1 Parallelization of an iterative Jacobi Solver

Go to the `jacobi` directory. Compile the `jacobi.c` code via ‘`make [debug|release]`’ and execute the resulting executable via ‘`OMP_NUM_THREADS=procs make run`’, where procs denotes the number of threads to be used.

Exercise 1 (optional - only if you are already familiar with the tool): Use the VTune Amplifier XE to find the compute-intensive program parts of the Jacobi solver. There should be three performance hotspots in the program (depending on the input dataset):

Number	Line Number	Function Name	Runtime Percentage
1			
2			
3			

Exercise 2: Parallelize the compute-intensive program parts with OpenMP. For a simple start, create one *parallel region* for each performance hotspot.

Exercise 3: Try to combine *parallel regions* that are in the same routine into one *parallel region*.

Exercise 4: If you are working on a NUMA machine, think about the data distribution of the jacobi code. Change the data initialization for a better data distribution if needed. If you wish, you can also parallelize the error check as well.

2 Reasoning about Work-Distribution (sin-cos)

Go to the `for` directory. Compile the `for` code via ‘`make [debug|release]`’ and execute the resulting executable via ‘`OMP_NUM_THREADS=procs make run`’, where *procs* denotes the number of threads to be used.

Exercise 1: Examine the code and think about where to put the parallelization directive(s).

Exercise 2: Measure the speedup and the efficiency of the parallelized code. How good does the code scale and which scaling did you expect?

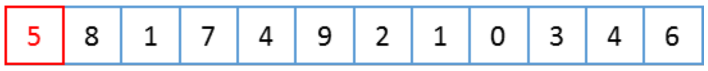
# Threads	Runtime [sec]	Speedup	Efficiency
1			

Is this what you expected?

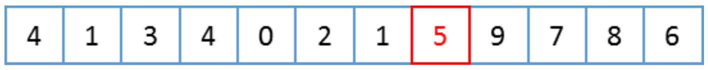
3 Quicksort

Quicksort is a recursive algorithm which, in this case, is used to sort an array of random integer numbers. How it works is described in the following steps.

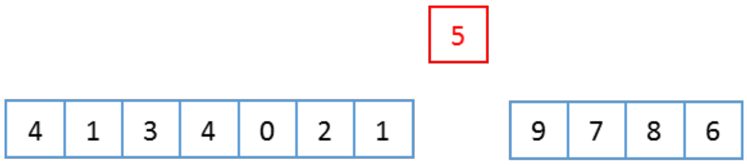
A pivot element is chosen. The value of this element is the point where the array is split in this recursion level.



All values smaller than the pivot element are moved to the front of the array, all elements larger than the pivot element to the end of the array. The pivot element is between both parts. Note, depending on the pivot element the partitions may differ in size.



Both partitions are sorted separately by recursive calls to quicksort.



The recursion ends, when the array reaches a size of 1, because one element is always sorted.

Go to the `quicksort` directory. Compile the `Quicksort` code via '`make [debug|release]`' and execute the resulting executable via '`OMP_NUM_THREADS=procs make run`', where procs denotes the number of threads to be used.

Exercise 1: The partitions created in step 3 can be sorted independent from each other, so this could be done in parallel. Use OpenMP Tasks to parallelize the quicksort program.

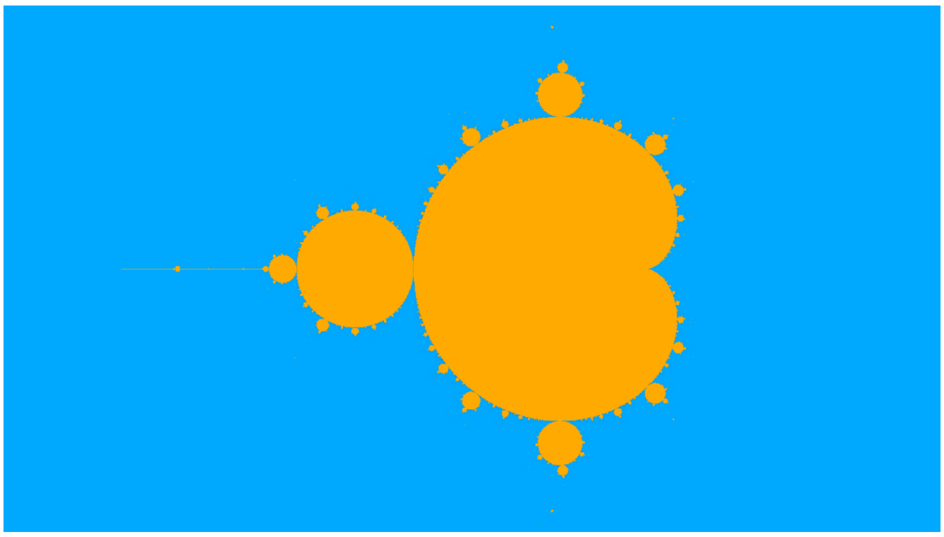
Exercise 2: Creating tasks for very small partitions is inefficient. Implement a cut-off to create tasks only if enough work is left. E.g. when more than 10k numbers have to be sorted, a task can be created, for smaller arrays no task is created.

Hint: You can add if clauses to the task pragmas.

Exercise 3: The if clause needs to be evaluated every time the function is called, although the array size does not exceed 10k elements on a lower level. Implement a serial_quicksort function and call this function when the array gets too small. This can help to avoid the overhead of the if clause.

8 Mandelbrot

The Mandelbrot set is a set of complex numbers that has a highly convoluted fractal boundary when plotted. The given code computes and plots the Mandelbrot set. The generated plot looks like this:



Go to the `mandelbrot` directory. Compile the `mandelbrot` code via '`make [debug|release]`' and execute the resulting executable via '`OMP_NUM_THREADS=procs make run`', where procs denotes the number of threads to be used.

Exercise 1: Execute the code with one thread and with multiple threads and compare the resulting pictures. Do they look as the picture above?

Exercise 2: One of the pictures is incorrect. Do you have an idea what is going wrong? Do you know a tool which can help you to find the error? Try to detect and fix the error in the code.