Parallel Programming

Exercise 3, Task Parallelism – Parallel Sorting

18.03.2021

1 Task Description

Implement the famous QuickSort (as in-place sorting algorithm) and MergeSort (not necessarily in-place) algorithms in a parallel manner. Both are typical divide-and-conquer algorithms. For program optimization in general it is crucial to document findings and draw conclusions given benchmarks. Thus, again submit your benchmarks results in a separate report file. Please include information about your test environment (most importantly CPU cores).

1.1 Sequential version of the algorithms

First, implement these algorithms without parallelism and experiment with the runtime on random data. Those implementations serve as baseline for the optimizations. Benchmarking is an essential part in developing parallel algorithms. Thus, make sure your tests are fair (e.g. test data is big enough, test data is not biased like typical worst cases for the algorithms).

1.2 Naive parallelization

Implement both algorithms with unlimited parallelism, i.e. spawn a new task on each recursion level. Compare your solutions to the sequential version and draw conclusions (just one or two sentences explaining your findings).

1.3 Recursion with threshholds

Finally, experiment with a treshold value that states at which remaining size of the problem size the sorting should not be split up into tasks but rather be done sequentially. Check for which values you get the optimum speed (for a suffciently large set of data).

2 Files to submit

Submit your solution as including the implementation and a report:

- `quicksort.{c,cpp,cs,fs,...}` Implementation of parallel quick sort with adjustable threshold value
- `mergesort.{c,cpp,cs,fs,...}` Implementation of parallel merge sort with adjustable threshold value
- `report.pdf` Documentation of your findings and speedup.