Comparing Parallel Sorting Algorithms

Aleksander Nikolajev[1], **Cathy Toomast**[2] and **Silver Maala**[1]

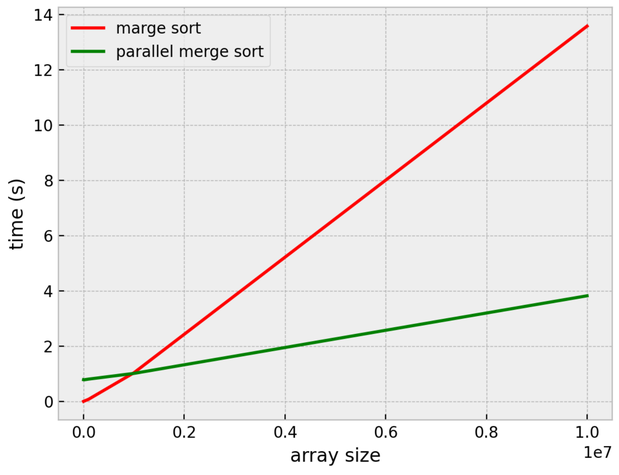
[1]Institute of Computer Science, University of Tartu
[2]Institute of Technology, University of Tartu

Project Description

In our project we implemented six different algorithms: merge sort, parallel merge sort, quick sort, hyper-quicksort, bitonic sort and parallel bitonic sort. We did that with the aim to compare parallel algorithms with their non-parallel versions. We have put a restriction that made the project more challenging - only using Python implementations. This has created some interesting blockades and improvised techniques. All of our implementations can be found in github [2]

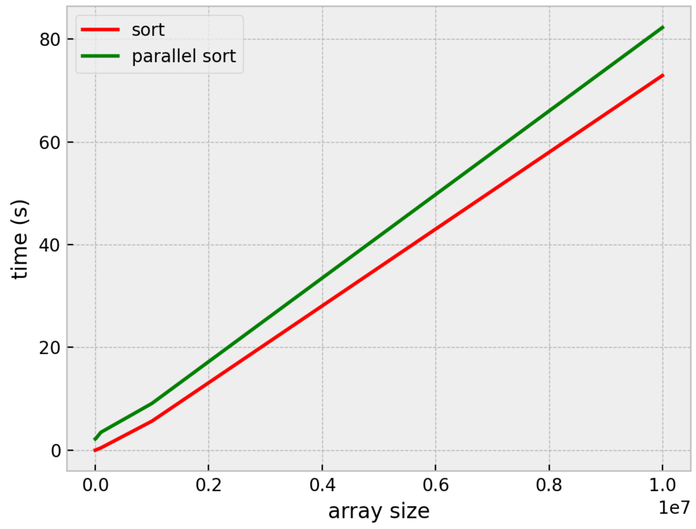
Merge Sort and Parallel Merge Sort

Firstly, we implemented merge sort, which is a divide and conquer algorithm. It divides the input array into two halves, calls itself for the two halves, and then merges the two sorted halves. [1]
After that, we did the parallel merge sort. It was implemented by splitting data into partitions and performing a regular merge sort across each partition. So it will work better if it is possible to use more than two processes.
Then we compared merge sort with parallel version, which results can be seen on the figure. We tested with 5 different array sizes. For smaller array sizes, the parallel implementation is slower than naive implementation due to the overhead. However, parallel starts to beat naive quite fast.



Quicksort and Hyper-Quicksort

Hyper-quicksort is an implementation of quicksort [5] on a hypercube [4]. Each node of the hypercube is a thread.
Hyper-quicksort works by splitting the array between each node of the hypercube and the broadcasting a pivot value. Each node will split its array with the pivot to lower values and higher values. After that the node will send its lower values to its lower partner node while getting back the lower partner node's higher values.
After this the lower nodes will hold values lower than the pivot and the higher node values hold values higher than the pivot.
Next the 2 groups (higher nodes and lower nodes) split and each group will run from the start. When no more group splits are possible, then each node will sort their arrays with the regular quicksort and return the sorted array. After joining the sorted array from each node, the array is then sorted.



Bitonic Sort

Bitonic Search is a comparison-based algorithm that works best when run in parallel. First, it converts random numbers of sequence into a *bitonic sequence*. A bitonic sequence is where the elements first come in increasing order, then start decreasing after some particular index. An array A[0...i...n-1] is called Bitonic if there exists an index i such that:

$$A[0] < A[1] < ... < A[i-1] < A[i] > A[i+1] > A[i+2] > ... > A[n-1] \quad (1)$$

,where 0 <= i <= n-1 [3].

The principle to sort with Bitonic Search is the following:

1. Form the Bitonic Sequence from an input of random integers. This gives us a sequence where both halves are sorted. One is sorted in the ascending order and the other is in descending.
2. Compare the first element of the first half with the first element of the second half, then the second one from the first half and the second from the second half and so on. If the element in the second half is smaller, swap it.
3. After the second step, all the elements in the first half are smaller than the second half. We will repeat the process described in step 2 onto every sequence until we get a sorted sequence.

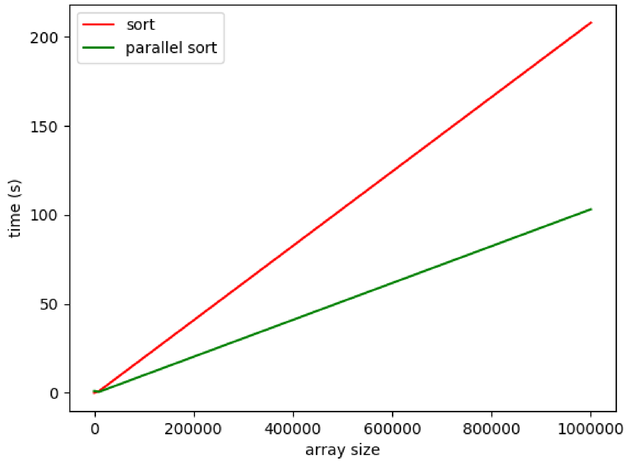


Fig. 1: Bitonic Sort vs. Parallel Bitonic Sort Comparison

The resulting graph shows the significant difference in computation time. At the very start, you can see how the normal implementation is faster due to parallel processor overhead.

References

[1] GeeksForGeeks. "Merge Sort". In: *https://www.geeksforgeeks.org/merge-sort/* (Jan. 2021).

[2] Github. "Parallel". In: *https://github.com/ctoomast/algo_project* (Jan. 2021).

[3] Javatpoint. "Bitonic Sort". In: *https://www.javatpoint.com/bitonic-sort* (Jan. 2021).

[4] Prashant Srivastava. "Hyper Quick Sort(Parallel Quick Sort)". In: *https://cse.buffalo.edu/faculty/miller/Courses/CSE633/Srivastava-Spring-2014-CSE633.pdf* (Jan. 2021).

[5] Wikipedia. "Quicksort". In: *https://en.wikipedia.org/wiki/Quicksort* (Jan. 2021).