Introduction to Parallel and Distributed Computing Exercise 3 (June 11)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.uni-linz.ac.at

April 28, 2008

The result is to be submitted by the deadline stated above via the Moodle interface as a .zip or .tgz file which contains

- a PDF file with
  - a cover page with the title of the course, your name, Matrikelnummer, and email-address,
  - the source code of the sequential program,
  - the demonstration of a sample solution of the program,
  - the source code of the parallel program,
  - the demonstration of a sample solution of the program,
  - a benchmark of the sequential and of the parallel program in the style of Exercise 2.
- the source (`.c`/`.java`) files of the sequential program and of the parallel program.

Exercise 3: Multi-Threaded Parallelization

The goal of this exercise is to develop a multi-threaded parallel version of the Gaussian Elimination program developed in Exercise 2 using

- either the programming language C/C++ with the POSIX Thread and Socket API,
- or the programming language Java with the standard API for multithreading and socket communication.

First, take the sequential solution (possibly translated to Java) and benchmark it with appropriate values for N (in the case of Java, adapt the constant in `smult` to give reasonable timings).

Next, develop a multi-threaded version of the program that can be started with a command-line parameter T denoting the number of threads that shall be used for parallel execution. Benchmark this program as in Exercise 2 and report the corresponding results.

Hint: for benchmarking Java programs, you may use the function

```
System.currentTimeMillis()
```

which returns the current wall clock time in milliseconds.

Alternative Version of Exercise (20% Bonus) Write the program such that it can be started

1. either with a command line parameter `-server`; in this case the program is executed in server mode in which it repeatedly constructs a random equation system and then waits (on some designated port) for the request of a client to solve the system with a particular number of threads,
2. or with a command line parameter `-client` T; in this case, the program is started as a client that contacts the server on the designated port, sends the parameter T to the server, and waits for an acknowledgement that the execution has terminated.

Both server and clients may be run on the Altix machine.

If you choose this version, please start a server process and repeatedly benchmark the time for the execution of the client from the point where it sends T to server until the time it receives the answer (rather than benchmarking the execution of the server).