CSc 422/522 — Parallel Programming Project

programs due Tuesday, March 7 (by midnight)
final programs and reports due the week of March 20 (details later)

In this project you will learn about the gravitational N-body problem, conduct timing and other experiments to analyze the behavior of your programs, and write a report presenting your results. The project is worth 40 points for undergraduates and 60 points for honors and graduate students. You may work on your own or with one other classmate.*

There are four parts to this project as described below: writing four programs, conducting timing experiments, conducting other experiments, and writing a report. Graduate and honors students are to do all parts. Undergraduates can *either* write just the first two programs—and do the other parts for those two programs—or write all four programs and do just the timing experiments on them.

Programs

The starting point for this project is the following four programs. You may write the programs in SR or in C with the Pthreads library. Develop your programs on Lectura or at home, but run experiments on Parallel.

1. a sequential N^2 program, as in Figure 11.9.
2. a parallel N^2 program, as in Figure 11.11.
3. a sequential $N \log N$ program using the Barnes-Hut method.
4. a parallel $N \log N$ program using the Barnes-Hut method.

The Barnes-Hut method is described in Section 11.2.4 of the text, but I do not give actual code. It is up to you to figure out the details for the sequential program and to figure out how to parallelize that program. There is lots of information on the Web. For example, a search for "Barnes-Hut" using AltaVista returns several links.

It is up to you to figure out how to initialize data in such a way that you can tell that your programs are correct. You also want to be sure that bodies do not get too close to each other. You need to implement an efficient barrier. Finally, you should make the sequential parts of your programs as efficient as possible; see the discussion on pages 539-40 of the text for ideas on things you can do to the source code so that the SR or C compiler generates faster assembly code.

Your programs should have the following command-line arguments, in this order:

`numBodies` -- the number of bodies
`numSteps` -- the number of time steps in the simulation
`far` -- the distance used to decide when to approximate (Barnes-Hut programs only)
`numWorkers` -- the number of workers (parallel programs only)

Feel free to use other arguments in your programs, such as for the value of `G` and the initial mass

*If a two-person team consists of an undergraduate and graduate or an undergraduate and honors student, then you must do the larger project. However, the undergraduate will only get a maximum of 40 points.

of the bodies. However, put them after the above arguments and assign default values in your programs so that we can test programs using just the above arguments. (See below for turnin details.) You may also read data from an input file if you wish, but I suggest that you simply use "canned" data that you can generate algorithmically.

The output of each program should be the execution time for the computational part. Read the clock after you have initialized all variables and just before you create the processes (in the parallel programs). Read the clock again as soon as you have completed the time steps and the worker processes have terminated (in the parallel programs). Then write a single line of output to standard out:

```
computation time:  xxxx
```

You may also want to write the final data values to a file—at least while your are debugging your programs—but write only the above line to standard out.

Timing Experiments

First experiment with your first program (sequential N^2) to determine how many time steps to use so that the computation time of the program is about 15 seconds for 120 bodies. In short, figure out what value to use for `numSteps` so that *your* program computes for 15 seconds for 120 bodies.

Next run a series of timing experiments to calculate the computation time for all of the following combinations of programs and values for the command-line arguments:

> program 1 for 120, 180, and 240 bodies.
> program 2 for 120, 180, and 240 bodies and 1-4 workers.
> program 3 for 120, 180, and 240 bodies.
> program 4 for 120, 180 and 240 bodies and 1-4 workers.

In all programs, set `numSteps` to the value you calculated in the first step. In the Barnes-Hut programs, set `far` to a value so that most bodies (80 to 90%) are considered to be far enough away to use approximations. You will, of course, have to experiment with program 3 to determine a reasonable value to use.

There are a total of 30 different timing runs—3 for program 1, 12 for program 2, 3 for program 3, and 12 for program 4.

Other Experiments

The experimental method consists of making and then testing hypotheses—or asking questions and then determining answers. There are lots of different questions one might want to ask about the above programs, such as:

> *Overheads*. How much time does it take to create processes? How long does it take to perform a barrier for 4 workers?
>
> *Time/Space Tradeoffs.* Is false sharing happening for the barrier synchronization variables, and if so, can you get rid of it by padding the declarations? Program 2 uses a matrix of values for forces to avoid critical sections; what happens to performance if you use a single vector of forces and add critical section locks to updates to that vector?
>
> *Load Balancing.* Program 2 uses the striped allocation pattern to assign bodies to workers. How much better is it to use stripes that to use blocks (strips)? Would it be even better to

use reverse stripes? If so, how much better?

Optimizations. What is the effect of turning on compile time optimizations (using the "`-O`" option to the `sr` or `gcc` compiler). How much faster are the programs? What happens to the speedups for the parallel programs?

Barnes-Hut Method. How does the choice of the value for `far` affect performance? How much of the computation time is spent constructing the quad tree? Computing masses in the quadtree? Calculating forces using the quadtree?

These are just some of the questions one might ask. You can probably think of others.

Pick *at least three nontrivial questions* and set up experiments to determine the answer. The choice of what to do is up to you, but do not choose just the simplest things. You might look at three different kinds of topics—such as overhead, time/space, and load-balancing—go into depth on one topic, or do a combination.

Reports

Once you have done the timing and other experiments, write a report to explain what you have done and what you have learned. Your report should be a few pages of text plus tables and figures. It 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. Explain the program-level optimizations you have implemented.
- *Timing Experiments.* Present the results from the timing experiments. Use tables to present the raw data and graphs to show speedups and comparisons. *Also explain your results.* Do not just present the output data! What do the results show? Why?
- *Other Experiments.* Describe the questions that you set out to answer, the experiments you conducted, the results you got, and your analysis of the results. 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.* Briefly summarize what your report has shown. *Also describe what you have learned from this project.*

Electronic Turnin

By midnight on March 7, use `turnin` to submit your programs. The assignment name is `parallel`. The programs should be named `prog1`, `prog2`, etc. *Also submit a makefile that we can use to compile your programs.* In particular, if we execute

```
make prog1
```

your makefile should compile the program and produce an executable file that resides in `a.out`. We should then be able to execute the program with command-line arguments as specified above.

Before you turn in your report, turn in the final versions of your programs. Please append commented listings of your final programs to your report. 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.