Comprehensive Framework for Sorting Benchmarks

Sergey Madaminov
Department of Computer Science
Stony Brook University
New Computer Science Building
Stony Brook, New York 11794-2424
smadaminov@cs.stonybrook.edu
supervised by Michael Ferdman

ABSTRACT

In the early days, sorting accounted for almost 25% of all cycles that computers were spending. That led to the development of a variety of sorting algorithms and their implementations, as well as the creation of sorting benchmarks. However, those benchmarks do not account well for increasing variability in the nature of data and they also fail to assess architectural features of different computer systems depending on the choice of the sorting algorithm. This work proposes the development of a comprehensive sorting benchmark framework to address those issues and to help with the evaluation of sorting algorithms from both software and hardware perspectives.

1. INTRODUCTION

Sorting is an important operation that computers have been performing from the early days [18]. This led to the development of various sorting algorithms. As it has proved to be important at datacenter scale [11, 12] and it targeted different scenarios and systems, various algorithms were developed for general purpose sorting by using CPUs [22, 15, 5], for sorting that is suitable for highly parallel systems [2], and for sorting using other types of architectures [10, 19]. However, with the rapid pace of increase in the scale of a sorting problem, the question of which algorithm to choose remains persistent. To answer this question, one needs to have a sorting benchmark that is capable of providing enough information for analyzing the needs and efficiency of available and proposed algorithms for a given purpose.

The idea of having benchmarks is not novel and there is a body of work done on the benchmarks for system components such as CPU [6], applications such as databases [25], and systems for processing cloud workloads [8]. Some existing studies have targeted sorting specifically [13, 7, 21, 14]. Generally stated, the different types of benchmarks cover different parts of sorting systems from both architectural perspectives as well as algorithmic and software implementations.

Proceedings of the VLDB 2019 PhD Workshop, August 26th, 2019.
Los Angeles, California.

In spite of the rich body of knowledge on benchmarks, drastic changes in computing today have made some of the benchmarks obsolete. For instance, benchmarks such as *PennySort* and *TeraByte Sort* are deprecated due to the substantial growth in computational power that allows handling much larger data sets [13]. Similarly, the nature of the data itself may also differ and while there is a suggested structure of a record to sort [7] that defines 100-byte records, not all studies follow it [20, 23]. Moreover, sorting task itself can vary a lot: it can be local to a single computer machine or distributed among many nodes in a cluster, or it can target different architectures.

The variety of different factors makes it unnecessarily complicated to evaluate sorting algorithms and sorting systems and compare them against each other. Without a defined structure of data record or defined distribution, it may become non-trivial how to compare different sorting algorithms or their implementations directly. It becomes even more complicated when targeted systems are FPGAs as they may be programmed to process a very specific set of data and changes in the structure of the data may either significantly affect results or make it unfeasible to even process that data.

To effectively analyze the choice of a sorting algorithm or sorting system, one needs to collect both hardware and software statistics of any viable approach. While hardware statistics may include cache performance, branch misprediction, and TLB misses, the software statistics may include running time on a particular system and scalability of the sorting algorithm with the increasing number of available parallelism or growth in the volume of data.

To overcome the above issues, this work proposes developing a comprehensive framework for sorting benchmarks capable of evaluating various hardware and software aspects of sorting algorithms and sorting systems while maintaining ease of use. This work is structured as follows: Section 2 justifies the development of such a framework, Section 3 discusses framework architecture, and Section 4 concludes.

2. THE NEED FOR COMPREHENSIVE FRAMEWORK

Multiple studies have been done on sorting benchmarks [3, 24, 9]. However, we advocate that there is a need for more research on that topic. There are three main reasons for that: first, existing benchmarks do not consider the variety of input data and its distribution, second, they do not assess hardware statistics, and, third, they are not a good fit for a variety of different computer architectures. The

last one is particularly important as there is a number of studies targeting various architectures such as GPUs [10], FPGAs [19], and AVX-based [4]. But without a systematic approach, the task of comparing them against each other becomes quite challenging. This task of comparing different architectures between themselves especially complicated when only part of the sorting algorithm is implemented. For example, some studies targeting FPGAs focus on the merging [20, 23]. As such implementations may require data transfer to and from the sorting system, some level of data preparation, or may depend on the problem size, it is unclear how to compare results obtained from different architectures. Thus, the proposed framework should provide a facility to perform a comparison between them. For similar sorting algorithms, it can be achieved by direct comparison of similar phases of the algorithms and estimating the remaining phases, which may include potentially required communication such as data transfer over the PCIe or another medium.

Many studies related to sorting use record structure suggested by Datamation sorting benchmark [7], but it is not universally accepted. Due to variations in record structure, comparing the results of different studies directly is not straightforward. On the other hand, the Datamation sorting benchmark that defines the structure of a data record being 100-byte with ten-byte key and ninety-byte value could have become outdated. The current database vendors and users should be surveyed to collect prevailing structures of records and data distributions. However, as some works may use the different input data, it is important to allow variations in the input data. First, it will allow analyzing studies that use different input data. Second, it will enable the comparison with prior work.

It deems important to understand how sorting algorithms scale with an increasing number of parallelism or volume of data, which requires collecting corresponding information. To perform a more thorough evaluation of the sorting algorithm it is crucial to collect systems statistics such as memory bandwidth and caches miss rate. While it is possible to use existing tools for profiling, it requires the algorithm developer to install and learn a variety of tools. It can be avoided by adding such functionality into the framework itself. Some of the algorithms exhibit different behavior on systems level, e.g., *Quicksort* algorithm is known for good cache behavior and utilization. Gathering more information can help to get a clear picture of the sorting algorithm, which in turn can help to reason about the differences between different sorting algorithms. We suggest that the framework should not just provide statistical data as feedback, but also provide an analysis report that identifies weak points of the algorithm and what potentially can be improved. Moreover, modern benchmark systems are not easy to use. Thus, the proposed framework should be user-friendly and should provide reports for further analysis in a readable format.

With a variety of studies on sorting including recent works on exploring new computer architectures such as FPGAs [19, 20, 23] and GPUs [10] and their suitability for sorting, comparing their result becomes a challenging task. The proposed framework will strive to address these challenges and needs while maintaining ease of use. It may be still unclear how to compare different computer architectures but this work sets resolving this problem as one of its targets.

Table 1: Example of the data distribution.

Uniform	Bernoulli
Poisson	Exponential
Gaussian	Log Normal
Gamma	Beta

3. FRAMEWORK ARCHITECTURE

This section provides a brief overview of potential framework components and argues for their need. It discusses various aspects of proposed framework such as data distribution, collectible system statistics, and some of the other aspects that include record structure.

3.1 Data Distribution

The record generator used in the Sort Benchmark [13] can produce two types of data distribution. Despite a bigger variety of data being considered by Helman et al. [14], their work focuses on the structure of the data rather than its distribution. Based on the nature of the data, it is possible to have more options in the data distribution and the proposed framework should account for both data structure and data distribution. Often, these two features may be independent of each other so the framework should provide facilities to combine them together. Thus, it may become possible to have both staggered data structure with Gaussian distribution or any other combination of data structure and data distribution. Table 1 provides the list of some of the possible distributions to account for. However, similar to the Datamation [7], a comprehensive list should be compiled using input from the database vendors and the database users to represent the actual workloads that may be found outside of research groups.

3.2 System Statistics

Currently, to assess systems performance such as memory bandwidth, the developer has to use tools such as Intel VTune [17]. While in some cases it may be inevitable to use external software, the framework should collect statistics where it can and at least provide the list of various metrics to account for. Table 2 provides the list of some of the suggested systems statistics to collect.

Many modern sorting algorithms have optimal or near-optimal complexity, but real implementations may result in noticeable differences between them. Collecting such statistics may help to identify bottlenecks that may lead to further research on how those bottlenecks can be mitigated. As a naïve example, *hugepages* may help to reduce TLB misses [16] and using recently introduced *high-bandwidth memories* may help to handle the memory bandwidth bound parts of the sorting algorithms. Moreover, identifying such bottlenecks may steer hardware research. One can imagine building a sorting specific accelerator to overcome them. For example, it may be an FPGA that accelerates a particular task or even a special-purpose processor that has an ISA targeting the sorting task.

3.3 Miscellaneous

The data distribution and systems statistics cover many different aspects of sorting but there are still some implementation details and guidelines that may become useful for

Table 2: Example of the collectible system statistics.

I/O Intensity	TLB Miss Rate
IPC Intensity	Caches Miss Rate
Cache Utilization	Branch Misprediction
Memory Bandwidth	Memory Peak B/W

algorithm developers. They include using custom comparators, avoiding using indirect function calls [1], and different record types with latter being tightly coupled with custom comparators. Ultimately, evaluation of sorting algorithms and sorting systems may have more factors to consider that we have previously defined and it is deemed important to identify them and leave the framework open to including them.

4. CONCLUSIONS

This work advocates for the development of a comprehensive framework for sorting benchmarks, which accounts for various aspects of sorting algorithms starting with defining the input data and measures both their software and hardware statistics. Such a framework may help to create a system to foster the development of sorting algorithms as well as designing new computer architectures for sorting. We envision that it will be beneficial for many communities outside of a group of scientists who work on the development of new sorting algorithms or modifying the existing ones. While the work in its preliminary stage, there are many design choices that have to be done and collecting feedback from database vendors and users is essential for what are the common data features and hardware statistics they do care.

5. REFERENCES

[1] V. Agrawal, A. Dabral, T. Palit, Y. Shen, and M. Ferdman. Architectural Support for Dynamic Linking. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 691–702, New York, NY, USA, 2015. ACM.

[2] M. Axtmann, T. Bingmann, P. Sanders, and C. Schulz. Practical Massively Parallel Sorting. In *Proceedings of the 27th ACM on symposium on Parallelism in Algorithms and Architectures*, pages 13–23. ACM Press, June 2015.

[3] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The NAS Parallel Benchmarks - Summary and Preliminary Results. In *Proceedings of the 1991 ACM/IEEE Conference on Supercomputing*, pages 158–165, New York, NY, USA, 1991. ACM.

[4] B. Bramas. A Novel Hybrid Quicksort Algorithm Vectorized using AVX-512 on Intel Skylake. *International Journal of Advanced Computer Science and Applications*, 8(10), 2017.

[5] C. Bron. Merge Sort Algorithm [M1]. *Communications of the ACM*, 15(5):357–358, May 1972.

[6] J. Bucek, K.-D. Lange, and J. v. Kistowski. SPEC CPU2017: Next-Generation Compute Benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, pages 41–42, New York, NY, USA, 2018. ACM.

[7] A. et al, D. Bitton, M. Brown, R. Catell, S. Ceri, T. Chou, D. DeWitt, D. Gawlick, H. Garcia-Molina, B. Good, J. Gray, P. Homan, B. Jolls, T. Lukes, E. Lazowska, J. Nauman, M. Pong, A. Spector, K. Trieber, H. Sammer, O. Serlin, M. Stonebraker, A. Reuter, and P. Weinberger. A Measure of Transaction Processing Power. *Datamation*, 31(7):112–118, April 1985.

[8] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi. Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware. *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 37–48, 2012.

[9] P. Garcia and H. Korth. Multithreaded Architectures and the Sort Benchmark. In *Proceedings of the 1st International Workshop on Data Management on New Hardware*, New York, NY, USA, 2005. ACM.

[10] N. Govindaraju, J. Gray, R. Kumar, and D. Manocha. GPUTeraSort: High Performance Graphics Co-processor Sorting for Large Database Management. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, pages 325–336, New York, NY, USA, 2006. ACM.

[11] G. Graefe. Sorting And Indexing With Partitioned B-Trees. In *In Proceedings of the 1st International Conference on Innovative Data Systems Research*, Asilomar, CA, USA, January 2003.

[12] G. Graefe. Implementing Sorting in Database Systems. *ACM Computing Surveys*, 38(3), September 2006.

[13] J. Gray, C. Nyberg, M. Shah, and N. Govindaraju. Sorting Benchmark. http://sortbenchmark.org/.

[14] D. R. Helman, D. A. Bader, and J. JáJá. Parallel Algorithms for Personalized Communication and Sorting With an Experimental Study (Extended Abstract). In *Proceedings of the eighth annual ACM symposium on Parallel Algorithms and Architectures*, pages 211–222. ACM Press, June 1996.

[15] C. A. R. Hoare. Quicksort. *The Computer Journal*, 5(1):10–16, January 1962.

[16] J. Hu, X. Bai, S. Sha, Y. Luo, X. Wang, and Z. Wang. HUB: Hugepage Ballooning in Kernel-based Virtual Machines. In *Proceedings of the International Symposium on Memory Systems*, pages 31–37, New York, NY, USA, 2018. ACM.

[17] Intel. Intel VTune. https://software.intel.com/en-us/vtune.

[18] D. E. Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison-Wesley Professional, 2nd edition, 1998.

[19] D. Koch and J. Torresen. FPGASort. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 45–54. ACM Press, 2011.

[20] S. Mashimo, T. V. Chu, and K. Kise. High-Performance Hardware Merge Sorter. In *2017 IEEE 25th Annual International Symposium on*

Field-Programmable Custom Computing Machines (FCCM), pages 1–8. IEEE, April 2017.

[21] C. Nyberg, T. Barclay, Z. Cvetanovic, J. Gray, and D. Lomet. AlphaSort: A Cache-sensitive Parallel External Sort. *The VLDB Journal*, 4(4):603–628, October 1995.

[22] T. Peters. Timsort. https://bugs.python.org/file4451/timsort.txt.

[23] M. Saitoh, E. A. Elsayed, T. V. Chu, S. Mashimo, and K. Kise. A High-Performance and Cost-Effective Hardware Merge Sorter without Feedback Datapath. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 197–204. IEEE, April 2018.

[24] K. Thearling and S. Smith. An Improved Supercomputer Sorting Benchmark. In *Proceedings of the 1992 ACM/IEEE Conference on Supercomputing*, pages 14–19, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[25] TPC. Active TPC Benchmarks. http://www.tpc.org/information/benchmarks.asp.