I/O (MMIO). This eliminates speed and coherency limitations of the PCIe interface from this study and yet leverages the current driver software stack with the OS and the application.

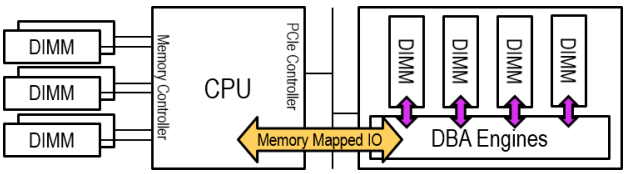

Figure 2: DBA System Architecture

Figure 3 describes the DBA FPGA micro-architecture with functional partitions of host interface, multiple DBA kernels and memory subsystem. The host interface exposes DBA control parameters to the driver that manages offloading from the application API call to the hardware accelerator. Each DBA kernel consists of data prefetcher reading the data, SIMD engine comparing the data with the predicate, and result handler writing the results. The memory subsystem provides the infrastructure to access device memories on the FPGA.

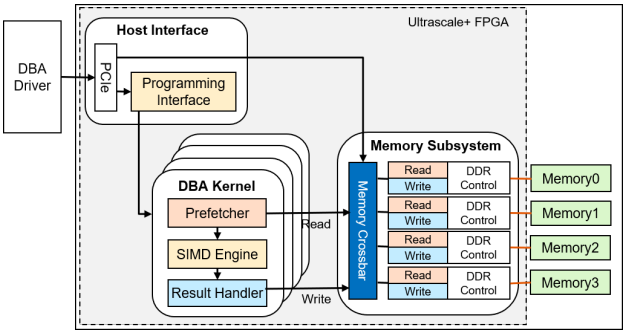

Figure 3: DBA FPGA Micro-Architecture

Internally, DBA kernels read 64B bit-compressed data at a time from the memory. A programmable extractor logic splits the data into multiple values. They are fed into an array of simple processing units and each unit performs a simple comparison independently. The number of parallel units in the array is determined by the number of values in the 64B data so that DBA kernels can keep up with the input data rate. Compared to fixed-length instruction-based processors, each DBA kernel takes the full advantage of parallelism in the data stream due to the flexibility of hardware design. The results are packed into 64B and written back to the device memories. Thus, the data flow is highly optimized for available memory bandwidth.

Figure 4 describes DBA software architecture. Unlike GPU/FPGA accelerators, the DBA engine is located within memory devices. Hence, it allows zero data copy with performance and energy gains. The DBA device driver assigns one DBA engine to a thread

of the IMDBMS application per request. Once offloaded, a thread that requested an offloading yields to free CPU for processing. When offloading is done, DBA driver wakes up the requester to resume.

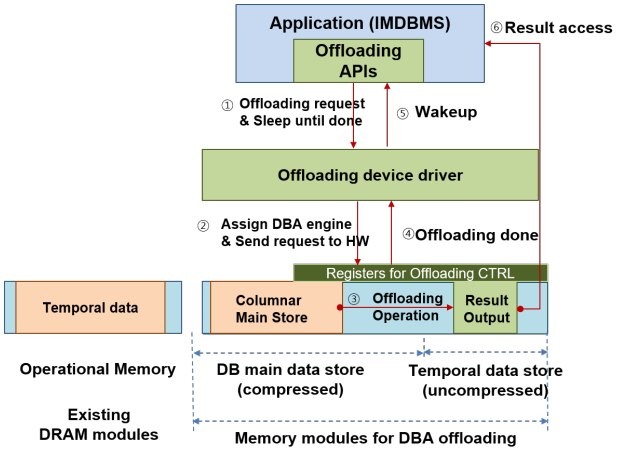

Figure 4: DBA Software Architecture

Normally, applications use a virtual address (VA) to access memory in the host system while the DBA engines access memory with a device physical address (DPA). This implies that the DBA driver is responsible to translate all VA parameters of an offloading request into DPA. The DBA driver first obtains the corresponding system physical address (SPA) by referring to a page table. Then, converting DPA to SPA is trivial because the system BIOS has stored the start SPA of device memory in the Base Address Registers (BAR) of the PCI device at boot time.

4 EVALUATION

4.1 Experimental setup

The system consists of the embedded TPCC benchmark in an IMDBMS and a separate micro-benchmark program to generate scan workloads in a single server as shown in Figure 5.

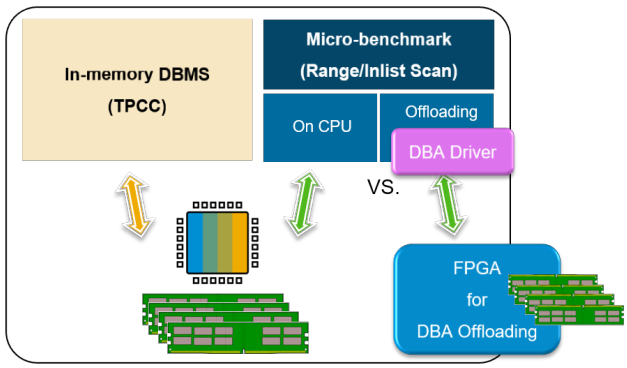

Figure 5: Conceptual diagram

In our experiments, we use the TPCC benchmark for OLTP workload. Its generator is embedded within the IMDBMS engine to remove the communication and session management overhead because the total throughput is usually bound by the session layer, not IMDBMS engine. We want to maximize the throughput (i.e. resource consumption) of the TPCC workload.

The micro-benchmark performs the scan workloads 1) on CPU, or 2) via FPGA. Its data is randomly generated and bit-compressed. The separate data for scans avoids the internal overhead of IMDBMS like locking by two different workloads and enables us to focus on the performance effect by hardware resources. In our experiment, scans read 2 billion bit-compressed integer values and return the row IDs satisfying the filter conditions. When it runs on CPU, the same number of scan threads are bound to each socket to prevent workload skew among the sockets on the 4-socket server (Intel Xeon Gold 6140@2.30GHz, 18 cores and 6 * 64 GB memories per socket). For DBA offloading, we attach one Ultrascale+ FPGA@250MHz per socket and populate 6 scan engines with 4 * 64 GB DDR4 DIMMs @1866MHz per FPGA. The scan data is copied to the memory in each FPGA to emulate that DBA offloading runs within memory devices where the data resides. We compare the performance variation of TPCC workloads and measure the latency and throughput scalability of scan workloads in both options (on CPU vs. on FPGA), while the number of scan threads increases.

4.2 Evaluation results

This section summarizes our DBA PoC evaluation results compared with a state-of-art 4-socket Skylake system having 72 physical cores.

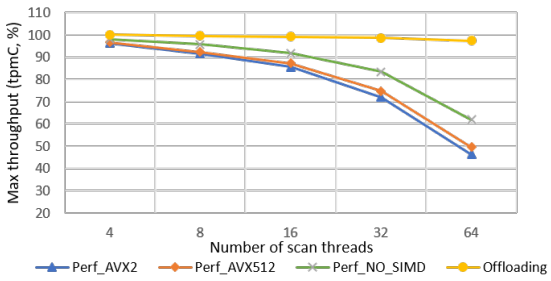

Figure 6: OLTP throughput gain by DBA

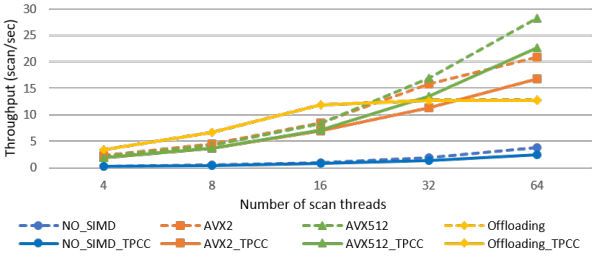

Figure 7: Scan throughputs with/without TPCC

Figure 6 demonstrates the system performance gain of IMDBMS. While TPCC workload runs in the server, the scan micro-benchmark runs on either CPU or DBA with a different number of threads. As a result, DBA offloading shows less performance slowdown as the number of scan threads increases. Therefore, DBA offloading shows 115% better tpmC (transactions per minute) in TPCC workloads when all 64 scan threads are offloaded than when 64 threads use AVX2 on CPU. The results confirm that DBA offloading can alleviate CPU conflict, cache thrashing and memory bandwidth conflict by data-intensive operations.

Table 2: Average latency (sec/scan) of a single scan

Latency of single scan (sec/scan)	On CPU			Offloading
	NO_SIMD	AVX2	AVX512	
	4.16	0.44	0.47	0.29

DBA offloading shows the better performance in scan operation itself, when scans run without OLTP workloads. DBA offloading shows 1.5x better latency (sec/scan) than AVX2 and 14.3x better than NO-SIMD as shown in Table 2.

As for the throughput (scans/sec) scalability, DBA offloading shows quite promising performance as shown in Figure 7. The solid lines represent the throughputs of scans when TPCC workloads are executed concurrently. The dotted lines mean the scan throughputs without TPCC workloads. DBA Offloading shows similar performance regardless of the presence of TPCC workloads, while scan with SIMD/NO-SIMD shows significant performance drop because of the interference from TPCC workloads. DBA offloading outperforms the scan with SIMD/NO-SIMD up to 16 threads and shows similar performance to SIMD scan with 32 threads. With the current implementation, each DBA FPGA has 6 scan engines and 4 DIMM slots. The results show the throughput by DBA offloading is saturated with 16 threads (4 threads per FPGA) because of the limited memory bandwidth of 4 memory channels and resources within the FPGA. Each CPU has 6 DDR channels with 128GB/sec bandwidth while each FPGA has 4 with 60GB/sec. When DBA has the same number of threads, it performs better than CPU running with SIMD. In a SoC (System on Chip) implementation where DBA offloading is embedded in real memory devices, these limitations will be relieved, and the overall performance will be improved further by higher clock frequency or more DBA engines.

In our work, we have the similar performance gain with both range and inlist scans, and similar results regardless of bit-cases used in bit-packed compression [9]. Due to the limited space in this paper, we show only the results of the range scan.

5 DISCUSSION

This research was done using FPGAs with attached DIMMs. The host system accesses the device memory through PCIe MMIO by mapping the device memory in the same address space of the host memory. Even with the slow performance of MMIO in PCIe, the offloading performance is not affected, because our offloading implementation only accesses the local device memory on FPGA once offloading operation starts.

DBA offloading can be implemented on the diverse memory form-factors with their own pros and cons. DIMM-based memory is quite common and very fast, but the memory controller will naturally interleave the data among memory channels. Therefore, even a single value can be crossed on two DIMMs and the DBA driver should handle the data interleaving while processing offloaded operations.

Recently proposed interfaces like CXL(Compute Express Link), Gen-Z and OpenCAPI will enable a new memory pool hosting the columnar main storage in IMDBMS. Although these interfaces introduce a bit higher latency than DIMM, the memory devices are not part of host memory controller pool where data are typically interleaved at 64B granularity. This allows DBA to assume a contiguous data layout in its attached memory and operates without considering data interleaving across memory channels. One more hurdle of DBA offloading would be non-contiguity in the physical address space of the contiguous data in the virtual address space. DBA offloading will provide so-called 'scatter and gather' feature by building a page translation table.

In Clouds, the micro-services of IMDBMS can be spread out among several nodes according to its role. The front-end computing nodes to process the transactions may be easily scaled out, but the storage node cannot be done simply having the same issues on our claim. We believe DBA offloading can contribute to resolving them in Clouds as well.

6 CONCLUSION

We showed that the OLTP-like mission critical workloads can interfere with data-intensive operations like massive scans. We proposed a near-memory database accelerator (DBA) to optimize the data movement and showed that performing the expensive scan operations in the memory devices can alleviate CPU load, cache conflict, host memory bandwidth bottleneck. To confirm its feasibility, we implemented the offloading system using FPGAs with attached DIMMs. Its results show more than 2x performance gain in OLTP workload when offloading the data-intensive operations, and higher or similar throughput scalability with better latency in offloaded scan workloads.

Aggregation is another data-intensive operation in IMDBMS consuming about 20-50% of CPU usage depending on the workloads. While it reads large amounts of data, most of it is rarely referenced again. DBA offloading on aggregation is being investigated as the next target operation.

ACKNOWLEDGMENTS

The authors would like to thank Ismail Oukid for his valuable feedback and contribution in developing our implementation.

REFERENCES

[1] Amirali Boroumand, Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun, and etc. 2018. Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '18)*. 316–331.

[2] Febrice Devaux. 2019. True Processing in Memory with DRAM accelerator. *Hot Chips 31* (2019).

[3] Boncheol Gu, Andre S. Yoon, Duck-Ho Bae, Insoon Jo, Jinyoung Lee, and etc. 2016. Biscuit: A Framework for Near-data Processing of Big Data Workloads. In *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA '16)*. 153–165.

[4] Insoon Jo, Duck-Ho Bae, Andre S. Yoon, Jeong-Uk Kang, Sangyeun Cho, and etc. 2016. YourSQL: A High-performance Database System Leveraging In-storage Computing. *Proc. VLDB Endow.* 9, 12 (Aug. 2016), 924–935.

[5] Tomas Karnagel, Dirk Habich, and Wolfgang Lehner. 2017. Adaptive Work Placement for Query Processing on Heterogeneous Computing Resources. *Proc. VLDB Endow.* 10, 7 (March 2017), 733–744.

[6] Tomas Karnagel, Renè Müller, and Guy M. Lohman. 2015. Optimizing GPU-accelerated Group-By and Aggregation.. In *ADMS@VLDB*. 13–24.

[7] Nusrat Jahan Lisa, Annett Ungethüm, Dirk Habich, Wolfgang Lehner, Tuan D. A. Nguyen, and Akash Kumar. 2018. Column Scan Acceleration in Hybrid CPU-FPGA Systems. In *International Workshop on Accelerating Analytics and Data Management Systems Using Modern Processor and Storage Architectures, ADMS@VLDB 2018, Rio de Janeiro, Brazil, August 27, 2018*. 22–33.

[8] J. McGlone, P. Palazzari, and J. B. Leclere. 2018. Accelerating Key In-memory Database Functionality with FPGA Technology. In *2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. 1–8.

[9] Hasso Plattner. 2014. The Impact of Columnar In-memory Databases on Enterprise Systems: Implications of Eliminating Transaction-maintained Aggregates. *Proc. VLDB Endow.* 7, 13 (Aug. 2014), 1722–1729.

[10] David Sidler, Zsolt István, Muhsen Owaida, and Gustavo Alonso. 2017. Accelerating Pattern Matching Queries in Hybrid CPU-FPGA Architectures. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17)*. 403–415.

[11] Thomas Willhalm, Ismail Oukid, Ingo Müller, and Franz Färber. 2013. Vectorizing Database Column Scans with Complex Predicates.

[12] Thomas Willhalm, Nicolae Popovici, Yazan Boshmaf, Hasso Plattner, and etc. 2009. SIMD-scan: Ultra Fast In-memory Table Scan Using On-chip Vector Processing Units. *Proc. VLDB Endow.* 2, 1 (Aug. 2009), 385–394.