COMP60011 - LAB 1

Parallel Programming with Java Threads, basic synchronization and locks

The marking will be done in two ways. The general design, structure of the algorithm and implementation will be explained through Q&A during the lab sessions. Secondly a written report will be produced describing the parallel algorithm implemented, the performance results obtained (see the specific questions asked below) and also any parallelism related bug that has occurred during the lab.

By the end of lab session on 16th Nov, each group has to describe through Q&A the design of the parallel algorithm and demonstrate debugging a parallel program (at 16:30 Mikel & Ian will stop one group at the time and you will sign the attendance sheet). The written report, a document (only PDF, ODF, or Word 2003), will be emailed with subject "COMP60011 Lab 1 Group X" to comp60011@googlemail.com before 9:00am on Monday 23rd Nov.

For these exercises you can use the locks part of the java.util.concurrent package, but nothing more from there.

Debugging a Parallel Program – The objective is to learn how to use a Java debugger for parallel programming (either Netbeans or Eclipse). Using the Deadlock.java file part of the Java Concurrency Tutorial (http://java.sun.com/docs/books/tutorial/essential/concurrency/example/Deadlock.java), demonstrate using a debugger the deadlock scenario. *This is not part of the report.*

Parallel Sorting - The objective is to develop a parallel program that

(1) reads as command line parameters the number of threads, a seed (integer value), size of an array of integers and timeout parameter (in seconds),
(2) creates an array of `int` of the appropriate length and randomly initialised (use the seed in the constructor of the random number object),
(3) performs parallel sorting using merge-sort,
(4) performs a sequential check to ensure that array is now ordered,
(5) prints to the standard output the execution times (excluding initialization and sequential checker), the parameters used, and the result of the test,
(6) should the application take longer than the timeout parameter, one of the threads needs to print to the standard output the problem and finish the application (System.exit(-1)).

In each question, you first need to run the parallel program with one thread and obtain the execution time. Use three times that execution time as the timeout parameter of the program.

QI.1 Produce a speed up graph, where you have fixed the seed to 1353, where you have fixed the array size to 100 and you change the number of threads 1, 2, and 4. Comment on the results that you obtain and compare them against the ideal speedup.
QI.2 Produce a speed up graph using the same seed, this range of threads 1, 2 and 4, and change the array size so that it does not fit in L1 cache of a core i7 system. Explain the selected array size and how you calculated it. Comment on the results that you obtain and compare them against the ideal speedup and the results obtained in QI.1.
QI.3 Produce a speed up graph using the same seed and the range of threads 1, 2, 4, 6, 7, 8, and 16, change the array size so that it does not fit in L2 cache of a core i7 system. Explain the selected array size and how you calculated it. Again, comment on the results that you obtain and compare them against the ideal speedup and the results obtained in QI.1 and QI.2. Specially comment on results with 6 ..16 threads.