1 Exercise #1: Basic Parallelism

In this first exercise you should explore different interfaces for parallel programming in native programming-languages. You will also analyze the runtime characteristics of your final programs. As a starting point you should use the example provided with the materials for this assignment[1]. This program paints a graphical representation of the approximation of the *Mandelbrot set*[2]. For the purpose of this exercise you can ignore the mathematics behind it. The `mandel(x, y, max)` function calculates an integer value based on its arguments `x,y` up to a maximum value of `max`. With `getRGB()` a nice-looking color can be computed for a pair of the return value and `max`. The example program does this for a range of `x` and `y` values and prints out the final image in PPM[3] format.

1.1 Make it parallel

Make yourself familiar with the code and identify the region where the iteration values of the *Mandelbrot set* are computed. Create multiple versions of this code and make them calculate the values in parallel. Use the following interfaces, one for each version of your program:

- Pthreads
- C++11 threads
- OpenMP
- `clone()` C-library wrapper around the Linux system-call
- `fork()` POSIX function
- MPI, preferably the *OpenMPI* implementation

It should be easily configurable how many control flows will do the computation. Figure out how to use each interface, look for documentation and examples.

1.2 Measure

Measure how your solutions scale in performance with respect to the number of CPUs used. Create plots for each solution, use the number of CPUs on the horizontal axis and the speedup for the vertical axis. Measure each data point several times and use the average number to compensate for fluctuations. Set the number of iterations high enough to run the computation for several seconds even with the maximum number of CPUs used. Measure the whole-program running-time using the *time* command. With the help of *Amdahl's Law*, calculate the percentage of the serial and parallel portions of your programs based on the execution numbers for 1 and 2 CPUs. Take a look at the memory usage output from *time*, too. Compare the data and draw conclusions for which parallelization API you could achieve the highest performance and/or least memory usage. Think of reasons why this could be. What are the main differences between `fork`, MPI and the other APIs?
Please provide your plots and answers in a single PDF document.

1.3 Submit

Submit your solutions by creating the directory /proj/i4cs/students/`your_login`/assignment1/ All files in this folder will be collected after the submission deadline. The file `comments.txt` will be created in this directory and contains comments from the tutors. Please create a file `group.txt` with your and your partner's login if you do your assignments in a group of 2 people.

Remarks:

- Try to keep it simple, in most interfaces you do not need much more than 2 concepts: start parallel execution and wait for it to end. The computation of each value is independent of the other computations.
- In the case of `fork` it is OK to print n picture-parts in n files, where n is the number of control flows.
- In this semester you can gain access to a few of our chair's servers, including `faui49big01` and `faui49big02` they might be interesting for measurements. **Beware:** those servers can be restarted at any time without notification and other processes may run that might interfere with your measurements.

Submit until: 2014-11-04

[1] `https://www4.cs.fau.de/Lehre/current/V_CS/Uebungen/aufgaben/material1.tar.gz`
[2] `https://en.wikipedia.org/wiki/Mandelbrot_set`
[3] `https://en.wikipedia.org/wiki/Netpbm_format`