Parallel Quicksort Algorithm

Ing. Gabriel Hévr, Supervisor: doc. Ing. Ivan Šimeček, Ph.D.

FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Motivation

Sorting algorithms are essential to computer science, underpinning systems like databases, search engines, file systems, and networking. Efficient sorting boosts performance and scalability, impacting everything from query optimization to database joins. Speeding up sorting improves application performance, reduces CPU usage, and lowers energy consumption, benefiting both efficiency and the environment.

The modern approach is to parallelize the sorting routines. This technique maximizes the utilization of multi-core CPUs, resulting in significant performance improvements compared to sequential methods.

Quicksort

Quicksort is a widely used sorting algorithm known for its superior average-case performance. It is commonly the default choice in many languages and libraries. Its proven efficiency makes it an ideal candidate for optimization, which is why we focused on enhancing its performance. New optimizations for quicksort are still being developed, mostly for the sequential version. We analyze these innovations and identify the ones that are best suited for efficient parallelization.

Existing Solutions

Many popular libraries offer parallel quicksort implementations, but they often lack the latest optimizations, missing out on the full speed potential of modern systems. Here are some current state-of-the-art implementations:

- **GCC BQS**: Parallel balanced quicksort from libstdc++
- **GCC QS**: Unbalanced quicksort variant
- **oneTBB**: Parallel quicksort from Intel oneTBB library
- **cpp11sort**: Parallel quicksort based on C++11 threading
- **AQsort**: Parallel multi-array in-place sort with OpenMP

Our Contribution: PPQSort

We have developed PPQSort, a fast parallel quicksort algorithm that combines and parallelizes novel sequential optimizations. Key features include:

- **Minimizing Branches:** Reduces branching to enhance performance on modern CPUs with large pipelines.
- **Detecting Distributions:** Adapts to various data patterns, optimizing for simple cases and adjusting for complex ones.
- **Additional Optimizations:** Includes recursion elimination, insertion sort for small partitions, memory order tuning, AlphaDev AI code for sorting three elements, and dynamic parallel-sequential switching.

Publications

[1] HÉVR Gabriel; ŠIMEČEK Ivan. PPQSort: Pattern Parallel Quicksort. In PPAM conference. 2024. **Best student paper**.

Thread Pool

PPQSort features a custom thread pool, developed due to C++ lacking a built-in option. It uses multiple queues, one per thread, where threads "try" to lock their preferred queue and, if busy, quickly move to the next, minimizing blocking and idle times. The same approach applies to task stealing. Additionally, a sleep mechanism prevents resource waste when no tasks are available.

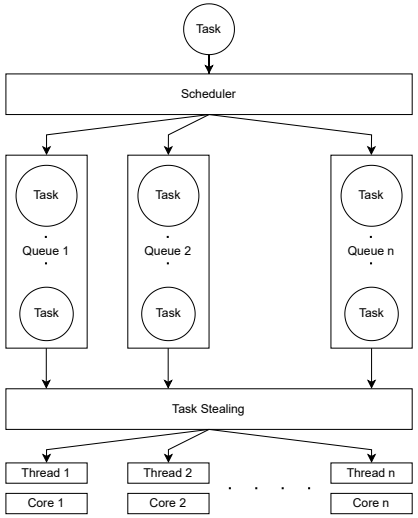


Figure 1. Thread Pool diagram.

Implementation

- **C++20:** Complies with the C++20 standard.
- **No Dependencies:** Runs independently without extra libraries.
- **Header-Only:** Easy to integrate as a header-only implementation.
- **Execution Policies:** Supports custom execution policies.
- **Two Variants:** Includes one with OpenMP and one in pure C++.
- **Public Resources:** Comprehensive benchmarks, tests, and implementation are on GitHub.

Benchmark Results

We benchmarked PPQSort on four clusters, and it consistently outperformed all current state-of-the-art parallel quicksort implementations.

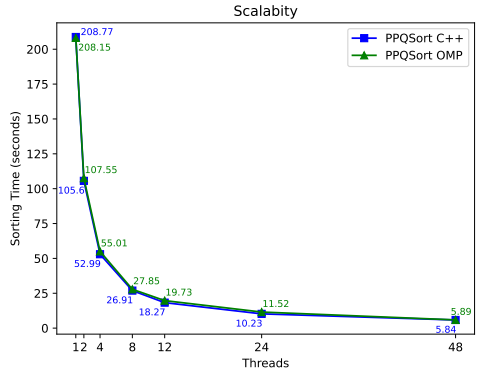
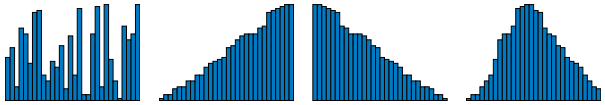


Figure 2. Results for 2×10^9 random integers.



(a) Random data (b) Ascending data (c) Descending data (d) Organ pipe data

Figure 3. Visualization of specific input data distributions.

Algorithm	Random	Ascending	Descending	Rotated	OrganPipe	Heap	Total	Rank
PPQSort C++	**5.84s**	1.84s	4.55s	**1.38s**	**2.96s**	**5.58s**	**22.15s**	1
GCC BQS	13.72s	4.18s	19.11s	49.89s	8.24s	13.78s	108.92s	4
GCC QS	18.33s	4.1s	14.62s	12.51s	14.01s	19.16s	82.73s	3
oneTBB	43.66s	**0.09s**	8.62s	13.84s	8.12s	43.9s	118.23s	5
cpp11sort	9.58s	2.47s	**2.66s**	5.47s	3.42s	9.9s	33.5s	2
AQsort	24.72s	3.66s	23.14s	21.83s	22.6s	25.31s	121.26s	6

Table 1. Results for 2×10^9 random integers.

Conclusion

PPQSort is the fastest parallel quicksort implementation to date and represents a new state-of-the-art solution.

hevrgabr@fit.cvut.cz