FADS @VLDB 2017

Single Instruction Multiple Data – Not Everything is a Nail for this Hammer

David Broneske
University of Magdeburg
Magdeburg, Germany
david.broneske@ovgu.de

Martin Schäler
Karlsruhe Institute of Technology
Karlsruhe, Germany
martin.schaeler@kit.edu

ABSTRACT

Hardware vendors have been struggling to fight the power and memory wall for decades [1, 2]. Since most of the processing time depends on the number of instructions, number of used registers and dependencies between instructions, but not on the size of a register, independent data items of a vector (i.e., a column) could be processed in parallel. Hence, a silver lining seems to be *Single Instruction Multiple Data* (SIMD) – a processing paradigm available on current CPUs, but also accelerator cards such as GPUs and MICs (i.e., Intel Xeon Phi). For instance, aggregations could perform sum or count instructions on several data items in parallel. By loading four 32-bit integers in an 128-bit SSE register and performing the addition in one cycle for all four data items, a four-fold performance benefit should be possible. However, these high expectations are rarely met in practice.

In this talk, we elaborate about pitfalls that we encountered while optimizing database operators with SIMD. Overall, these pitfalls can be found at different levels: especially the *data movement* within the operator and the *data layout* plays a vital role for the performance improvements.

Data Movement

A primary challenge is to avoid mixing vectorized (i.e., SIMD) and scalar code as this results in moving data from SSE registers to normal registers. For instance we implemented a vectorized selection with a position list as a result. The vectorized predicate evaluation produces a bit mask, which has to be evaluated in a scalar fashion to produce the position list. However, our results show that this is mostly inefficient on several modern processors [3]. In particular, the vectorized scan that we used has a performance penalty between Factor 0.2 - 2, while it only outperforms the scalar version for selectivity factors of 0.05 and less.

To reduce data movement and to get the best out of SIMD, operators should reuse the content of a SIMD register as often as possible. That is multiple operators should operate on current data in the register. For instance, when using query compilation, several selection predicates can be merged in order to reuse the intermediate bit mask [5]. Moreover, if we add also aggregations to the code, the usability of SIMD increases further (i.e., the selectivity range in which the vectorized version outperforms the scalar one).

Data Layout

SIMD operates best if the vector content is aligned to 16-bit boundaries, because unaligned reads will lead to accesses across cache lines which may bring a penalty.[1] Hence, data alignment is a vital task, which becomes complicated for data structures such as indexes. For our index structure Elf [4] having an explicit memory layout, we tested several linearization strategies, but faced three main problems: (1) Storing node entries at aligned storage will blow up the size of the structure due to padding space. (2) Storing values and pointers in an intermixed fashion diminishes the n-fold performance benefit while separate storage leads to an extra cache miss. (3) SIMD does not work well for nodes with little amount of entries, because the glue code deteriorates the performance benefits.

In summary, SIMD performs best for operators that do the whole work using SIMD with little or no amount of scalar code. Furthermore, a clever data layout is necessary to exploit SIMD at most – this does not only apply to tree-based index structures, but also hash tables.

1. REFERENCES

[1] C. Balkesen, G. Alonso, J. Teubner, and M. T. Özsu. Multi-core, main-memory joins: Sort vs. hash revisited. *PVLDB*, 7(1):85–96, 2013.

[2] D. Broneske, S. Breß, M. Heimel, and G. Saake. Toward hardware-sensitive database operations. In *EDBT*, pages 229–234, 2014.

[3] D. Broneske, S. Breß, and G. Saake. Database scan variants on modern CPUs: A performance study. In *VLDB Workshop IMDM*, volume 8921 of *LNCS*, pages 97–111. Springer, 2014.

[4] D. Broneske, V. Köppen, G. Saake, and M. Schäler. Accelerating multi-column selection predicates in main-memory - the Elf approach. In *ICDE*, pages 647–658, April 2017.

[5] D. Broneske, A. Meister, and G. Saake. Hardware-sensitive scan operator variants for compiled selection pipelines. In *BTW*, 2017.

[1] Recent CPU architectures are said to have the same performance for unaligned as for aligned memory access.