CSc 422/522 — Parallel Programming Experiments and Reports

due Tuesday, March 23

General Information

How often when given a programming assignment have you been satisfied just to get it to work on a reasonable number of test cases? (I know the answer!) How often have you stopped to analyze its performance? (I know this answer too :-)

Your task for this part of the assignment is to find out how well your program runs, and then to ask why. Do program development on Lectura, and convince yourself that the programs are correct. (Look at the output results and check for consistency and convergence.) Then run a series of timing experiments on Parallel as specified below.

Parallel is a Sun Microsystems multiprocessor with four 50 MHz SPARC processors and 128M bytes of main memory. Each processor has first-level instruction and data caches and a second-level combined cache. The instruction cache size has 20K bytes and is organized as a five-way set associative cache with 64-byte cache lines (blocks), which are loaded as two contiguous 32-byte lines. The data cache has 16K bytes and is organized as a four-way set associative cache with 32-byte lines. The second-level cache has 1M bytes, with 128 byte cache lines; it is loaded in parallel as four separate (but contiguous) 32-byte lines.

For each test of a parallel program, be sure to set the `SR_PARALLEL` environment variable to the number of worker processes. *Do not just set* `SR_PARALLEL` *to 4 and then run all tests.*

Be sure to run each timing test enough times to gain confidence in the result. For example, for each combination of command-line arguments, execute each program say three times. If the times are about the same, the average of those three numbers would be a reasonable approximation to the actual time. You might want to use a shell script to automate running timing tests. (You can also use the Unix `at` command to schedule a script to run later.)

You can do most, if not all, your performance testing on your own. We are the only class using Parallel, so you are competing only with your classmates. If necessary, we will establish a sign-up system for testing times.

Cautions. Be very careful with your timing experiments, especially if you use the `at` command. Make sure your programs don't run too long. If you kill a program, make sure that no Unix processes are left lying around (use the `ps` command to check for them, and the `kill` command to kill them). Killing an SR program can sometimes lead to peculiar results. If you have runaway processes, you will affect future timings for both yourself and everyone else.

Basic Tests

Everyone is to run the following tests for 1000 iterations each:

sequential programs for grids of size 48 and 96
parallel programs for grids of size 48 and 96, using 1, 2, 3, and 4 workers

There are a total of 20 different tests. Graduate students will want to adjust grid sizes slightly for your multigrid programs so the different levels work out.

Additional Tests for Graduate Students

Devise additional tests to answer the following questions:

How much time does it take for each algorithm to get the same level of accuracy? For example, take the less accurate program (presumably Jacobi) and see how many more iterations and time it takes to get the same value of epsilon as the other program.

How much more accuracy can you get for the same amount of time. For example, take the faster program and run it for more iterations so that it takes about the same amount of time as the slower program. How much does the accuracy improve?

What is the effect of hand optimizing the barriers? For example, specialize the code to the exact number of workers and inline it.

What is the effect of assigning *stripes* to each worker rather than contiguous strips. A striped allocation assigns every `W`'th row to a worker; see page 1124 of the text for an explanation and figure. Does the program get faster or slower?

You are encouraged (but not required) to come up with additional questions and to answer them.

Reports

After conducting your experiments, write a report to explain what you have done and analyze the results. Undergraduate student reports can be brief; graduate student reports should be more substantial. Your report should have four or five sections, as follows:

- *Introduction.* Briefly describe the problem and what your report will show.
- *Programs.* Describe your programs, stating what each does and how. For the sequential programs, explain the optimizations you have implemented.
- *Basic Results.* Present the results from the 20 basic tests and explain what they show. Do not just present the output data! For the parallel programs, draw a graph that has 16 data points and then draw four curves through the points: one for each combination of program and grid size. The four points connected by each curve will be the times for the different numbers of workers for that program and grid size. Give a table that lists the execution time for each test (20 entries); explain how you got the numbers. Finally, give a table that shows the speedup for each parallel program. It will have 16 entries, including one for the parallel program on 1 processor. Recall that the numerator for speedup is the time of the corresponding sequential program.
- *Additional Experiments* (graduates only). Describe the additional experiments you conducted, present the results, and analyze them. Present the results in whatever form seems most compelling to you. Your analysis should explain why you think you got the results you did.
- *Conclusion.* Summarize what your report has shown. Also describe what you have learned from this project.

Append commented listings of your four programs to your report. (Also submit them electronically, unless they are the same as the programs you submitted on March 9.) You do not need to turn in the actual output from any of your tests, but you should have it available or readily be able to reproduce it. In short, your report should contain all the information someone else would need to reproduce your results.