UNIVERSITÄT SIEGEN

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

Excercise Sheet 1

(To be processed until 05.11.)

Lecture Parallel Processing
Winter Term 2024/25

Exercise 1: Run-Time Analysis of a Sequential Program

Before a program (which typically is not known in all its details) is parallelized, it is common practice to use performance analysis tools in order to identify the most compute intensive parts of the program code.

A simple, sampling-based tool for this purpose is the program `gprof`, which measures for each procedure (or method) P in the program how often it is called and how much time is required to process P. In doing so, the tool differentiates between

- the *inclusive time* of P, which also includes the processing time of the procedures called by P (*children*), and
- the *exclusive time* (or *self time*) of P, which does not include the processing time of these calls.

When, e.g., the procedure

```
void funcA() {
   for (int i=0; i<N; i++) { ... }
   funcB();
}
```

is invoked, the *exclusive time* of `funcA` is only the runtime of the `for` loop, while the *inclusive time* also includes the runtime of `funcB`.

In this exercise, you should familiarize yourself with `gprof`. First, compile the file `example.cpp` (in the archive `u01eFiles.zip`[1] on the lecture's web page) with instrumentation for `gprof`:[2]

```
g++ -pg -g -o example example.cpp
```

Start the program with `./example`. During the execution, the gathered profiling data is written to a file `gmon.out`. A readable form of this data is then obtained using the command `gprof ./example`. Thoroughly look at the output, in particular the included explanations, and answer the following questions:

a) Which function requires the most computing time (*exclusive*) and would therefore be the first candidate for parallelization?

- **(i)** What is the maximum speedup you could achieve, if only the body of this function were parallelized?
- **(ii)** How often is the function called?
- **(iii)** What other functions does the function call?

b) Which function is most frequently called and from which other functions?

c) How is the computation time of the function `func5` composed?

Exercise 2: Data Dependences, Programming with C++ Threads (Compulsory Exercise! Submit until Tuesday, November 05th, 10:00 via moodle)

Look at the file `compute.cpp` in the archive `u01eFiles.zip`[3] on the lecture's web page. First, analyse the data dependences between the different function calls. The functions `f1` to `f7` do not have any side effects, thus they, e.g., do

[1] `http://www.bs.informatik.uni-siegen.de/web/wismueller/vl/pv/u01eFiles.zip`
[2] By the way, the code in this file is a nice example of *obfuscation*, to prevent you from inspecting the (rather short) source code.
[3] `http://www.bs.informatik.uni-siegen.de/web/wismueller/vl/pv/u01eFiles.zip`

not access global variables.

Based on the data dependences, determine which calls of the functions `f1` to `f7` can be executed in parallel. Try to remove the occurring anti dependences and output dependences (if any) by renaming one or several variables.

Then, draw a task graph (see Section 2.7.7 of the lecture) representing the dependences between the different tasks, where in this case a task corresponds to a function call. In the example, all functions have an execution time of $1s$. What is the execution time of the parallel program?

Parallelize the program using C++ Threads. If neccessary, implement a proper synchronization using condition variables (see the comment in the code for a hint). Pay attention to ensure that the printed result is *exactly* the same as with the sequential program.