Hasso Plattner Institute
Chair for Data Engineering Systems

Proposal Master's Thesis

Hardware-Conscious SIMD-Accelerated Sort-Merge Joins in Multi Core In-Memory Database Systems

Finn Schöllkopf

Time frame: October 2024 - March 2024

Supervisor
Prof. Dr. Tilmann Rabl

Advisor
Florian Schmeller
Martin Boissier

1 Motivation

Historically, many systems were designed for computer systems and architectures where I/O dominates performance. However, modern systems with multi-core architectures, larger DRAM capacities, advanced instruction sets, and other hardware accelerants like vector operations (SIMD), which allow us to perform the same operation on multiple data items simultaneously, have significantly altered this landscape. Hence, we must reconsider how we implement modern systems.

1.1 Importance for Database Systems

In-memory database systems, which store data directly in the main memory rather than on disk, provide significantly faster data access and processing due to reduced latency. Therefore, they are no longer I/O bound and, need high intra-operator parallelism. To fully utilize the multi-core architecture and other hardware features such as cache locality and SIMD instructions for higher data parallelism should be considered to achieve maximum performance.

1.2 Efficient Join Implementation

In this thesis we want to look at one common database operator: the join [6]. The join operator is a fundamental component of every database system. In recent years, the difference in performance between the sort-merge and radix-hash join has been the subject of ongoing debate. Kim et al. [11] projected that Sort-Merge Join would outperform hash-based alternatives with 512-bit SIMD. Albutiu et al. [1] reinforced this claim with recent results reporting that their implementation of sort-merge join is superior to that of hash joins (without leveraging SIMD). Balkesen et al. [2] experimentally show contradicting results by implementing optimized versions for sort-merge and radix-hash join, showing that their implementation of radix-hash join is still superior.

1.3 Open-Source Implementation

Despite ongoing research, public implementations of join algorithms optimized for modern hardware are hard to find. Most existing implementations are proprietary or experimental[1], limiting their accessibility and usefulness to the research community and database developers. An open-source, state-of-the-art implementation of a sort-merge join optimized for different architectures would help address this need. Such an implementation serves as a valuable baseline for researchers looking to evaluate or improve upon existing methods, and it also contributes to advancing database system design by providing a solid foundation for future innovation.

[1]Implementation by Balkesen et al.: `https://archive-systems.ethz.ch/node/334`

2 Goal of Thesis

This thesis aims to efficiently implement the sort-merge join algorithm, explicitly optimized for specific architectures and hardware components. As equi-joins are the most common type of join operation, we will restrict ourselves to an equi-join implementation and then optionally extend upon this. While multiple papers exist about modern implementation approaches for sort-merge joins in in-memory database systems and SIMD sorting, only some have public implementations[2]. The goals of this thesis include an open-source implementation of a sort-merge join algorithm integrated into the Hyrise in-memory database. The sorting should be accelerated through SIMD parallelism and support AVX2 and AVX-512. The implementation should support int, float, and string data. We also want to experiment with other architectures like Arm and Power and run benchmarks to evaluate our approach.

2.1 Hyrise Integration

While some public implementations exist for modern and optimized sort-merge join, they have usually isolated implementations with a strong focus on the sorting step using randomly chosen input data, often already in the required data format. Also, they often skip the lookup of matching rows and the construction of the joined table. Hence, in this thesis, we want to integrate our implementation of the sort-merge join into Hyrise [9], a research in-memory database. Hyrise contains both a radix-based hash-join and sort-merge join. The sort-merge join uses radix cluster sorting, which uses "pdqsort" (Boost C++ library) but no explicit SIMD instructions. It fundamentally differs from the modern approaches in the literature. These differences allow us to test our implementation against the existing sort-merge and hash-based join.

2.2 SIMD Sorting

Most SIMD sorting algorithms presented in the literature are not directly applicable to join operations as they usually use sorting keys of only 32 bits. We must additionally track the row ID (rid) corresponding to the sorting key for a join, resulting in 64-bit elements. Most architectures only support 8-, 16-, 32-, and 64-bit elements for SIMD. Therefore, larger elements like 128-bit are usually not supported.

The current implementations of sort-merge join in literature use SSE and AVX2 intrinsics, but to our knowledge, there has yet to be an implementation using AVX-512. Therefore, in the scope of this thesis, we want to integrate support for modern AVX-512 sorting algorithms [17, 18].

[2]Implementation of [2] published at `https://archive-systems.ethz.ch/node/334`

2.3 Architectures

It would also be of value to see how new and existing approaches transfer to other CPU architectures like Arm with its Scalable Vector Extension (SVE) or Power with its Vector Scalar Extension (VSX).

Hence, in addition to x86 systems, we optinally want to experiment with AWS Graviton 4 and Power10. This requires adapting the implementation to use the respective architecture's vector extension (e.g., ARM SVE or Power VSX).

2.4 Benchmarking

Complete integration into an in-memory database allows us to run decision support benchmarks like TCP-H, TCP-DS, and the Join Order Benchmark (JOB) [12] to compare operators to other implementations in a more realistic scenario.

Benchmarks like TCP-H have schemas and datatypes carefully designed by experts in database design. Hence, they can fail to capture the chaotic nature of real-world applications [16]. For instance, TCP-H only uses integer values for keys.

However, many Business Intelligence applications use strings for various data types, e.g. to deal with dirty data that is not parsable. Hence, we frequently encounter string-type join keys. For that, we chose JOB due to its real-world data and ease of use, as it has already been integrated into Hyrise. The Public BI benchmark[3] also provides a realistic setting but is more difficult to use, and we only consider it optional.

Benchmarking should also include measuring the sorting throughput in tuples per second and all algorithmic steps: initial data construction in the format of (key, rid) from the input relations, sorting, finding join partners, and the final construction of the joined table.

3 Approach

The sort-merge join involves sorting both input relations. It is the most crucial and time-consuming part of the sort-merge join operation. Therefore, high optimization efforts should be spend on this step, as it largely determines the runtime.

Due to modern multi-core architectures, sorting should intensively utilize thread-level parallelism by multithreading. With the recent architectural trends of wider register widths for SIMD, sorting should also heavily use SIMD instructions to exploit data parallelism.

[3]`https://github.com/cwida/public_bi_benchmark`

3.1 Selection of Mergesort

In a multi-core context, we prefer mergesort over quicksort, as the parallelization of the divide-and-conquer approach is straightforward, and it has other advantages over quicksort such as more predictable and cache-friendly memory access patterns and better load balancing through equal-sized partitioning.

3.2 Data Preparation

Before we can sort our input relations, the tuples need to be translated into a SIMD sortable format. Usually, a 64-bit pair (key, rid) (key & rid both 32-bit) is assumed. We, therefore, support a maximum relation size of 2^{32}. With most value types greater than 32 bits, we need to compress the values of the join columns to 32 bits. Methods like key-prefix [14] and XOR- and shift-based hash functions [7] have been used to generate keys of 32 bits.

3.3 Partitioning

3.4 Sorting and Merging

3.4.1 SIMD Sort- and Merge-Networks

Sorting through SIMD registers can be achieved through sorting networks [4]. The sorting network compares elements in parallel in each step using SIMD min/max operations. A final transposition is needed, which requires additional SIMD shuffle instructions to complete the sorting. We can build sorting kernels for various input sizes[4] depending on the data type and register size.

Merging can also benefit from SIMD acceleration. There are two standard merging networks: bitonic merge networks and odd-even merge networks ([4], SIMD accelerated [10]). Both scale poorly for bigger input sizes, with odd-even networks requiring slightly fewer comparisons but instead involving data movement and element masking. We can use small SIMD-accelerated merging networks as a kernel, to sort bigger input sizes.

3.4.2 Merging Higher Levels

We can merge different subparts of the data in different threads as long as we have enough sorted sublists. In the later round of the merge tree, with only a few sorted sublists remaining, it becomes increasingly more challenging to parallelize efficiently.

However, even at this point, we can parallelize. One way parallelization is made possible is through the Merge Path [15]. This conceptual path allows us to parallelize a two-way merge by splitting it into non-overlapping segments that form disjoint

[4]`https://bertdobbelaere.github.io/sorting_networks.html`

sets of elements. We can then sequentially merge these segments in parallel. The sequential merging can again benefit from SIMD acceleration [17].

In the later stages, out-of-cache merging can become necessary, quickly resulting in the memory bandwidth becoming the bottleneck of even a single-threaded merge routine. Therefore, multi-way merging [2] is introduced. The merge-tree consists of multiple two-way merge units (managed as tasks) connected via FIFO queues , is introduced. Only the leaves of the merge tree load data from memory. Blocking and task switching ensure the combined FIFO queues fit into the CPU cache. This way, memory bandwidth can be reduced with a slight CPU overhead. Optionally, we could explore merging through other primitives, such as tournament trees and priority queues.

3.5 String Types

As strings are variable in size, it makes sense to consider their internal representation and encoding. Certain string representations allow for cheap access to a prefix or short string. For instance, Umbra's string representation [13] consists of a 128-bit struct and is adopted by more recent databases like CedarDB and DuckDB. The first 32 bits represent the length. The remaining bits hold the complete string if the length is at most 12. Otherwise, the struct consists of the 32-bit length, a 32-bit prefix, and a pointer to a storage location. Due to saving pointer dereferences, this can speed up comparison, lexicographical sorting, and other prefix operations.

3.6 Joining

After sorting both input relations, a final loop over both sorted input relations suffices to find all join candidates. The sorted data is of the form (key, rid). Hence, we can use the row ID (rid) to find the respective tuples. As compression can result in false positives in the merging step, we might rEquijoinequire additional validation and filtering. Further parallelization of this final merge step is also possible.

4 Related Work

Several papers describe how SIMD-accelerated sorting can be done efficiently on mod- ern multi-core architectures. Chhugani et al. [8] describe the concepts needed for efficient SIMD sorting for both single- and multi-core execution, including sorting networks, bitonic- and odd-even merge networks, and how to deal with memory bandwidth limitations for large problem sizes through multiway merging. There are other ideas like MergePath [15] for merging a few very large sublists in parallel and SIMD accelerated. Kim et al. [11] implemented a sort-merge join using SSE intrinsics using these same concepts, projecting performance for wider SIMD widths that would outperform hash joins. Albutiu et al. [1] present MPSM, a sort-merge join implementation using their custom sorting routine without SIMD, concluding that their sort-merge join implementation is faster than the respective hash join implementation of Blanas et al. [5]. Balkesen et al. [2] experimentally studied the performance of sort-merge and radix-hash join. They claim to provide the fastest in-memory join processing algorithms using sorting and hashing, and that sort-merge join gets more comparable in performance to radix-hash join with very large input sizes. Still, they conclude that the radix-hash join exceeds the sort-merge join for 256-bit SIMD. The same authors claim that their parallel radix-hash join is the most efficient hash-join implementation yet [3]. The hash join operator implemented in Hyrise is also based on [3] and [2].

None of the papers mentioned above take advantage of 512-bit SIMD. There is research on SIMD sorting using AVX-512 [17, 18]. To the best of our knowledge, no literature exists on implementing sort-merge join with the same optimizations and concepts like multiway merging for AVX-512.

5 Project Plan

Time	Writing/Research	Prototype
Oct - Nov	– Setup for different achitectures – SIMD sorting building blocks – Single-threaded SIMD sorting – Adapt to all architectures	– Setup for x86, IBM Power, AWS Graviton. – Implement sorting- and bitonic-merge networks on different architectures. – Implement single-threaded SIMD sorting (for 64-bit keys and starting with 256-bit support)
Nov - Jan	– AVX-512 specific sorting – Multiway-Merging – Hyrise integration – Simd sorting of string types	– Scale up sorting- and bitonic-merge networks to wider bit width. – Explore further possible improvements through AVX-512 specific instructions. – Implement Multiway Merging (Explore alternatives: tournament-trees, priority-queue). – Working sort-merge-join implementation for integer/float types on different architectures. – Implement prefix- and hash-based key compressions (real string data from Public BI benchmark).
Jan-Feb	– Benchmarking & Evaluation	– Parameter tuning for different architectures. – Compare to hash-join and old sort-merge join (Hyrise). – Run TCP-H, TCP-DS and Pulic BI benchmark.
Feb-Mar	– Thesis Writing & Evaluation	– Draft of master thesis
Mar-Apr	– Thesis Writing & Evaluation	– Finished master thesis

Table 1: Planned Time Table

References

[1] Martina-Cezara Albutiu, Alfons Kemper, and Thomas Neumann. Massively parallel sort-merge joins in main memory multi-core database systems. *Proc. VLDB Endow.*, 5(10):1064–1075, 2012.

[2] Cagri Balkesen, Gustavo Alonso, Jens Teubner, and M. Tamer Özsu. Multi-core, main-memory joins: Sort vs. hash revisited. *Proc. VLDB Endow.*, 7(1):85–96, 2013.

[3] Cagri Balkesen, Jens Teubner, Gustavo Alonso, and M. Tamer Özsu. Main-memory hash joins on multi-core cpus: Tuning to the underlying hardware. In Christian S. Jensen, Christopher M. Jermaine, and Xiaofang Zhou, editors, *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, pages 362–373. IEEE Computer Society, 2013.

[4] Kenneth E. Batcher. Sorting networks and their applications. In *American Federation of Information Processing Societies: AFIPS Conference Proceedings: 1968 Spring Joint Computer Conference, Atlantic City, NJ, USA, 30 April - 2 May 1968*, volume 32 of *AFIPS Conference Proceedings*, pages 307–314. Thomson Book Company, Washington D.C., 1968.

[5] Spyros Blanas, Yinan Li, and Jignesh M. Patel. Design and evaluation of main memory hash join algorithms for multi-core cpus. In Timos K. Sellis, Renée J. Miller, Anastasios Kementsietsidis, and Yannis Velegrakis, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*, pages 37–48. ACM, 2011.

[6] Spyros Blanas and Jignesh M. Patel. Memory footprint matters: efficient equi-join algorithms for main memory data processing. In Guy M. Lohman, editor, *ACM Symposium on Cloud Computing, SOCC '13, Santa Clara, CA, USA, October 1-3, 2013*, pages 19:1–19:16. ACM, 2013.

[7] Shimin Chen, Anastassia Ailamaki, Phillip B. Gibbons, and Todd C. Mowry. Improving hash join performance through prefetching. In Z. Meral Özsoyoglu and Stanley B. Zdonik, editors, *Proceedings of the 20th International Conference on Data Engineering, ICDE 2004, 30 March - 2 April 2004, Boston, MA, USA*, pages 116–127. IEEE Computer Society, 2004.

[8] Jatin Chhugani, Anthony D. Nguyen, Victor W. Lee, William Macy, Mostafa Hagog, Yen-Kuang Chen, Akram Baransi, Sanjeev Kumar, and Pradeep Dubey. Efficient implementation of sorting on multi-core SIMD CPU architecture. *Proc. VLDB Endow.*, 1(2):1313–1324, 2008.

[9] Markus Dreseler, Jan Kossmann, Martin Boissier, Stefan Klauck, Matthias Uflacker, and Hasso Plattner. Hyrise re-engineered: An extensible database system for research in relational in-memory data management. In Melanie Herschel, Helena Galhardas, Berthold Reinwald, Irini Fundulaki, Carsten Binnig, and Zoi Kaoudi, editors, *Advances in Database Technology - 22nd International Conference on Extending Database Technology, EDBT 2019, Lisbon, Portugal, March 26-29, 2019*, pages 313–324. OpenProceedings.org, 2019.

[10] Hiroshi Inoue, Takao Moriyama, Hideaki Komatsu, and Toshio Nakatani. Aa-sort: A new parallel sorting algorithm for multi-core SIMD processors. In *16th International Conference on Parallel Architectures and Compilation Techniques (PACT*

2007), Brasov, Romania, September 15-19, 2007, pages 189–198. IEEE Computer Society, 2007.

[11] Changkyu Kim, Eric Sedlar, Jatin Chhugani, Tim Kaldewey, Anthony D. Nguyen, Andrea Di Blas, Victor W. Lee, Nadathur Satish, and Pradeep Dubey. Sort vs. hash revisited: Fast join implementation on modern multi-core cpus. *Proc. VLDB Endow.*, 2(2):1378–1389, 2009.

[12] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter A. Boncz, Alfons Kemper, and Thomas Neumann. How good are query optimizers, really? *Proc. VLDB Endow.*, 9(3):204–215, 2015.

[13] Thomas Neumann and Michael J. Freitag. Umbra: A disk-based system with inmemory performance. In *10th Conference on Innovative Data Systems Research, CIDR 2020, Amsterdam, The Netherlands, January 12-15, 2020, Online Proceedings*. www.cidrdb.org, 2020.

[14] Chris Nyberg, Tom Barclay, Zarka Cvetanovic, Jim Gray, and David B. Lomet. Alphasort: A RISC machine sort. In Richard T. Snodgrass and Marianne Winslett, editors, *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, USA, May 24-27, 1994*, pages 233–242. ACM Press, 1994.

[15] Saher Odeh, Oded Green, Zahi Mwassi, Oz Shmueli, and Yitzhak Birk. Merge path - parallel merging made simple. In *26th IEEE International Parallel and Distributed Processing Symposium Workshops & PhD Forum, IPDPS 2012, Shanghai, China, May 21-25, 2012*, pages 1611–1618. IEEE Computer Society, 2012.

[16] Adrian Vogelsgesang, Michael Haubenschild, Jan Finis, Alfons Kemper, Viktor Leis, Tobias Mühlbauer, Thomas Neumann, and Manuel Then. Get real: How benchmarks fail to represent the real world. In Alexander Böhm and Tilmann Rabl, editors, *Proceedings of the 7th International Workshop on Testing Database Systems, DBTest@SIGMOD 2018, Houston, TX, USA, June 15, 2018*, pages 1:1–1:6. ACM, 2018.

[17] Alex Watkins and Oded Green. A fast and simple approach to merge and merge sort using wide vector instructions. In *8th IEEE/ACM Workshop on Irregular Applications: Architectures and Algorithms, IA3@SC 2018, Dallas, TX, USA, November 12, 2018*, pages 37–44. IEEE, 2018.

[18] Zekun Yin, Tianyu Zhang, André Müller, Hui Liu, Yanjie Wei, Bertil Schmidt, and Weiguo Liu. Efficient parallel sort on avx-512-based multi-core and many-core architectures. In Zheng Xiao, Laurence T. Yang, Pavan Balaji, Tao Li, Keqin Li, and Albert Y. Zomaya, editors, *21st IEEE International Conference on High Performance Computing and Communications; 17th IEEE International Conference on Smart City; 5th IEEE International Conference on Data Science and Systems, HPCC/SmartCity/DSS 2019, Zhangjiajie, China, August 10-12, 2019*, pages 168–176. IEEE, 2019.