UNIVERSITÄT SIEGEN

Fakultät IV
Betriebssysteme und verteilte Systeme
Prof. Dr. rer. nat. Roland Wismüller

Excercise Sheet 2

(To be processed until 12.11.)

Lecture Parallel Processing
Winter Term 2024/25

Exercise 1: Parallelization of Quicksort using C++ Threads (Compulsory Exercise! Submit until Tuesday, November 12th, 10:00 via moodle)

In this exercise, the Quicksort algorithm is to be parallelized (see Section 2.7.4 of the lecture). The basic idea of Quicksort is as follows:

- From a given array, an element is selected as a pivot or a reference element, e.g., the element from the center.
- A larger element is sought from below and a smaller one from above.
- If such items are found, they are placed incorrectly and are swapped.
- This search and swap will continue until 'below' and 'above' meet. After that, there are only smaller (or equal) elements below, and only larger (or equal) elements above.
- These two areas must now be sorted in turn, which translates to two recursive calls to quicksort.

Have a look at the code in the archive `u02eFiles.zip`[1] on the lecture's web page. The procedure

```
void quicksort(int *a, int lo, int hi)
```

in `qsort.cpp` sorts the portion of the array starting at index value `lo` up to and including index value `hi` in ascending order. In addition, some auxiliary functions are implemented:

- `void initialize(int *a, int n)` initializes an array `a` of length `n` with random numbers.
- `int checkSorted(int *a, int n)` checks if an array `a` of length `n` is sorted.
- `void printArray(int *a, int n)` can be used as necessary to print a (small!) array.

a) Compile the program and run it. Test the code for different array lengths (recommended starting value: 10,000,000 elements). Record the time that the program requires for sorting the array in each case.

b) Rewrite the program that one of the two recursive invocations of `quicksort` will execute in a new thread, but only if the array length for this invocation is greater than a constant `MINSIZE`. Try different values for `MINSIZE` (recommended starting value: 1,000,000).

Compare the performance of your implementation with the previous measurements, using a table. Calculate the speedup and interpret your results.

For support, you can use the following tutorial on C++ threads:
https://solarianprogrammer.com/2011/12/16/cpp-11-thread-tutorial

For motivated students: The C library offers a function `qsort` (documentation: `man qsort`):

```
void qsort(void *base, size_t nmemb, size_t size,
           int(*compar)(const void *, const void *))
```

Compare the performance of your implementation against the function `qsort` in the C library.

[1] http://www.bs.informatik.uni-siegen.de/web/wismueller/vl/pv/u02eFiles.zip