B. Mitschang et al. (Hrsg.): Datenbanksysteme für Business, Technologie und Web (BTW 2017),
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 507

Query Processing and Optimization in Modern Database Systems

Viktor Leis[1]

Abstract: Relational database management systems, which were designed decades ago, are still the dominant data processing platform. Since then, large DRAM capacities and servers with many cores have fundamentally changed the hardware landscape. As a consequence, traditional database systems cannot exploit modern hardware effectively anymore. This paper summarizes author's thesis, which focuses on the challenges posed by modern hardware for transaction processing, query processing, and query optimization. In particular, we present a concurrent transaction processing system based on hardware transactional memory and show how to synchronize data structures efficiently. We further design a parallel query engine for many-core CPUs that supports the important relational operators including join, aggregation, window functions, etc. Finally, we dissect the query optimization process in the main memory setting and show the contribution of each query optimizer component to the overall query performance.

1 The Architecture of Relational Database Systems

Relational database management systems have stood the test of time and are still the dominant data processing platform. The basic design of these systems stems from the 1980s and was largely unchanged for decades. The core ideas include row-wise storage as well as B-trees on fixed-sized pages backed by a buffer pool, ARIES-style logging, and Two Phase Locking. Recent years, however, have seen many of the design decisions become obsolete due to fundamental changes in the hardware landscape. Before describing the contributions of the author's thesis [Le16a], we give a brief outline of modern database systems and discuss some of the challenges posed by modern hardware.

1.1 Column Stores

After decades of only minor, incremental changes to the basic database architecture, a radically new design, column stores, started to gain traction in the years after 2005. C-Store [St05] (commercialized as Vertica) and MonetDB/X100 [BZN05] (commercialized as Vectorwise) are two influential systems that gained significant mind share during that time frame. The idea of organizing relations by column is, of course, much older [BMK09]. Sybase IQ [MF04] and MonetDB [BQK96] are two pioneering column stores that originated in the 1990s.

Column stores are read-optimized and often used as data warehouses, i.e., non-operational databases that ingest changes periodically (e.g., every night). In comparison with row stores,

[1] Technische Universität München, leis@in.tum.de

column stores have the obvious advantage that scans only need to read those attributes accessed by a particular query resulting in less I/O operations. A second advantage is that the query engine of a column store can be implemented in a much more CPU-efficient way: Column stores can amortize the interpretation overhead of the iterator model by processing batches of rows ("vector-at-a-time"), instead of working only on individual rows ("tuple-at-a-time").

The major database vendors have reacted to the changing landscape by combining multiple storage and query engines in their products. In Microsoft SQL Server, for example, users now can choose between the

- traditional general-purpose row store,
- a column store [La11b] for OnLine Analytical Processing (OLAP), and
- in-memory storage optimized for Online transaction processing (OLTP) [Di13].

Each of these options comes with its own query processing model and specific performance characteristics, which must be carefully considered by the database administrator.

The impact of column stores can be seen in the official benchmark numbers of TPC-H, which is a widely used OLAP benchmark. Before 2011, multiple vendors competed for the TPC-H crown, with the lead changing from time to time between Oracle, Microsoft, IBM, and Sybase. This changed with the arrival of Actian Vectorwise in 2011, which disrupted the incremental "rat race" between the traditional vendors by doubling the reported performance. The dominance of Vectorwise as official TPC-H leader lasted until 2014, when Microsoft submitted new results with their column store engine Apollo [La11b], which has been the leading system in 2016.

1.2 Main-Memory Database Systems

The lower CPU overhead of column store query engines was of only minor importance as long as data was mainly stored on disk (or even SSD). In 2000 one had to pay over $1000 for 1 GB of DRAM[2]. At these prices, any non-trivial database workload resulted in a significant number of disk I/O operations, and main-memory DBMSs—which were a research topic as early as the 1980s [GS92]—were still niche products. In 2008, with the same $1000 one could already buy 100 GB of RAM[3]. This rapid decrease in DRAM prices had consequences for the architecture of database management systems.

Harizopoulos et al.'s paper from 2008 [Ha08] showed that on the—suddenly very common—memory-resident OLTP workloads virtually all time was wasted on overhead like

- buffer management,

[2] DRAM prices are taken from `http://www.jcmit.com/memoryprice.htm`.
[3] The cost continues to decline. At the time of writing, in 2016, the cost was around $4 per GB.

- locking,
- latching,
- heavy-weight logging, and
- an inefficient implementation.

The goal of any database system's designer thus gradually shifted from minimizing the number of disk I/O operations to reducing CPU overhead and cache misses. This lead to a resurgence of research into main-memory database systems. The main idea behind main-memory DBMSs is to assume that all data fits into RAM and to optimize for CPU and cache efficiency. Using careful engineering and by making the right architectural decisions that take modern hardware into account, database systems can achieve orders of magnitude higher performance. Well-known main-memory database systems include H-Store/VoltDB [Ka08, SW13], SAP HANA [Fä11], Microsoft Hekaton [La13], Oracle TimesTen [LNF13], Calvin [Th12], Silo [Tu13], MemSQL, and HyPer [KN11].

1.3 HyPer

The author's thesis [Le16a] has been done in the context of the HyPer project, which started in 2010 [KN10]. HyPer follows some of the design decisions of other main-memory systems (e.g., no buffer manager, no locks, no latches, and (originally) command logging). To avoid fine-grained latches, HyPer also initially followed H-Store's approach of relying on user-controlled, physical partitioning of the database to enable multi-threading.

HyPer has, however, a number of features that distinguish it from many other main-memory systems: From the very beginning, HyPer supported both OLTP and OLAP in the same database in order to make the physical separation between the transactional and data warehouse databases obsolete. Initially, HyPer used OS-supported snapshots [KN11], which were later replaced with a software-controlled Multi-Version Concurrency Control (MVCC) approach [NMK15]. The second unique feature of HyPer is that, via the LLVM [LA04] compiler infrastructure, it compiles SQL queries and stored procedures to machine code [Ne11, NL14]. Compilation avoids the interpretation overhead inherent in the iterator model and thereby enables extremely high performance. LLVM is a widely used open source compiler backend that can generate efficient machine code for many different target platforms, which makes this approach portable. In contrast to previous compilation approaches (e.g., [KVC10]), HyPer compiles multiple relational operators from the same query pipeline into a single intertwined code fragment, which allows it to keep values in CPU registers for as long as possible.

In terms of architecture, most column stores have converged to a similar design [Ab13], which was pioneered by systems like Vectorwise [BZN05] and Vertica [St05]. In-memory OLTP systems, in contrast, show more architectural variety. Compilation is, however, becoming a common building block for OLTP systems, as can be observed by the use of compilation by HyPer [Ne11], Hekaton [Di13], and MemSQL. Other high-performance

systems like Silo [Tu13] also implicitly assume (but do not yet implement) compilation, as the stored procedures are hand-written in C or C++ in these systems. In other areas like concurrency control (e.g., [Tu13] vs. [La11a] vs. [NMK15]), indexing (e.g., [LKN13] vs. [MKM12] vs. [LLS13]), and logging (e.g., [Ma14] vs. physiological) there is much more variety between the systems.

2 The Challenges of Modern Hardware

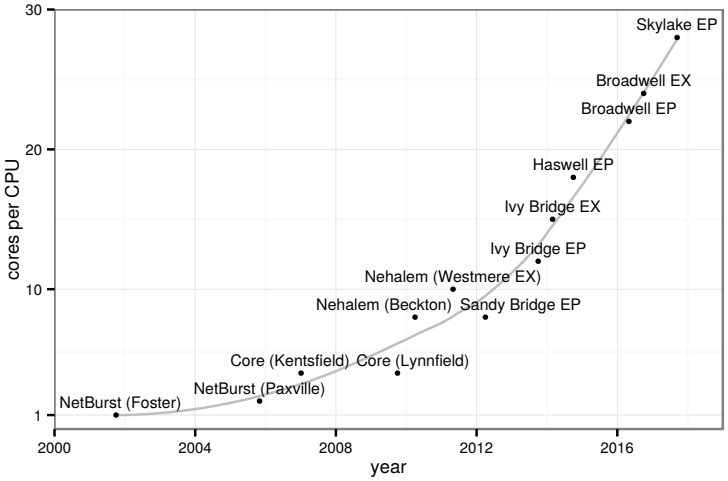

Fig. 1: Number of cores in Intel server processors (largest configuration in each microarchitecture)

Besides increasing main-memory sizes, a second important trend in the hardware landscape is the ever increasing number of cores. Figure 1 shows the number of cores for server CPUs[4]. Over the entire time frame, the clock rate stayed between 2 GHz and 3 GHz and, as a result, single-threaded performance increased only very slowly (by single-digit percentages per year). Note that the graph only shows "real" cores for a single socket. Many servers have 2, 4, or even 8 sockets in a single system and each Intel core nowadays has 2-way HyperThreading. As a result, the affordable and commonly used 2-socket configurations will soon routinely have over 100 hardware threads in a single system. Memory bandwidth has largely kept up with the increasing number of cores and will reach over 100 GB/s per socket with Skylake EP. However, it is important to note that a single core can only utilize a small fraction of the available bandwidth, making effective parallelization essential.

Long before the many-core revolution, high-end database servers often combined a handful of processors—connected by a shared memory bus—in a Symmetric Multi-Processing (SMP) system. Furthermore, database systems have, for a long time, been capable of executing queries concurrently by using appropriate locking and latching techniques. So one

[4] The data is from `https://en.wikipedia.org/wiki/List_of_Intel_Xeon_microprocessors`. For Broadwell EX and Skylake EP server CPUs we show estimates from the press as they were not yet released at the time of writing.

might reasonably ask if any fundamental changes to the database architecture are required at all. Modern hardware, however, has unique challenges not encountered in the past:

Latches are expensive and prevent scaling. Traditional database systems use latches extensively to access shared data structures from concurrent threads. As long as disk I/O operations were frequent, the overhead of short-term latching was negligible. On modern hardware, however, even short-term, uncontested latches can be expensive and prevent scalability. The reason is that each latch acquisition causes cache line invalidations for *all other cores*. As we show experimentally, this effect often prevents scalability on multi-core CPUs.

Intra-query parallelism is not optional any more. For a long time, many systems relied on parallelism from the "outside", i.e., inter-query parallelism. With dozens or hundreds of cores, intra-query parallelism is not an optional optimization because many workloads simply do not have enough parallel query sessions. Without intra-query parallelism, the computational resources of modern servers lie dormant. The widely used PostgreSQL system, for example, will finally introduce (limited) intra-query parallelism in the upcoming version 9.6—20 years after the project started.

Query engines should be designed with multi-core parallelism in mind. Some commercial systems added support for intra-query parallelism a decade ago. This was often done by introducing "exchange" operators [Gr90] that encapsulate parallelism without redesigning the actual operators. This pragmatic approach was sufficient at a time when the degree of parallelism in database servers was low (e.g., 10 threads). To get good scalability on systems with dozens of cores, the query processing algorithms should be redesigned from scratch with parallelism in mind.

Database systems should take Non-Uniform Memory Architecture (NUMA) into account. In contrast to earlier SMP systems, where all processors shared a common memory bus, current systems are generally based on the Non-Uniform Memory Architecture (NUMA). In this architecture each processor has its own memory, but can transparently and cache-coherently access remote memory through an interconnect. Because remote memory accesses are more expensive than local accesses, NUMA-aware data placement can improve performance considerably. Thus, database systems must optimize for NUMA to obtain optimal performance.

Together, these changes explain why traditional systems (e.g., as described in [HSH07]) cannot fully exploit the resources provided today's commodity servers. To utilize modern hardware well, fundamental changes to core database components including storage, concurrency control, low-level synchronization, query processing, logging, etc. are necessary. Database systems specifically designed for modern hardware can be orders of magnitude faster than their predecessors.

3 Contributions

The author's thesis [Le16a] addresses the challenges enumerated above. The solutions were developed within a general-purpose, relational database system (HyPer) and most experiments measure end-to-end performance. Our contributions span the transaction processing, query processing, and query optimization components.

We design a low-overhead, **concurrent transaction processing** engine based on Hardware Transactional Memory (HTM) [LKN14, LKN16]. Until recently, transactional memory—although a promising technique—suffered from the absence of an efficient hardware implementation. Since Intel introduced the Haswell microarchitecture hardware transactional memory is available in mainstream CPUs. HTM allows for efficient concurrent, atomic operations, which is also highly desirable in the context of databases. On the other hand, HTM has several limitations that, in general, prevent a one-to-one mapping of database transactions to HTM transactions. We devise several building blocks that can be used to exploit HTM in main-memory databases. We show that HTM allows one to achieve nearly lock-free processing of database transactions by carefully controlling the data layout and the access patterns. The HTM component is used for detecting the (infrequent) conflicts, which allows for an optimistic—and thus very low-overhead execution—of concurrent transactions. We evaluate our approach on a 4-core desktop and a 28-core server system and find that HTM indeed provides a scalable, powerful, and easy to use synchronization primitive.

While Hardware Transactional Memory is easy to use and can offer good performance, it is not yet widespread. Therefore, we study alternative **low-overhead synchronization** mechanisms for in-memory data structures [Le16b]. The traditional approach, fine-grained locking, does not scale on modern hardware. Lock-free data structures, in contrast, scale very well but are extremely difficult to implement and often require additional indirections. We argue for a middle ground, i.e., synchronization protocols that use locking, but only sparingly. We synchronize the Adaptive Radix Tree (ART) [LKN13] using two such protocols, Optimistic Lock Coupling and Read-Optimized Write EXclusion (ROWEX). Both perform and scale very well while being much easier to implement than lock-free techniques.

We describe the **parallel and NUMA-aware query engine** of HyPer, which scales up to dozens of cores [Le14]. Our "morsel-driven" query execution framework, where scheduling becomes a fine-grained run-time task that is NUMA-aware. Morsel-driven query processing takes small fragments of input data ("morsels") and schedules these to worker threads that run entire operator pipelines until the next pipeline-breaking operator. The degree of parallelism is not baked into the plan but can elastically change during query execution. The dispatcher can react to the execution speed of different morsels but also adjust resources dynamically in response to newly arriving queries in the workload. Furthermore, the dispatcher is aware of data locality of the NUMA-local morsels and operator state, such that the great majority of executions takes place on NUMA-local memory. Our evaluation on the TPC-H and SSB benchmarks shows extremely high absolute performance and an average speedup of over 30 with 32 cores.

We complete the description of HyPer's query engine by proposing a design for the **SQL:2003 window function operator** [Le15b]. Window functions, also known as analytic OLAP functions, have been neglected in the research literature—despite being part of the SQL standard for more than a decade and being a widely-used feature. Window functions can elegantly express many useful queries about time series, ranking, percentiles, moving averages, and cumulative sums. Formulating such queries in plain SQL-92 is usually both cumbersome and inefficient. Our algorithm is optimized for high-performance main-memory database systems and has excellent performance on modern multi-core CPUs. We show how to fully parallelize all phases of the operator in order to effectively scale for arbitrary input distributions.

The only thing more important for achieving low query response times than a fast and scalable query engine is **query optimization**. We shift our focus from the query engine to the query optimizer [Le15a]. Query optimization has been studied for decades, but most experiments were in the context of disk-based systems or were focused on individual query optimization components rather than end-to-end performance. We introduce the Join Order Benchmark (JOB) and experimentally revisit the main components in the classic query optimizer architecture using a complex, real-world data set and realistic multi-join queries. We investigate the quality of industrial-strength cardinality estimators and find that all estimators routinely produce large errors. We further show that while estimates are essential for finding a good join order, query performance is unsatisfactory if the query engine relies too heavily on these estimates. Using another set of experiments that measure the impact of the cost model, we find that it has much less influence on query performance than the cardinality estimates. Finally, we investigate plan enumeration techniques comparing exhaustive dynamic programming with heuristic algorithms and find that exhaustive enumeration improves performance despite the sub-optimal cardinality estimates.

4 Future Work

We have shown that a modern database system that is carefully optimized for modern hardware can achieve orders of magnitude higher performance than a traditional design. However, there are still many unsolved problems, some of which we plan to address in the future.

One important research frontier nowadays lies in supporting mixed workloads in a single database. Many systems that start out as pure OLTP systems, over time add OLAP features thus blurring the distinction between the two system types. While there may be sound technical reasons for running OLTP and OLAP in separate systems, in reality, OLTP and OLAP are more platonic ideals than truly separate applications. One major consequence is that, even for main-memory database systems, the convenient assumption that *all* data fits into RAM generally does not hold. Despite a number of recent proposals, we believe that the general problem of efficiently maintaining a global replacement strategy over relational as well as index data is still not fully solved.

Major changes are also happening on the hardware side and database systems must keep evolving to benefit from these changes. One aspect is the ever increasing number of cores per CPU. While it is not clear whether servers with 1000 cores will be common in the near future—if this indeed happens—it will have a major effect on database architecture. It is a general rule, that the higher the degree of parallelism, the more difficult scalability becomes. Any efficient system that scales up to, for example, 100 cores, will likely require some major changes to scale up to 1000 cores. Thus, some of the architectural decisions may need to be revised if the many-core trend continues.

A potentially even greater challenge is the increasing heterogeneity of modern hardware, which has the potential of disrupting the architecture of database systems. From the point of view of a software developer, in the past, software became faster "automatically" due to increasing clock frequencies. The hardware technologies mentioned above have, however, one thing in common: Programmers have to invest effort to get any benefit from them. Programming for SIMD, GPUs, or FPGAs is very different (and more difficult) than using the instruction set of a conventional, general-purpose CPU. A database system will have to decide which part of a query should be executed on which device, all while managing the data movement between the devices and the energy/heat budget of the overall system. Put simply, no one currently knows how to do this and no doubt it will take considerable research effort to find a satisfactory solution.

Whereas the technologies mentioned above promise faster computation, new storage technologies like PCIe-attached NAND flash and non-volatile memory (NVM) like Phase Change Memory threaten to disrupt the way data is stored and accessed. In order to avoid changing the software stack much, it is certainly possible to hide modern storage devices behind a conventional block device interface. However, this approach leaves performance on the table as it ignores the specific physical properties like the block erase requirement of NAND flash or the byte-addressability of non-volatile memory. Thus, research is required to find out how to best utilize these new storage technologies.

Finally, even the venerable field of query optimization still has many unsolved problems. One promising approach is to rely more heavily on sampling, which is much cheaper than in the past when CPU cycles were costly and random disk I/O would have been required. Sampling, for example across indexes, opens up new ways to estimate the cardinality of multi-way joins [Le17], which after decades of research, is still done naively in most systems.

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), 2013.

[BMK09] Boncz, Peter A.; Manegold, Stefan; Kersten, Martin L.: Database Architecture Evolution: Mammals Flourished long before Dinosaurs became Extinct. PVLDB, 2(2), 2009.

[BQK96] Boncz, Peter A.; Quak, Wilko; Kersten, Martin L.: Monet And Its Geographic Extensions: A Novel Approach to High Performance GIS Processing. In: EDBT. 1996.

[BZN05] Boncz, Peter; Zukowski, Marcin; Nes, Niels: MonetDB/X100: Hyper-Pipelining Query Execution. In: CIDR. 2005.

[Di13] Diaconu, Cristian; Freedman, Craig; Ismert, Erik; Larson, Per-Åke; Mittal, Pravin; Stonecipher, Ryan; Verma, Nitin; Zwilling, Mike: Hekaton: SQL server's memory-optimized OLTP engine. In: SIGMOD. 2013.

[Fä11] Färber, Franz; Cha, Sang Kyun; Primsch, Jürgen; Bornhövd, Christof; Sigg, Stefan; Lehner, Wolfgang: SAP HANA database: data management for modern business applications. SIGMOD Record, 40(4), 2011.

[Gr90] Graefe, Goetz: Encapsulation of Parallelism in the Volcano Query Processing System. In: SIGMOD. 1990.

[GS92] Garcia-Molina, Hector; Salem, Kenneth: Main Memory Database Systems: An Overview. IEEE Trans. Knowl. Data Eng., 4(6), 1992.

[Ha08] Harizopoulos, Stavros; Abadi, Daniel J.; Madden, Samuel; Stonebraker, Michael: OLTP through the looking glass, and what we found there. In: SIGMOD. 2008.

[HSH07] Hellerstein, Joseph M.; Stonebraker, Michael; Hamilton, James R.: Architecture of a Database System. Foundations and Trends in Databases, 1(2), 2007.

[Ka08] Kallman, Robert; Kimura, Hideaki; Natkins, Jonathan; Pavlo, Andrew; Rasin, Alex; Zdonik, Stanley B.; Jones, Evan P. C.; Madden, Samuel; Stonebraker, Michael; Zhang, Yang; Hugg, John; Abadi, Daniel J.: H-store: a high-performance, distributed main memory transaction processing system. PVLDB, 1(2), 2008.

[KN10] Kemper, Alfons; Neumann, Thomas: HyPer - Hybrid OLTP&OLAP High Performance Database System. Technical report, TUM, 2010.

[KN11] Kemper, Alfons; Neumann, Thomas: HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In: ICDE. 2011.

[KVC10] Krikellas, Konstantinos; Viglas, Stratis; Cintra, Marcelo: Generating code for holistic query evaluation. In: ICDE. 2010.

[LA04] Lattner, Chris; Adve, Vikram: LLVM: A Compilation Framework for Lifelong Program Analysis and Transformation. In: CGO. Mar 2004.

[La11a] Larson, Per-Åke; Blanas, Spyros; Diaconu, Cristian; Freedman, Craig; Patel, Jignesh M.; Zwilling, Mike: High-Performance Concurrency Control Mechanisms for Main-Memory Databases. PVLDB, 5(4), 2011.

[La11b] Larson, Per-Åke; Clinciu, Cipri; Hanson, Eric N.; Oks, Artem; Price, Susan L.; Rangarajan, Srikumar; Surna, Aleksandras; Zhou, Qingqing: SQL Server column store indexes. In: SIGMOD. 2011.

[La13] Larson, Per-Åke; Zwilling, Mike; ; Farlee, Kevin: The Hekaton Memory-Optimized OLTP Engine. IEEE Data Eng. Bull., 36(2), 2013.

[Le14] Leis, Viktor; Boncz, Peter; Kemper, Alfons; Neumann, Thomas: Morsel-Driven Parallelism: A NUMA-Aware Query Evaluation Framework for the Many-Core Age. In: SIGMOD. 2014.

[Le15a] Leis, Viktor; Gubichev, Andrey; Mirchev, Atanas; Boncz, Peter; Kemper, Alfons; Neumann, Thomas: How Good Are Query Optimizers, Really? PVLDB, 9(3), 2015.

[Le15b] Leis, Viktor; Kundhikanjana, Kan; Kemper, Alfons; Neumann, Thomas: Efficient Processing of Window Functions in Analytical SQL Queries. PVLDB, 8(10), 2015.

[Le16a] Leis, Viktor: Query Processing and Optimization in Modern Database Systems. Disseration, Technische Universität München, 2016.

[Le16b] Leis, Viktor; Scheibner, Florian; Kemper, Alfons; Neumann, Thomas: The ART of practical synchronization. In: DaMoN. 2016.

[Le17] Leis, Viktor: Cardinality Estimation Done Right: Index-Based Join Sampling. In: CIDR. 2017.

[LKN13] Leis, Viktor; Kemper, Alfons; Neumann, Thomas: The Adaptive Radix Tree: ARTful Indexing for Main-Memory Databases. In: ICDE. 2013.

[LKN14] Leis, Viktor; Kemper, Alfons; Neumann, Thomas: Exploiting hardware transactional memory in main-memory databases. In: ICDE. 2014.

[LKN16] Leis, Viktor; Kemper, Alfons; Neumann, Thomas: Scaling HTM-Supported Database Transactions to Many Cores. IEEE Trans. Knowl. Data Eng., 28(2), 2016.

[LLS13] Levandoski, Justin J.; Lomet, David B.; Sengupta, Sudipta: The Bw-Tree: A B-tree for new hardware platforms. In: ICDE. 2013.

[LNF13] Lahiri, Tirthankar; Neimat, Marie-Anne; Folkman, Steve: Oracle TimesTen: An In-Memory Database for Enterprise Applications. IEEE Data Eng. Bull., 36(2), 2013.

[Ma14] Malviya, Nirmesh; Weisberg, Ariel; Madden, Samuel; Stonebraker, Michael: Rethinking main memory OLTP recovery. In: ICDE. 2014.

[MF04] MacNicol, Roger; French, Blaine: Sybase IQ Multiplex - Designed For Analytics. In: VLDB. 2004.

[MKM12] Mao, Yandong; Kohler, Eddie; Morris, Robert Tappan: Cache craftiness for fast multicore key-value storage. In: EuroSys. 2012.

[Ne11] Neumann, Thomas: Efficiently Compiling Efficient Query Plans for Modern Hardware. PVLDB, 4(9), 2011.

[NL14] Neumann, Thomas; Leis, Viktor: Compiling Database Queries into Machine Code. IEEE Data Eng. Bull., 37(1), 2014.

[NMK15] Neumann, Thomas; Mühlbauer, Tobias; Kemper, Alfons: Fast Serializable Multi-Version Concurrency Control for Main-Memory Database Systems. In: SIGMOD. 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. 2005.

[SW13] Stonebraker, Michael; Weisberg, Ariel: The VoltDB Main Memory DBMS. IEEE Data Eng. Bull., 36(2), 2013.

[Th12] Thomson, Alexander; Diamond, Thaddeus; Weng, Shu-Chun; Ren, Kun; Shao, Philip; Abadi, Daniel J.: Calvin: fast distributed transactions for partitioned database systems. In: SIGMOD. 2012.

[Tu13] Tu, Stephen; Zheng, Wenting; Kohler, Eddie; Liskov, Barbara; Madden, Samuel: Speedy transactions in multicore in-memory databases. In: SOSP. 2013.