Parallel Programming Assignment 2

Nils Bäckström, Jonathan Löfgren, Anton Sundin

February 24, 2015

1 Problem description

In this assignment, the task was to sort an array of double precision numbers using parallel sorting algorithms.

2 Implementation

Quicksort: Takes an array as input. It then picks a pivot element and places lower numbers left of the pivot and larger numbers to the right. Non-parallel use is to call quicksort two times (with array elements left of the pivot and elements right of the pivot) recursively until everything is sorted.

2.1 Divide and Conquer

This algorithm uses a parallel implementation of a standard quicksort algorithm. Every time the array is split in two around the pivot, the current thread starts up a new thread that calls quicksort with the right part of the array and then proceeds to call quicksort itself with the left part of the array. This is done until we have reached a certain recursion level l. Number of working threads n would then be $n = 2^l$.

2.2 Pear sort

In this algorithm, n threads are created as neighbors, over which the whole array of elements is divided. Each thread performs a serial quicksort on its own part of the array, after which threads with an even thread id alternates between merging its list with its right and left neighbor's lists. After n steps of this alternation, the whole array is sorted.

3 Results

The two algorithms were tested with different number of threads (determined by maximum depth in the divide and conquer algorithm). The speedup is measured as $S = T(1)/T(n)$ where n is number of threads used. The result is presented in figure 1. From this figure we can see that the speedup differs between the two algorithms, the divide and conquer algorithm keeps getting faster as more threads are added while the pear sort algorithm slows down after 8 threads have been reached.

4 Conclusion

So apparently the divide and conquer algorithm scales well with increasing number of threads. The overhead of creating new threads does not seem to overcome the speedup we receive from running several quicksorts simultaneously.

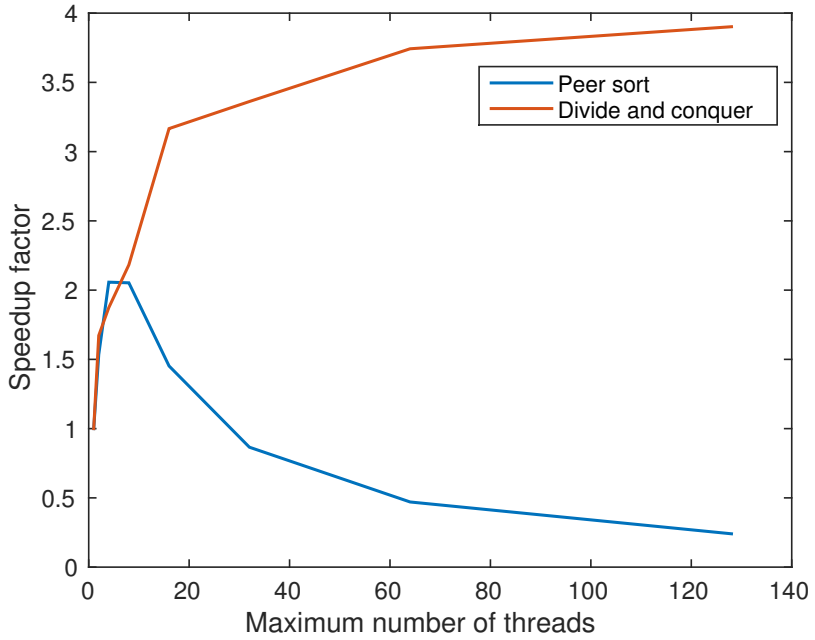

Figure 1: Speedup for the two different algorithms!

The Pear sort on the other hand seems to have a harder time managing many threads. We suspect that this is because the algorithm has to merge all sub-lists p times which mean that we have to perform a lot of memory acrobatics.

5 Appendix

All code is in qs_serial.c, qs_pthread.c and qs_peer.c. In the tar-file you also find the Makefile.

1. qs_serial.c
2. qs_pthread.c
3. qs_peer.c
4. Makefile