Multicore Programming

Lab 1: Introduction and laws of parallel programming

Assistant: Janwillem Swalens
Mail: jswalens@vub.ac.be
Office: 10F719

12 February 2014

This first lab session starts with some hands-on experience in parallel programming. You will fix a parallel Java "Hello World", and see why concurrency matters even when performance doesn't. Next, we will have a more theoretical discussion of Amdahl's and Gustafson's law.

Material

You will find an archive with an Eclipse project on Pointcarré. If not noted otherwise, the files are in the default package.

Import the Eclipse project into your workspace before you start the assignments, using File ► Import...; select General ► Existing Projects into Workspace; choose Select archive file, and select the downloaded .tar.gz file.

1 A parallel Hello World

`HelloWorld.java`

It cannot be so hard to print Hello World correctly on the screen, can it?

`HelloWorld.java` implements a typical parallel version; splitting the problem up into smaller tasks. Each task takes one character of the `Hello World!` string out of a queue, processes it, and puts it in an output queue. The tasks themselves are scheduled to be executed on one of four threads. Thus, by processing the tasks in parallel, the guarantees implied by sequential execution are not given anymore.

Fix the `processLists(...)` method to re-enforce the sequential guarantees.

Hints:

- HelloWorld is a JUnit unit test and it can be execute as one. In Eclipse, press ▼ next to the run icon, and select "Run as... ► JUnit Test".

- Java has a `synchronized` keyword that might be of use. (cf. http://docs.oracle.com/javase/tutorial/essential/concurrency/syncmeth.html)

2 Concurrency vs. Parallelism

`ConcurrencyInsteadOfPerformance.java`

Concurrency is not necessarily about performance, as parallelism is. Instead, many use cases for threads in today's applications focus on providing responsiveness to improve the user experience. You do not need multiple cores for that. The operating system offers time-multiplexing for such purposes.

Try and execute the example implemented in `ConcurrencyInsteadOfPerformance.java`. You will notice that its behavior is not user-friendly: the user interface blocks while calculating prime numbers.

Move the loop that checks for prime numbers into a worker thread and update the text area every time a prime number is found, like it is currently done (even though it is not visible currently).

Use `ProgressBarDemo.java`[1] as a guideline on how to improve responsiveness of the user interface. The worker thread can be implemented by creating a class that implements `Runnable`, and providing a `run()` method.

3 Amdahl's Law: The Bad News First

The goal of this assignment is to visualize the consequence of Amdahl's law for two examples. Use a graphing program such as a spreadsheet to plot the following implications of Amdahl's Law.[2]

$$\text{Amdahl's Law: } S_P = \frac{T_1}{T_P} = \frac{1}{S + \frac{1-S}{P}}$$

- P is the number of processors
- S_P is the speed-up on P processors
- S is the sequential portion (between 0 and 1)
- Consider the speed-up when $P = 256$ of a program with sequential portion S where the portion $1 - S$ enjoys perfect linear speed-up. Plot the speed-up as S ranges from .01 (1% sequential) to .25 (25% sequential).

[1] Adapted from http://docs.oracle.com/javase/tutorial/uiswing/examples/components/ProgressBarDemoProject/src/components/ProgressBarDemo.java

[2] This assignment is borrowed from Dan Grossman http://www.cs.washington.edu/homes/djg/teachingMaterials.

- Consider again the speed-up of a program with sequential portion S where the portion $1 - S$ enjoys perfect linear speed-up. This time, hold S constant and vary the number of processors P from 2 to 32. On the same graph, show three curves, one each for $S = .01$, $S = .1$, and $S = .25$.

What do you conclude from these graphs? Which implication does Amdahl's law have for the parallelization of existing sequential code?

4 Gustafson's Law: The Good News

The goal of this assignment is to visualize the consequence of Gustafson's Law. Similar to the first assignment, use a spreadsheet to plot the implications implied by the two following examples.

$$\text{Gustafson's Law: } S_P = \frac{T_1}{T_P} = P - (P-1)S$$

- Consider the speed-up when $P = 256$ of a program with sequential portion S where the portion $1 - S$ enjoys perfect linear speed-up. Plot the speed-up as S ranges from .01 (1% sequential) to .25 (25% sequential).
- Consider again the speed-up of a program with sequential portion S where the portion $1 - S$ enjoys perfect linear speed-up. This time, hold S constant and vary the number of processors P from 2 to 32. On the same graph, show three curves, one each for $S = .01$, $S = .1$, and $S = .25$.