Parallel Project — n-Bodies and Collisions

Assignment due:

Email: group names or solo decision by: Monday, March 23rd.

Due: Program (electronic) and report (electronic) due: 9:00 pm, Tuesday, April 14th.

Presentations: April 15th, 16th, or 17th

This project is worth 15% of your final grade.

You may work solo on this project, or with a partner. If you choose to work with a partner, understand that means you will be working together for both parts: the development of the programs and the timing report. The final project for the course can also be done in teams of two; however, you do not have to be on the same team for the final project. Email your group members, or your decision to work solo by March 13th.

This project involves solving the n-body problem. The textbook provides an explanation of this problem. See section 11.2, 11.2.1, and 11.2.2, pages 553 to 559. (Section 11.2.3 covers distributed versions of this problem). The textbook also explains optimizations that take advantage of the inverse-square nature of the gravity equation (see section 11.2.4).

Place the objects in a 2-dimensional space (note that the textbook's examples are in a 2-dimensional space).

For this problem, we will use bodies that are relatively close together, so the approximate methods will be not necessary. Instead, to make the problem more "interesting", your solution will check for collisions among the objects. Assume elastic collisions. An elastic collision is one in which two objects bounce off each other without deformation or loss of total energy. The directions of travel for both objects will be different after the collision. You can assume that all objects are the same size (this will be a command-line parameter, see below).

You will submit your programs and a report presenting the results of your timing runs and experiments. You will also make an oral presentation of your project and results. The presentations will be scheduled April 15th through April 17th. If you finish your project early and wish to do the presentation early, please let Patrick know.

Programs:

Write a good sequential program for this problem. Look specifically for optimizations that you can apply to the sequential program to improve its performance. Include a description of these optimizations in your report.

Develop a parallel solution for the problem from your sequential solution. You will need an efficient barrier to synchronize your workers. Your solution should work for any number of workers from two through sixteen. In your report, include a discussion of the barrier that you used and optimizations that you applied to the parallel algorithm.

Both your sequential and parallel solutions will take the following command-line arguments:

- number of workers, 1 to 16. This argument will be ignored by the sequential solution.
- number of bodies.
- size of each body.
- number of time steps.

You may add additional command-line arguments as needed to help in your experimentation. Provide default values for these additional arguments when they are not present. Examples can include a random seed, a random selection of sizes if all the balls are not the same size, etc.

The output of each program should be the execution time for the computational part. Read the clock after you have initialized all the bodies. Read the clock just before you create the processes (in the parallel program). Read the clock again as soon as you have completed the time steps and the worker processes have terminated (in the parallel programs).

Write to standard out:

```
   computation time: xxxx seconds xxxx milliseconds
```

and the number of collisions detected.

Write the final positions and velocities to a file.

Language:

You may write your program in Java or C.

You can use C++, if you wish.

If you wish to use other variants of C, please talk to Patrick first.

Timing Experiments:

For your sequential program, use a test case that will take at most a few minutes to complete on one of the multi-core machines in the CS department labs. Now, use the same test case for your parallel solution. Run tests to determine the computation time needed for one, two, three, four, etc. workers. You should run each individual test more than once to get an average (and to allow you to filter out occasional timing fluctuations that occur due to system routines).

Your report should include a table and/or a chart showing the results of the timing runs that you perform. Include enough information so that we can replicate your results. We should be able to compile and run your code on the machine you used and see results close to what you report.

Additional Experiments/Extensions:

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

- *Overheads*. How long does it take to perform a barrier for 2, 3, 4, etc. workers?
- *Time/Space Trade-offs*. Is false sharing happening for the barrier synchronization variables, and if so, can you get rid of it by padding the declarations?

- *Optimizations*. What is the effect of turning on compile time optimizations (for example, the **`-Olevel`** option for the gcc compiler). How much faster are the programs? What happens to the speedups for the parallel programs vs. the sequential program?
- *Barrier performance*. Measure the percent of time your code spends in barriers.
- *Busy waiting vs. semaphores/monitors*. Try two different approaches for the waits that are needed in the barrier synchronization, such as busy-waiting vs. semaphores, semaphores vs. monitors (there are other ways that you can try as well). Does one technique show a marked improvement over another?
- *3 Dimensions*. Write a solution that moves and collides particles in three dimensions. Measure and report the performance of this solution.
- *Graphical interface*. Show the particles as they move and collide. This can be useful in verifying the correctness of your solution. You should be able to run your solution with and without the graphics. How does the execution time vary when using graphics vs. not? Do you still see improvements in the execution time across multiple cores when the graphics are running?

There are other questions you might ask. Pick at least two 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. You are welcome to discuss these options with us, if you wish.

Group vs. Solo:

If you are working with a partner, we expect a better project than if you work solo. "Better" can be in terms of quality or quantity. Examples include performing more experiments, extending the solution to three dimensions with a graphical interface, writing both a C and a Java solution and comparing their performance, measuring the performance of specific sections of your program (such as the barriers mentioned above, how much time is spent in detecting and resolving collisions vs. force and velocity calculations) etc.

You are welcome to discuss options with us in advance.

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/or 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. Here is where you can describe your program-level optimizations for the sequential and parallel versions. Describe any enhancements in your solution.
- Verification. Describe how you verified that your program was working correctly. How do you know that the gravity force and velocities are correct? How do you know that collisions are resolved correctly?

- *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.

Demonstration:

You will demonstrate your project for me. You will need to prepare the demonstration. Plan a presentation that will last about 10 minutes. This will be followed by my questions. The total time for demonstration and questions will be no longer than 20 minutes. If doing a two-person project, both members of the group must be present. A sign-up sheet will be available for times during the period April 15th to 17th. You can also make an appointment for a demonstration time prior to the 15th. I will have a sign-up sheet available in class on Tuesday, April 14th.

Turnin:

Turn in your programs. Include a **`Makefile`** that supports the command **`make all`** that will create the executable or class files for your program. You do not need to turn in the actual output from any of your tests. Instead, turnin a **`README.txt`** file that gives examples of commands that you used in your tests. The idea is to tell us what commands so we can duplicate your results. We will not exhaustively test your program. We will selectively run tests to verify that your program works as you describe.

Note: We are asking for submission of programs via D2L only.

- For the sequential and parallel programs, submit all source files. Include a **`Makefile`** with the target **`all`**. The command “make all” should produce a compiled C executable or all needed Java class files. For the report use **`report.txt`** or **`report.pdf`**.
- Logon to D2L. Select CSc 422. Click on the “Add a File” button. A pop-up window will appear. Click on the “Choose File” button. Use the file browser that will appear to select your file(s). Click on the “Upload” button in the lower-right corner of the pop-up window.
- You are now back at “Submit Files ”. Click on “Upload” in the lower-right of this window. You should now be at the “File Upload Results” page, and should see the message **`File Submission Successful`**.

D2L will send you a confirmation email after each item is placed in the Dropbox. This email is sent to your UA email (which is <yourUANetId>@email.arizona.edu). Retain these emails at least until you have received a grade for the assignment; it is your confirmation that you did submit the program(s).

You may submit the various items more than once. We will grade the last one that you put in the Dropbox.