Parallel Sorting in C++11

Klára Schovánková, Supervisor: Daniel Langr
Faculty of Information Technology, Czech Technical University in Prague

FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Motivation

Sorting is a time-consuming part of many algorithms. It has to be parallelized to make full use of the computing power of the hardware. Parallel sorting is part of the standard of C++17, but it isn't implemented yet in many significant compilers, such as GCC or Clang.
Existence of the parallel sorting algorithm in implementations of standard libraries of these compilers would allow developers to use better the multi-core hardware without the need for the deeper knowledge of the parallelization. So far when programmers want to use parallel sorting algorithms in C++ they must use non-standard extensions of C++ language or external libraries.

Contributions

We have created a new **efficient parallel variant** of the Quicksort algorithm called C++11Sort. It is based on the C++11 threads and respective synchronization techniques, i.e. without the usage of the non-standard C++ extensions nor external libraries. The algorithm is **in-place** and efficiently **balances the load** between the physical threads. The essential part of the algorithm is also **efficient partitioning** that can also be used in other applications.
The algorithm was also implemented in the version that, unlike the majority of the sorting algorithms, allows **sorting multiple arrays at once**, which is useful e.g. when working with sparse matrices and sorting it's non-zero elements.

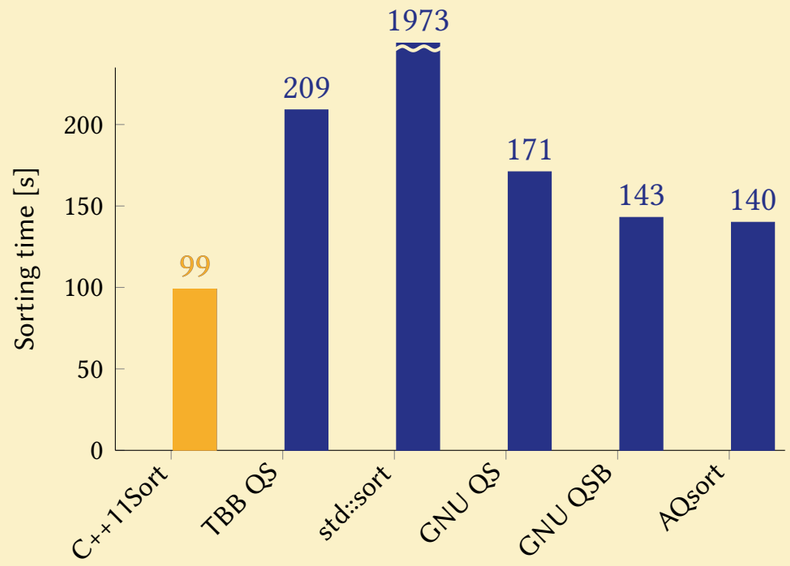

Figure: Comparing of the sorting time of two billion integers by different implementations.

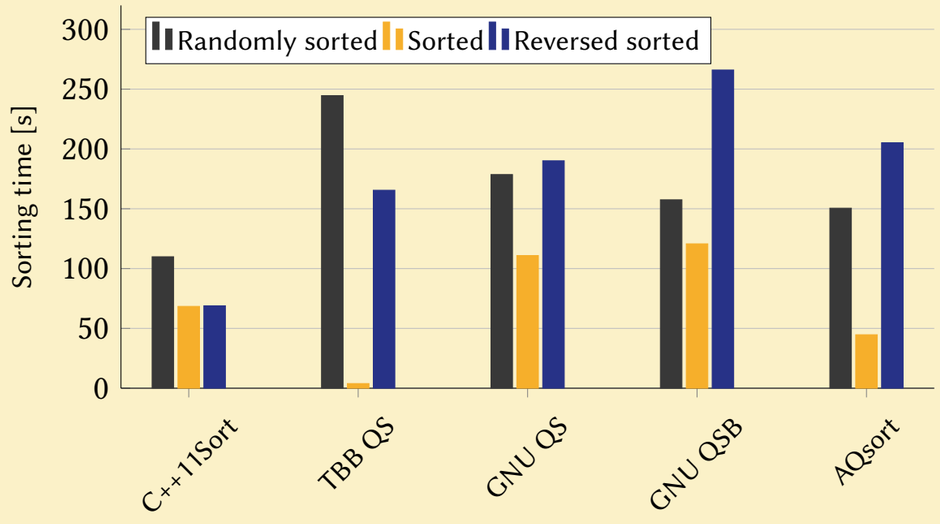

Figure: Effect of the initial sorting of the data on the sorting time. Two billion doubles, comparing different implementations.

Experiments and Results

The created algorithm was tested on many different types of data and their different initial sorting. The results were also compared with other existing implementations of the parallel in-place sorting algorithms. The experiments were measured on the cluster STAR with 20 cores.
The created sorting algorithm achieved excellent results. It **outperformed** known in-place parallel sorting algorithms on the average case. For example, when sorting two billion integers, it was **faster by 28% than any other known implementation**.
C++11Sort also achieved better results when sorting multielements of the sparse matrices than the currently used algorithm, so it could save computing hours on supercomputers.

Conclusions

Created efficient parallel sorting algorithm C++11Sort allows developers to **parallelize sortings facilely** without the usage of the non-standard extensions or external libraries. It achieves excellent results when sorting various types of data. The experiments showed that the created algorithm is **general, scalable, and both space and time efficient.**