First Investigations of the Vector Supercomputer SX-Aurora TSUBASA as a Co-Processor for Database Systems

Johannes Pietrzyk[1] Dirk Habich[1] Patrick Damme[1] Wolfgang Lehner[1]

Abstract: The hardware landscape is currently changing from homogeneous multi-core systems towards heterogeneous systems with many different computing units, each with their own characteristics. This trend is a great opportunity for database systems to increase the overall performance if the heterogeneous resources can be utilized efficiently. Following that trend, NEC cooperation has recently introduced a novel heterogeneous hardware system called SX-Aurora TSUBASA. This novel heterogeneous system features a strong vector engine as a (co-)processor providing world's highest memory bandwidth of 1.2TB/s per vector processor. From a database system perspective, where many operations are memory bound, this bandwidth is very interesting. Thus, we describe the unique architecture and properties of this novel heterogeneous system in this paper. Moreover, we present first database-specific evaluation results to show the benefit of this system to increase the query performance. We conclude the paper with an outlook on our ongoing research activities in this direction.

Keywords: Database Systems; Heterogeneous Hardware; Vector Processor

1 Introduction

In our digital world, efficient query processing is still an important aspect due to the ever-growing amount of data. To satisfy query response times and query throughput demands, the architecture of database systems is constantly evolving [BKM08, HZH14, KHL17, Li16, Ou17]. For instance, the database architecture shifted from a disk-oriented to a main memory-oriented architecture to efficiently exploit the ever-increasing capacities of main memory [Ab13, Id12, Ki13, St05]. This in-memory database architecture is now state-of-the-art and characterized by the fact, that all relevant data is completely stored and processed in main memory. Additionally, relational tables are organized by column rather than by row [Ab13, BKM08, CK85, Id12, St05] and the traditional tuple-at-a-time query processing model was replaced by newer and adapted processing models like column-at-a-time or vector-at-a-time [Ab13, BKM08, Id12, St05, ZvdWB12].

To further increase the performance of queries, in particular for analytical queries in these in-memory column stores, two key aspects play an important role. On the one hand, data

[1] Technische Universität Dresden, Institut for Systems Architecture, Dresden Database Systems Group, Nöthnitzer Straße 46, 01187 Dresden, firstname.lastname@tu-dresden.de

doi:10.18420/btw2019-ws-03

compression is used to tackle the continuously increasing gap between computing power of CPUs and memory bandwidth (also known as memory wall [BKM08]) [AMF06, BHF09, Da17, Hi16, Zu06]. Aside from reducing the amount of data, compressed data offers several advantages such as less time spent on load and store instructions, a better utilization of the cache hierarchy, and less misses in the translation lookaside buffer. On the other hand, in-memory column stores constantly adapt to novel hardware features like vectorization using Single-Instruction Multiple Data (SIMD) extensions [PRR15, ZvdWB12], GPUs [HZH14, KML15] or non-volatile main memory [Ou17].

From a hardware perspective, we currently observe a shift from homogeneous CPU systems towards hybrid systems with different computing units mainly to overcome physical limits of homogeneous systems [Es12, LUH18]. Following that hardware trend, NEC cooperation recently released a novel heterogeneous hardware system called SX-Aurora TSUBASA [Ko18]. The main advantage of this novel hardware system is its strong vector engine which provides world's highest memory bandwidth of up to 1.2TB/s per vector processor [Ko18]. From that point, this novel hardware could be very interesting for database systems. In particular, from the following aspects:

1. Vectorization is a hot-topic in database systems to improve query processing performance by parallelizing computations over vector registers [PRR15, ZvdWB12]. Intel's latest vector extension is AVX-512 with vector registers of size 512 bits. In contrast to that, the vector engine of SX-Aurora TSUBASA features a vector length of 216KB (16.384 bits).
2. SX-Aurora TSUBASA offers enough memory bandwidth to fill these large vectors with data for efficient processing.

Therefore, we describe the unique architecture and properties of this novel heterogeneous system in this paper (Section 2). Moreover, we present first database-specific evaluation results to show the benefit of this system to increase the query performance in Section 3. Based on that, we briefly introduce our ongoing research activities in this direction in Section 4. We conclude the paper in Section 5 with a short summary.

2 Vector Supercomputer SX-Aurora TSUBASA

NEC Cooperation has a long tradition in vector supercomputers with a series of NEC SX models starting 1983. The current model is NEC SX-Aurora TSUBASA. In the following sections, we will describe the overall architecture, the vector processing and the programming approach of this novel SX-Aurora TSUBASA model.

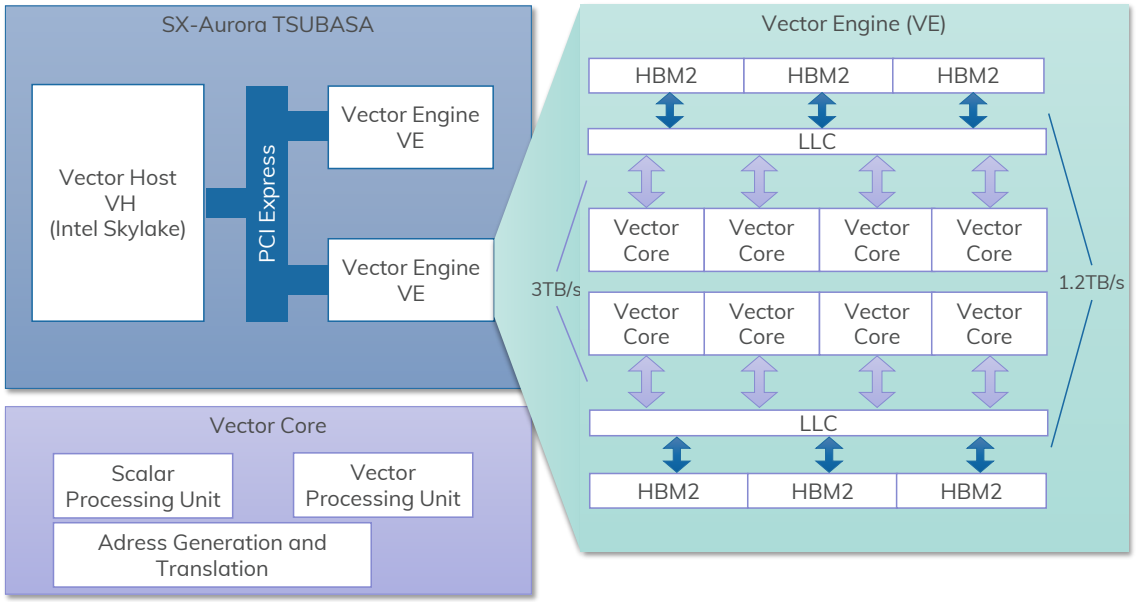

Fig. 1: SX-Aurora TSUBASA Architecture.

2.1 Overall Architecture

The overall architecture of SX-Aurora TSUBASA completely differs from its predecessors in the SX series. The new system model is a heterogeneous system consisting of a vector host (VH) and one or more vector engines (VE). As illustrated in Figure 1, the VH is a regular Intel Xeon Skylake CPU featuring a standard x86 Linux server that provides standard operating systems functions. Moreover, the VH also includes a special operating system for the VE called *VEOS* which runs in the user mode of the VH. *VEOS* controls the VE, whereby each VE is implemented as a PCI Express (PCIe) card equipped with a newly developed vector processor [Ko18].

As illustrated in Figure 1 on the right side, each vector processor consists of 8 vector cores, 6 banks of HBM2[2] high speed memory, and only one shared last-level cache (LLC) of size 16MB between memory and the vector cores. The LLC is one both sides of the vector cores, and it is connected to each vector core through a 2D mesh network-on-chip with a total cache bandwidth of 3TB/s [Ko18]. Moreover, this vector processor design provides the world highest memory bandwidth of up to 1.2TB/s per vector processor [Ko18]. Each vector core consists of three core units: (i) a scalar processing unit (SPU), a vector processing unit (VPU), and (iii) a memory addressing vector control and processor network unit (AVP). The SPU has almost the same functionality as modern processors such as fetch, decode, branch, add, and exception handling. However, the main task of the SPU is to control the status of the vector core.

[2] High Bandwidth Memory Version 2

Type	Frequency	DP Performance of a core	DP Performance of a Processor	Memory Bandwidth	Memory Capacity
VE 10A	1.6 GHz	307.2Gflop/s	2457.6Gflop/s	1228.8GB/s	48 GB
VE 10B	1.4 GHz	268.8.2Gflop/s	2150.4Gflop/s	1228.8GB/s	48 GB
VE 10C	1.4 GHz	268.8.2Gflop/s	2150.4Gflop/s	750.0GB/s	24 GB
VH Intel Xeon Gold 6126	2.5GHz	83.2Gflops/s	998.4Gflops/s	128GB/s	96GB

Tab. 1: Specifications for SX-Aurora TSUBASA.

2.2 Vector Processing and Specific Systems

Besides the high bandwidth, the architecture of the VPU of the vector core is a further advantage of this processor. The VPU has three vector fused multiply add units, which can be independently executed by different vector instructions, whereby each unit has 32 vector pipelines consisting of 8 stages [Ko18]. Generally, the vector length of the VPU is 256 elements[3], each of which is 8 Byte [Ko18]. One vector instruction executes 256 arithmetic operations within eight cycles [Ko18]. The major advantage, compared to wider SIMD functionalities e.g., in Intel processors like AVX-512, is that the operations are not only executed spatially parallel, but also temporally parallel which better hides memory latency [Ko18]. Each VPU has 64 vector registers and each vector register is 2Kb in size (32 pipeline elements with 8 Byte per element). Thus, the total size of the vector registers is 128Kb per vector core, which is larger than a L1 cache in modern regular processors. To fill these large vector registers with data, the LLC is directly connected to the vector registers and the connection has roughly 400GB/s bandwidth per vector core [Ko18].

Generally, NEC offers three types of these VE called 10A, 10B, and 10C as shown in Table 1, which only differs in frequency, memory bandwidth and memory capacity. In every case, the VH is an Intel Xeon Gold 6126 with 12 cores. Table 1 also compares VE and VH with respect to double precision (DP) performance per core and per processor as well as memory bandwidth and memory capacity. As we can see, memory bandwidth of each VE is many times higher than that of the VH, but maximum memory capacity of the VE is 48GB.

The SX-Aurora TSUBASA approach has a high level configuration flexibility and the series includes three product types:

- A100 is a workstation model with one VH and one VE.
- A300 is standard rack-mount model with up to eight VE with one VH. In this case, the maximum size of the vector main memory is 384GB.

[3] In comparison, the vector length of Intel's latest vector extension AVX-512 is limited to 8 elements with 8 Byte per element.

- A500 is designed as large-scale supercomputer with up to eight A300 models connected which results in maximum vector main memory capacity of 3.072GB.

With these memory capacities and bandwidths, this heterogeneous system approach is very interesting for memory-intensive applications such as database management systems.

2.3 Execution Model and Programming Approach

Unlike other accelerators, SX-Aurora TSUBASA is pursuing a different execution model. In general, the VE is entirely responsible for executing applications, while the VH provides basic OS functionalities such as process scheduling and handling of system calls invoked by the applications on the VE [Ko18]. Applications for the VE are written in standard programming languages such as C, C++ or Fortran without having to use special programming models. For this, a C library compliant with standards is ported to VE [Ko18]. Therefore, existing (non-vectorized) applications can be ported to VE just by recompiling using the NEC compiler.

In summary, the high bandwidth of the SX-Aurora TSUBASA VE and big vector registers are a promising combination for improving query processing performance in database systems.

3 Database-oriented Evaluation

Since many in-memory database operations are memory bound, we basically have focused our evaluation on examining the specified memory throughput of the introduced hardware. Therefore, we measured plain sequential memory access in a first step, followed by a column-scan which can be considered as a fundamental physical query operator.

3.1 Investigated Operators

Fundamentally, memory access can be distinguished between reading from memory, writing to memory and copying data. As a first step in our evaluation, we focused on these core primitives. While reading from memory without any further operations can be considered to be optimized out by the compiler, an aggregation is performed over the read memory using the bitwise *OR* operator. Thus, only one cache-resident value has to be updated per actual read. Given a relatively fast aggregation operation with regard to the memory access, it can be assumed that the measured throughput is not distorted by computation effort.

To measure the behaviour of write intense sequential memory access, we filled a given array with a constant value, like a *memset*. As a combination of both classes of memory access

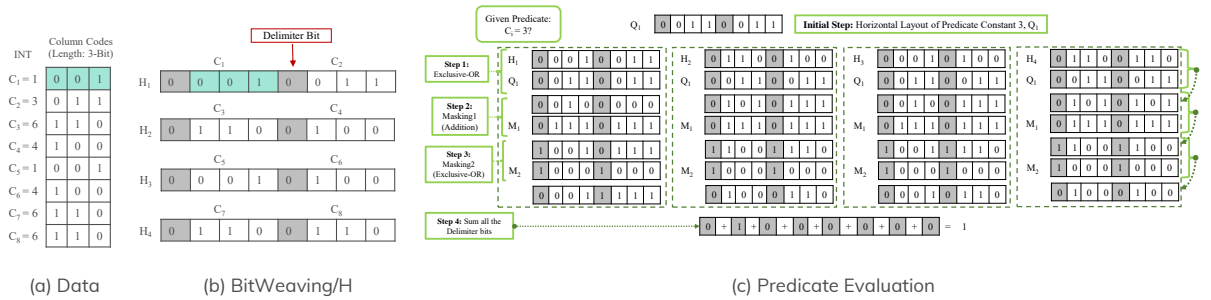

Fig. 2: Illustration of BitWeaving/H (taken from [Li18]).

patterns, we measured the throughput of copying the values from a given array into another array.

As a second step and as a more database relevant I/O bound operation, we evaluated a column scan operation using a state-of-the-art approach called BitWeaving/H [LP13]. Fundamentally, BitWeaving assumes a fixed-length order preserving compression scheme, so that all compressed column codes of a column have the same bit length [LP13]. Then, the bits of the column codes are aligned in main memory in a way that enables the exploitation of intra-cycle parallelism using ordinary processor words. An example is shown in Figure 2(a), where eight 32-bit integer values C_i are represented using 3-bit compressed column codes. As illustrated in Figure 2(b), the column codes are contiguously stored in processor word H_i in BitWeaving/H, where the most significant bit of every code is used as a delimiter bit between adjacent column codes. In our example, we use 8-bit processor words, so that two 3-bit column codes fit into one processor word including one delimiter bit per code. The delimiter bits are used later to store the result of a predicate evaluation.

Now, the task of a column scan is to compare each column code with a constant C and to output a bit vector indicating whether or not the corresponding code satisfies the comparison condition. To efficiently perform such a column scan using the BitWeaving/H, Li et al. [LP13] proposed an arithmetic framework to directly execute predicate evaluations on the compressed column codes. There are two main advantages: (i) predicate evaluation is done without decompression and (ii) multiple column codes are simultaneously processed within a single processor word using full-word instructions (intra-cycle parallelism) [LP13]. The supported predicate evaluations include equality, inequality, and range checks, whereby for each evaluation a function consisting of arithmetical and logical operations is defined [LP13].

Figure 2(c) highlights the *equality* check in an exemplary way, whereby the other predicate evaluations work in a similar way. The input from Figure 2(b) is tested against the condition $C_i = 3$. Then, the predicate evaluation steps are as follows:

Initially: All given column codes and the query constant number *3* are converted into the BitWeaving/H storage layout (H_1, H_2, H_3, H_4) and Q_1, respectively.

Type	Max. vector size	Cache accessible from vector reg.	OS	Compiler
VE 10 B/C	16384 Bit ($256 \cdot 64$ Bit)	16 MB LLC	VEOS 1.3.2	nc++ 1.6.0
VH Intel Xeon Gold 6126	512 Bit ($8 \cdot 64$ Bit)	32 KB L1 1 MB L2 19.25 MB L3	CentOs 7.5	g++ 7.3.1

Tab. 2: Specification for hardware, operating system and compiler used for experiments.

Step 1: An *Exclusive-OR* operation between the words (H_1, H_2, H_3, H_4) and Q_1 is performed.

Step 2: *Masking1* operation (*Addition*) between the intermediate results of Step 1 and the M_1 mask register (where each bit of M_1 is set to one, except the delimiter bits) is performed.

Step 3: *Masking2* operation (Exclusive-OR) between the intermediate results of Step 2 and the M_2 mask register (where only delimiter bits of M_2 are set to one and the rest of all bits is set to zero) is performed.

Step 4 (optional): Add delimiter bits to achieve the total count (final result).

The output is a result bit vector, with one bit per input code that indicates if the code matches the predicate on the column. In our example in Figure 2, only the second column code (C_2) satisfies the predicate which is visible in the resulting bit vector.

3.2 Experimental Setup

All operations were measured on two different versions of the SX-Aurora TSUBASA co-processor. General specifications are denoted in Table 2. To compile the implemented operators for the VH system, a gcc 7.3.1 was used with the optimization flags *-O3 -flto* and disabled auto-vectorization (*-fno-tree-vectorize*). For the VE, the proprietary NEC compiler nc++ 1.6.0 was used with the optimization flag *-O3 -fipa* and *-mvector*, enabling the auto-vectorization. A distinction between single-thread performance and multi-thread performance were made by linking the binary with and without OpenMP. To minimize the runtime overhead through dynamic linking, all files were linked statically.

3.3 Experimental Methodology

The time measurements were performed using a c++ wall-time clock on the VH and inline assembly for retrieving user clock cycles on the VE. While an experiment consists of a measurement of all specified task, every experiment was repeated 10 times and and the reported runtimes are averaged. To avoid distortion by the actual time measurement, all

(a) C++ Code for Aggregation

```
1  /*...*/
2  #pragma omp parallel
3  {
4  #pragma _NEC vreg(aRes)
5  #pragma omp for
6  #pragma _NEC noouterloop_unroll
7   for(;i<nBuf;i+=256){
8  #pragma _NEC shortloop
9    for(j=0;j<256;++j){
10     aRes[j]|=aSrc[i+j];
11   }
12  }
13  for(k=0;k<256;++k){
14   nRes|=aRes[k];
15  }
16 }
17 /*...*/
```

(b) nc++ Listing after compilation

```
1  LINE DIAGNOSTIC MESSAGE
2    1: Vector reg. assigned.: aRes
3    2: Parallel routine generated.
4    7: Parallelized by "for".
5    9: Vectorized loop.
6   13: Vectorized loop.
7   14: Idiom detected.: Bit-op
8  /*...*/
9  LINE LOOP STATEMENT
10 /*...*/
11   7- P---> for(;i<nBuf;i+=256){
12   8- |      #pragma _NEC shortloop
13   9- |V-->   for(j=0;j<256;++j){
14  10- ||  V    aRes[j]|=aSrc[i+j];
15  11- |V--    }
16  12- P---   }
17  13- V---> for(k=0;k<256;++k){
18  14- |   V  nRes|=aRes[k];
19  15- V---> }
20 /*...*/
```

Fig. 3: Excerpt of C++-Code for read intense task and corresponding diagnostic listing, produced by nc++ while compilation with optimization indications.

tasks were repeated multiple times and the accumulated time was divided by the amount of repetitions.

Thus the focus was on evaluating the computing performance of vectorized code alongside the memory bandwidth, all tasks were implemented using vectorization. This was done using intrinsics for the VH. To examine the best performance of the different SIMD extensions offered by the VH, all tasks were implemented and tested using either SSE, AVX2 or AVX512.

As shown in Figure 3(a), a combination of compiler specific preprocessor pragma directives, strip mining and local buffers were used for the VE to facilitate the auto-vectorization feature of the NEC compiler. At first, the main loop which iterates over the buffer as a whole is fragmented into strips according to the size of a vector register (see Line 7). To prevent the compiler from undo the so called loop strip-mining, an according pragma was used (see Line 6). The most inner loop, containing the operator specific instructions, is marked as a shortloop (see Line 8) giving the compiler a hint that the loop should completely be transformed into vector code. In addition, the specific instructions work on temporal buffers which are forcedly assigned to a vector register using *#pragma _NEC vreg(arrayName)* (see Line 4). First measurements showed that this specific hint can significantly improve the performance of the algorithm. OpenMP were introduced through parallel regions using *#pragma omp parallel* (see Line 2) alongside loop parallelism using *#pragma omp for*

VE VPU	VH	Buffer sizes				
LLC	L1	16 KB	32 KB			
	L2	512 KB	1 MB			
	L3	4 MB	8 MB	16 MB		
HBM2	RAM	32 MB	128 MB	1 GB	4 GB	8 GB

Tab. 3: Measured buffer sizes.

depicted at Line 5. When compiling the annotated C++-Code using the NEC compiler, a diagnostic listing, indicating all applied optimization's, can be generated (see Figure 3(b)).

Within the VE, two different element sizes (32, 64 Bit) were measured. Within the VH, different load and store modes (stream, aligned, unaligned) were measured. To evaluate a possible influence of different memory structures on the performance, every experiment were executed while processing different sizes of data (see Table 3).

3.4 Evaluation

The measured time needed for executing the described tasks were used to calculate the average throughput per task. Except for the copying-task the read and write intense tasks either read one buffer or store one buffer. The column scan reads one buffer. Thus the processed buffer size was used for calculating the throughput. When it comes to copying, actually every element from a source buffer is read and stored into the destination buffer. Taken this into consideration the number of processed elements would be the sum of the source and destination buffer sizes. However, the throughput of a copy-operation is typically deduced directly from the size of the source buffer. Thus only this quantity was used to calculate the throughput.

3.4.1 Single-Thread Evaluation

Figure 4 shows the throughput of plain memory access measured on the VH (a)-(c) as well as on the VE (d)-(f). Using the VH, a maximum throughput of around 100 GB/s were obtained when the processed data fits completely into L1. In general the performance gets lower if the buffer sizes exceed the cache sizes. While read intense tasks can utilize L2 without significant performance penalties, write intense tasks suffer from accessing higher levels of cache. Conversely, the throughput measured on the VE gets better with bigger buffer size obtaining an overall maximum throughput of around 300 GB/s (processing 1MB) for write intense tasks (see Figure 4(e)) and 250 GB/s (processing 2MB) while executing read intense tasks (see Figure 4(f)). Accessing the HBM2 leads to a marginal decrease in measured throughput.

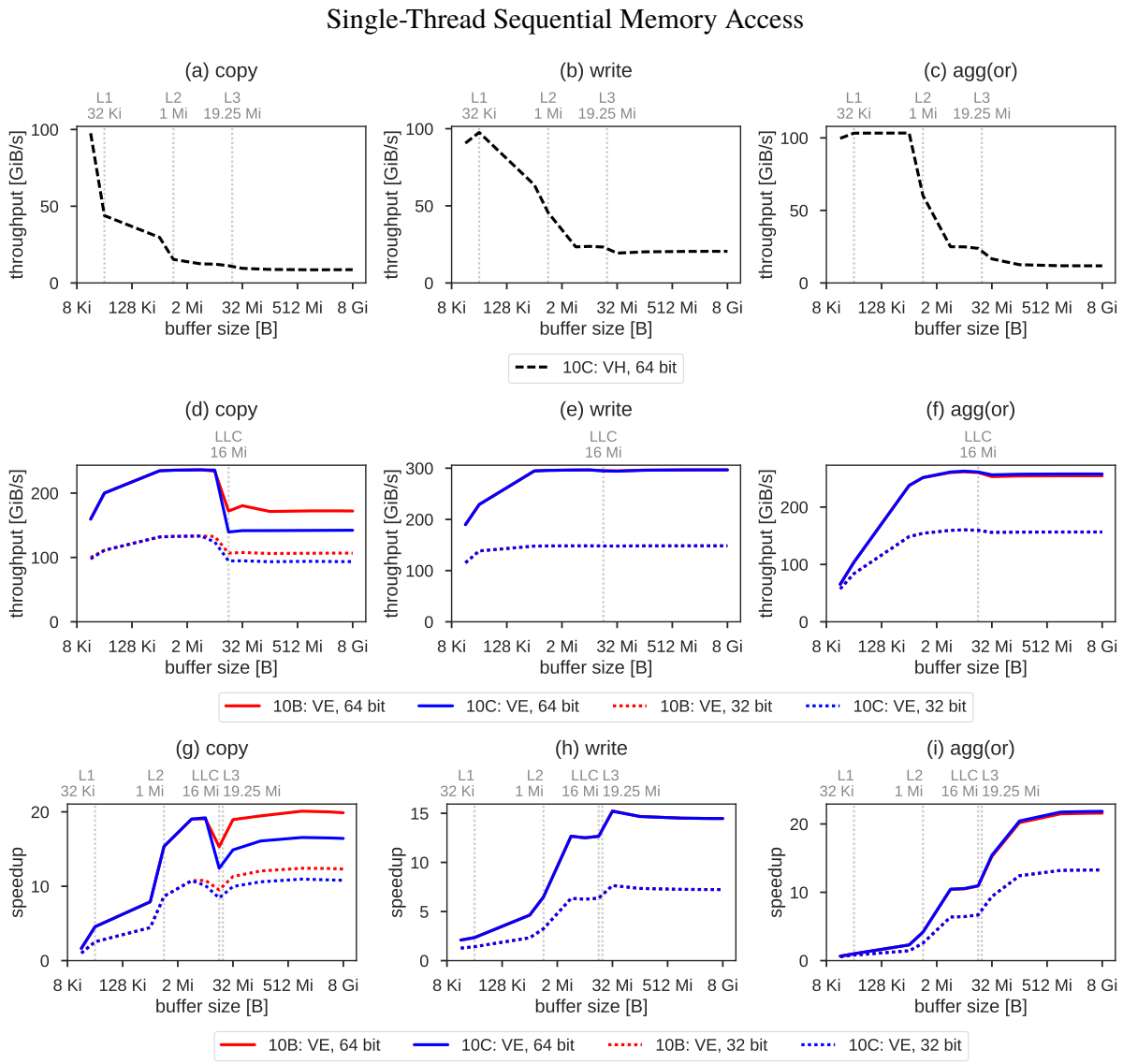

Fig. 4: Measured throughput of VH and engine as well as the speedup obtained by the VE of different IO-operations executed using one thread.

As shown in Figure 4(d), only the performance of a vectorized copy drops dramatically when the processed data sizes exceeds the boundaries of the existing LLC. Since the tasks were executed vectorized and a single vector register can hold up to 2 KB data, small buffers prevent the VE of using the given vector registers in an efficient manner. In general the experiments have shown that processing 64-bit wide elements led to significantly higher throughputs than working on 32-bit sized data. This results from the under-utilization of available vector registers. The vector pipeline processes its elements at a granularity of 64 bit. If only 32 bit wide elements were processed, the remaining bits left unused. An improvement in terms of the memory access could not be achieved on the formally faster TSUBASA 10B neither for vectorized write-intense nor read-intense tasks. Only the performance of the copy task could benefit from the better memory bandwidth of the 10B.

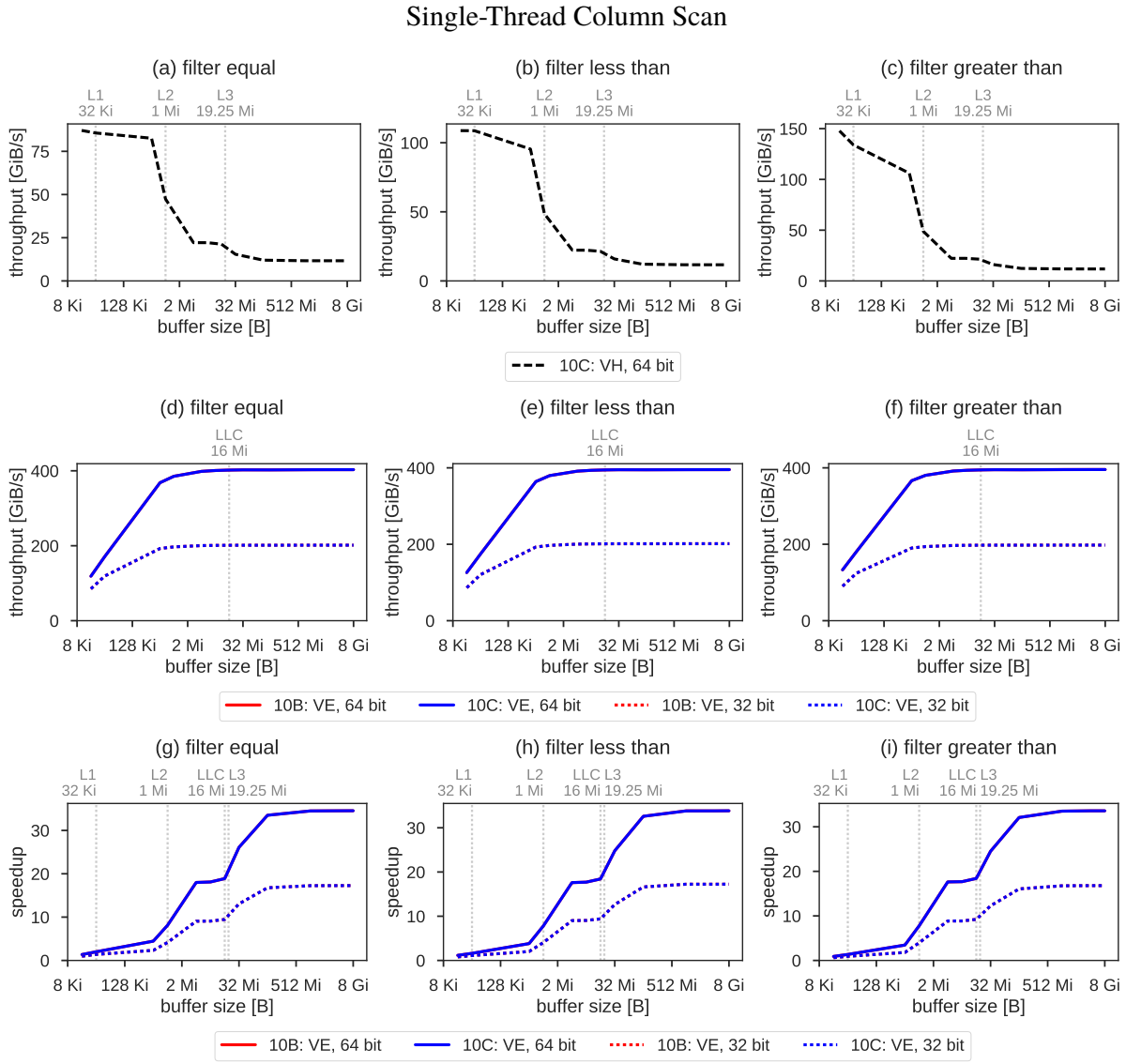

Fig. 5: Measured throughput of VH and engine as well as the speedup obtained by the VE of different Bitweaving-H operations executed using one thread.

As shown in Figure 4(g)-(i) both VE outperform the VH for write intense tasks up to a factor of 15 on the 10C and 20 on the 10B respectively. A maximum speedup of around 21 were obtained for read intense tasks when the processed data exceeds the cache and has to be loaded from DRAM (vh) or HBM2 (ve) respectively.

As mentioned in Section 3.1, the actual scan is executed using arithmetic operations. While a filter for equality needs two bitwise operators (*XOR*, *NOT*) and an addition, a filter for less than needs only one bitwise operator (*XOR*) and an addition. To scan for elements which are greater than the predicate only one addition is executed. These characteristics can be seen in Figure 5(a)-(c) where the discussed column scan operators achieve a maximum throughput in the range of 80 GB/s (Figure 5(a)) up to 150 GB/s (Figure 5(c)) when the processed data

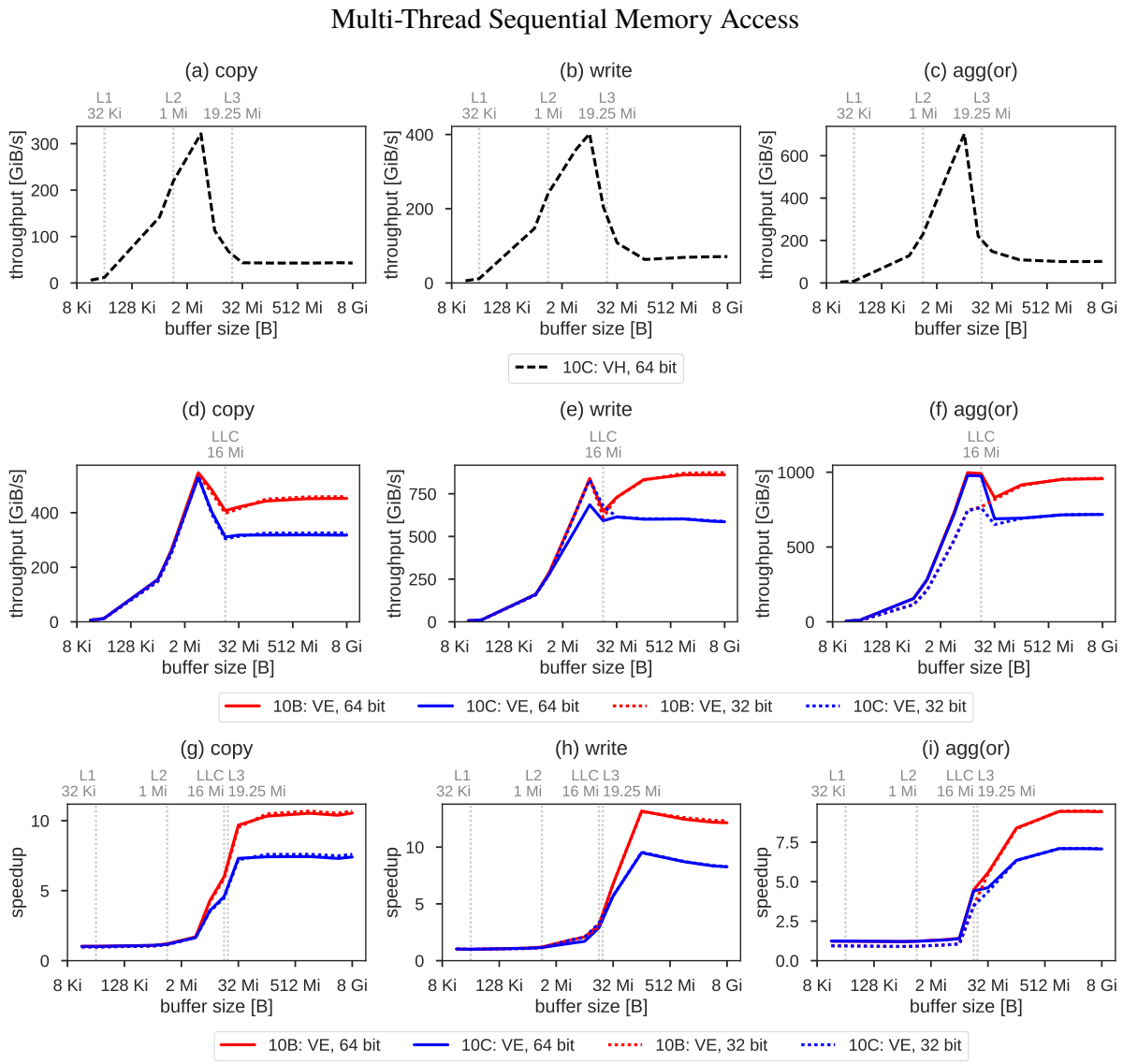

Fig. 6: Measured throughput of different IO-operations executed on the VH using multiple threads.

fits entirely into L1. Running onto the VE the total amount of executed operations does not affect the overall throughput leading to mostly similar behaviour like the write task. A maximum throughput of around 400 GB/s were reached when the processed buffer exceeds the boundaries of the LLC. As shown in Figure 5(g)-(i) both VE outperform the VH up to a factor of 20 when the processed buffer is resident in the caches. When the buffer exceeds the LLC a speedup of factor 33 could be achieved by the VE.

3.4.2 Multi-Thread Evaluation

Using multiple threads introduce additional complexity in terms of thread creation as well as data partitioning. As shown in Figure 6 this overhead leads to significant lower throughputs

for both the VH and the VE when processing small buffers. Nevertheless multithreading pays off on the VH when the processed data exceeds the L2 cache (see Figure 6(a)-(c)) obtaining a maximum throughput of around 320 GB/s for copying, 400 GB/s for writing and 700 GB/s for aggregating respectively. Taken this into account, using multiple threads for plain memory access can speedup the processing up to a factor of 7 compared with single-thread execution.

The same observation holds for the VE in terms of processing small buffers up to 2 MB. While plain memory access using a single thread reaches a maximum throughput when processing 1MB and upwards, multiple threads reach a local maximum with around 4 MB buffer size for copying and 8 MB buffer size for reading and writing respectively. For bigger buffers which still fit into the LLC, the throughput decreases probably caused by effects like cache pollution. As shown in Figure 6(d)-(f) the measured throughput of the 10C remain stable while the throughput at around 300 GB/s for copying and 600 GB/s to 700 GB/s for write and read intense tasks respectively when the processed buffer size exceeds the boundaries of the LLC. The TSUBASA 10B even outperforms the TSUBASA 10C when processing HMB resident buffers through the higher maximum bandwidth resulting in an overall maximum throughput of around 800 GB/s for writing and nearly 1 TB/s for aggregation. Using multiple threads the VE TSUBASA 10B outperforms the VH up to a factor of 10 for copying and aggregating, 13 for writing respectively.

Executing bitparallel column-scans using multiple threads show similar behaviour like the reading task by exceeding the reached throughput of single thread execution by a factor of around 2. Interestingly the bitwidth has an significant influence when running on the VE. As shown in Figure 7(d)-(f) processing 64-bit wide elements led to higher throughputs in general. This impact of processed word size decreases for processing big buffers using less operations.

3.5 Summary

The conducted experiments show that the vector co-processor SX-Aurora TSUBASA can on the one hand improve the performance of computational-bound algorithms through the utilization of wide vector registers alongside an efficient vector processing pipeline. On the other hand memory-bound algorithms can benefit from the integrated high bandwidth memory in combination with the shared LLC which is accessible from the VPU directly.

Furthermore, the existing NEC compiler with its integrated automatic vectorization feature facilitates the reuse of existing scalar C/C++-Code with only minor adjustments.

4 Future Work

As we have clearly shown in the previous section, the VE of SX-Aurora TSUBASA offers outstanding performance characteristics from a database query processing perspective. To

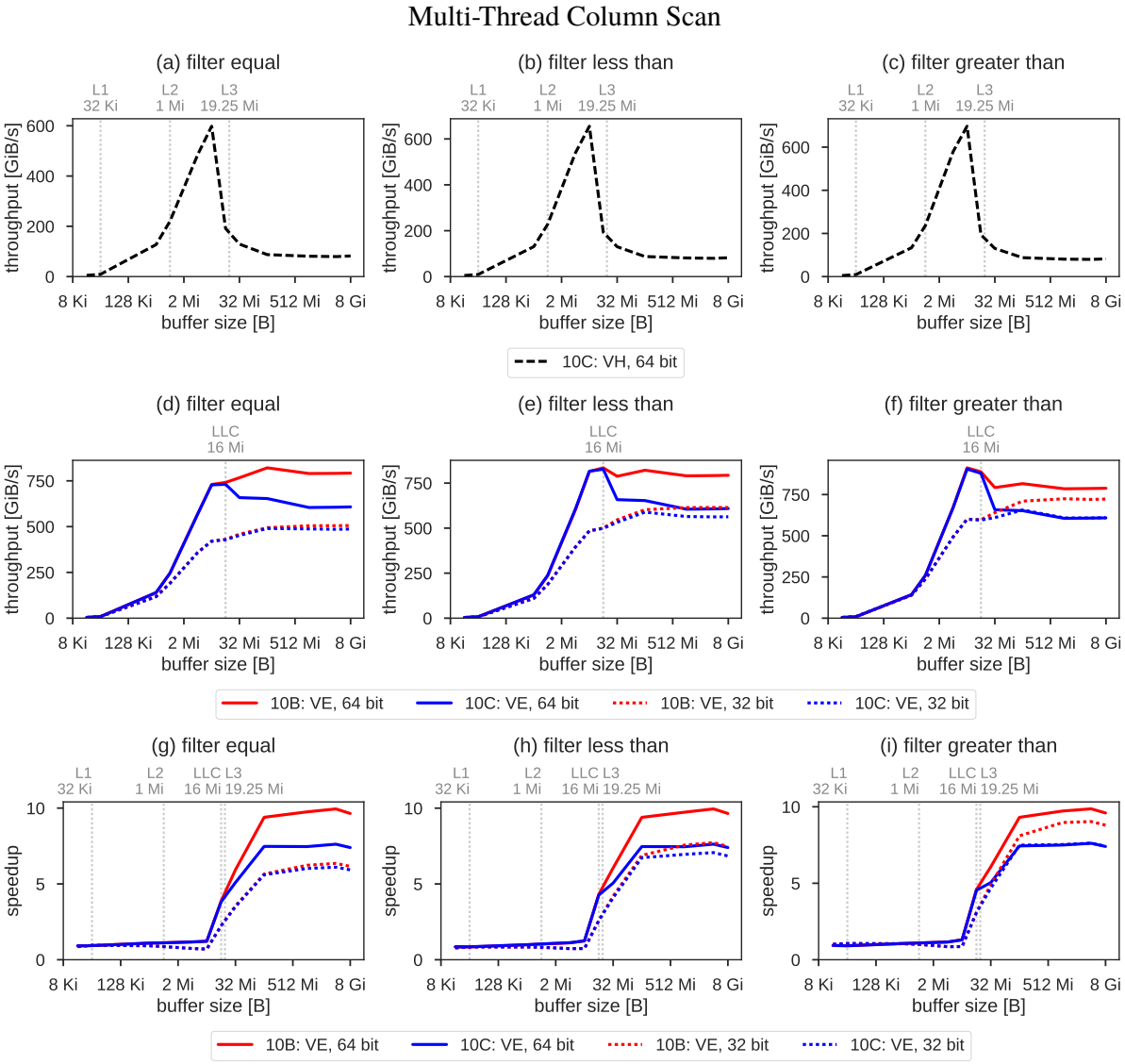

Fig. 7: Measured throughput of different Bitweaving-H operations executed on the VH using multiple threads.

further investigate this hardware system for database systems in detail, our ongoing research goes into two directions as described next.

4.1 Highly Vectorized Query Processing on Compressed Columnar Data

In-memory column store systems are state-of-the-art for the efficient processing of analytical workloads. In such systems, data compression as well as vectorization are extremely relevant aspects. On the one hand, data compression is applied to tackle the continuously increasing gap between computing power of common CPUs and main memory bandwidth (also known as memory wall). Therefore, all in-memory column store systems roughly pursue a similar

approach: (i) each column of a relational table is treated independently, (ii) values of each column are encoded as a sequence of integers usually applying dictionary coding and (iii) different data compression schemes are applied to each sequence of integers resulting in a sequence of compressed data strings. For such lossless integer compression schemes, a large variety of lightweight algorithms has been developed. On the other hand, vectorization is used to improve the processing performance by parallelizing computations over vector registers. This vectorization is usually performed using SIMD (Single Instruction Multiple Data) extensions such as Intel's SSE (Streaming SIMD Extensions) or AVX (Advanced Vector Extensions) and has been available in modern processors for several years. Up to now, Intel's latest SIMD extensions supports 512-bit vector registers. From a database perspective, this vectorization technique is not only very interesting for compression and decompression, but also for all a set of database operator like joins, scans, as well as groupings.

In contrast to Intel's SIMD extensions, the SX-Aurora TSUBASA supports vector registers with size of 16,384-bit (256x64-bit) which significantly outperforms techniques offered by other vendors today. Thus, we want to examine the effects of these large vector registers on the query processing in the context of state-of-the-art in-memory column store systems. Therefore, we plan to investigate two aspects: (i) lightweight data compression as well as decompression and (ii) high-performance and highly vectorized execution of compressed data using scans (point and range queries), joins (sort-merge as well as hash joins), and groupings (sort as well as hash-based).

4.2 Near-Memory Data Processing on Multiple Vector Engines

Modern applications demand data management systems to provide strong analytical and transactional power at the same time while facing a continuously increasing amount of data that needs to be maintained and processed. To challenge those issues, modern data management systems need scale up on today's massively parallel hardware and need to be capable to adapt to the current workload by changing data layout, data partitioning, and hardware resource configurations on-the-fly without negatively affecting the running queries. Since traditional disk-based or in-memory database systems were never designed to scale to the degree of parallelism that is offered by modern hardware, scale-out solutions such as Apache Spark came up to address the issue of analytical power by assuming a read-only data access similar to batch processing setups on HPC systems. With the ERIS data management system [Ki14, KHL18], we already invented a novel near-memory computing-optimized database system architecture that aims at scalability, adaptivity, and a superior absolute performance compared to systems like Spark focusing on scale-up hardware systems.

Currently, our ERIS data management system is optimized for traditional x86 based server hardware. The main objective of this research direction is to port the current C++ implementation to entirely run on multiple VE. In particular, we will address the following research questions:

- Is the SX-Aurora TSUBASA architecture suitable for running sophisticated data management applications and how does this architecture perform compared to a traditional x86-only system. To do this comparison, we plan to employ a rich set of standard transactional and analytical benchmarks for relational and graph workloads.
- Is the database system able to take advantage of the high memory bandwidth and the huge vector register sizes of the VE and which modifications to ERIS are necessary to better utilize the key features offered by this hardware architecture?
- Do the integrated adaptivity features of ERIS offer any advantage on the SX-Aurora TSUBASA architecture in case of a changing workload, for instance, by adapting the physical data layout in case of a changing data access pattern? Are there different kinds of adaptivity features necessary to fully unleash the performance of the hardware architecture?

5 Conclusion

The hardware landscape is currently changing from homogeneous multi-core systems towards heterogeneous systems with many different computing units, each with their own characteristics. This trend is a great opportunity for database systems to increase the overall performance if the heterogeneous resources can be utilized efficiently. Following that trend, NEC cooperation has recently introduced a novel heterogeneous hardware system called SX-Aurora TSUBASA. The core and unique feature of this novel hardware system is its strong vector engine processor providing world's highest memory bandwidth of 1.2TB/s per vector processor. Thus, we introduced this architecture and properties in this paper. Moreover, we presented first insight of database-specific evaluation results to show the benefits of this hardware system to increase the query performance. We concluded the paper with an outlook on our ongoing research activities in this direction and a short summary.

References

[Ab13] Abadi, Daniel; Boncz, Peter A.; Harizopoulos, Stavros; Idreos, Stratos; Madden, Samuel: The Design and Implementation of Modern Column-Oriented Database Systems. Foundations and Trends in Databases, 5(3):197–280, 2013.

[AMF06] Abadi, Daniel J.; Madden, Samuel; Ferreira, Miguel: Integrating compression and execution in column-oriented database systems. In: SIGMOD. pp. 671–682, 2006.

[BHF09] Binnig, Carsten; Hildenbrand, Stefan; Färber, Franz: Dictionary-based order-preserving string compression for main memory column stores. In: SIGMOD. pp. 283–296, 2009.

[BKM08] Boncz, Peter A.; Kersten, Martin L.; Manegold, Stefan: Breaking the memory wall in MonetDB. Commun. ACM, 51(12):77–85, 2008.

[CK85] Copeland, George P.; Khoshafian, Setrag: A Decomposition Storage Model. In: SIGMOD. pp. 268–279, 1985.

[Da17] Damme, Patrick; Habich, Dirk; Hildebrandt, Juliana; Lehner, Wolfgang: Lightweight Data Compression Algorithms: An Experimental Survey (Experiments and Analyses). In: EDBT. pp. 72–83, 2017.

[Es12] Esmaeilzadeh, Hadi; Blem, Emily R.; Amant, Renée St.; Sankaralingam, Karthikeyan; Burger, Doug: Dark Silicon and the End of Multicore Scaling. IEEE Micro, 32(3):122–134, 2012.

[Hi16] Hildebrandt, Juliana; Habich, Dirk; Damme, Patrick; Lehner, Wolfgang: Compression-Aware In-Memory Query Processing: Vision, System Design and Beyond. In: ADMS. pp. 40–56, 2016.

[HZH14] He, Jiong; Zhang, Shuhao; He, Bingsheng: In-Cache Query Co-Processing on Coupled CPU-GPU Architectures. PVLDB, 8(4):329–340, 2014.

[Id12] Idreos, Stratos; Groffen, Fabian; Nes, Niels; Manegold, Stefan; Mullender, K. Sjoerd; Kersten, Martin L.: MonetDB: Two Decades of Research in Column-oriented Database Architectures. IEEE Data Eng. Bull., 35(1):40–45, 2012.

[KHL17] Karnagel, Tomas; Habich, Dirk; Lehner, Wolfgang: Adaptive Work Placement for Query Processing on Heterogeneous Computing Resources. PVLDB, 10(7):733–744, 2017.

[KHL18] Kissinger, Thomas; Habich, Dirk; Lehner, Wolfgang: Adaptive Energy-Control for In-Memory Database Systems. In: SIGMOD. pp. 351–364, 2018.

[Ki13] Kissinger, Thomas; Schlegel, Benjamin; Habich, Dirk; Lehner, Wolfgang: QPPT: Query Processing on Prefix Trees. In: CIDR. 2013.

[Ki14] Kissinger, Thomas; Kiefer, Tim; Schlegel, Benjamin; Habich, Dirk; Molka, Daniel; Lehner, Wolfgang: ERIS: A NUMA-Aware In-Memory Storage Engine for Analytical Workload. In: ADMS. pp. 74–85, 2014.

[KML15] Karnagel, Tomas; Müller, René; Lohman, Guy M.: Optimizing GPU-accelerated Group-By and Aggregation. In: ADMS. pp. 13–24, 2015.

[Ko18] Komatsu, Kazuhiko; Momose, Shintaro; Isobe, Yoko; Watanabe, Osamu; Musa, Akihiro; Yokokawa, Mitsuo; Aoyama, Toshikazu; Sato, Masayuki; Kobayashi, Hiroaki: Performance evaluation of a vector supercomputer SX-aurora TSUBASA. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2018, Dallas, TX, USA, November 11-16, 2018. pp. 54:1–54:12, 2018.

[Li16] Li, Feng; Das, Sudipto; Syamala, Manoj; Narasayya, Vivek R.: Accelerating Relational Databases by Leveraging Remote Memory and RDMA. In: SIGMOD. pp. 355–370, 2016.

[Li18] Lisa, Nusrat Jahan; Ungethüm, Annett; Habich, Dirk; Lehner, Wolfgang; Nguyen, Tuan D. A.; Kumar, Akash: 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. pp. 22–33, 2018.

[LP13] Li, Yinan; Patel, Jignesh M.: BitWeaving: Fast Scans for Main Memory Data Processing. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. SIGMOD '13, ACM, New York, NY, USA, pp. 289–300, 2013.

[LUH18] Lehner, Wolfgang; Ungethüm, Annett; Habich, Dirk: Diversity of Processing Units - An Attempt to Classify the Plethora of Modern Processing Units. Datenbank-Spektrum, 18(1):57–62, 2018.

[Ou17] Oukid, Ismail; Booss, Daniel; Lespinasse, Adrien; Lehner, Wolfgang; Willhalm, Thomas; Gomes, Grégoire: Memory Management Techniques for Large-Scale Persistent-Main-Memory Systems. PVLDB, 10(11):1166–1177, 2017.

[PRR15] Polychroniou, Orestis; Raghavan, Arun; Ross, Kenneth A.: Rethinking SIMD Vectorization for In-Memory Databases. In: SIMD. pp. 1493–1508, 2015.

[St05] Stonebraker, Michael; Abadi, Daniel J.; Batkin, Adam; Chen, Xuedong; Cherniack, Mitch; Ferreira, Miguel; Lau, Edmond; Lin, Amerson; Madden, Samuel; O'Neil, Elizabeth J.; O'Neil, Patrick E.; Rasin, Alex; Tran, Nga; Zdonik, Stanley B.: C-Store: A Column-oriented DBMS. In: VLDB. pp. 553–564, 2005.

[Zu06] Zukowski, Marcin; Héman, Sándor; Nes, Niels; Boncz, Peter A.: Super-Scalar RAM-CPU Cache Compression. In: ICDE. p. 59, 2006.

[ZvdWB12] Zukowski, Marcin; van de Wiel, Mark; Boncz, Peter A.: Vectorwise: A Vectorized Analytical DBMS. In: ICDE. pp. 1349–1350, 2012.