



Introduction to High Performance Computing Systems CS 1645

University of Pittsburgh
Department of Computer Science
Spring, 2015
Homework 7

Instructions

- The solutions to all the problems in this homework should be the result of the student's individual effort. Presenting the words or ideas of somebody else under your name hinders your skills in academic and professional work and violates the university's policy on academic integrity:
<http://www.provost.pitt.edu/info/ai1.html>
- The submission process for this homework will use the SVN repository:
<https://collab.sam.pitt.edu/svn/CS1645/users/<pittID>>
Create a `hw7` directory into your repository.
- You have to check-in the following files into the `hw7` directory:
 - File `particles.cpp` containing the parallel code for the particle interaction algorithm.
 - File `particles_serial.cpp` containing the serial code for the particle interaction algorithm.
 - File `report.pdf` with a report summarizing your results (it needs to be a PDF file).by 11:59pm **Monday April 13, 2015**.
- Late submissions will not be accepted.
- Your code ***MUST*** run on the Stampede system with the software infrastructure provided (Makefile, main files, and other utility files). You should use the files provided in the following repository:
<https://collab.sam.pitt.edu/svn/CS1645/assignments/hw7>
- You must implement a serial program to compute the particle interaction algorithm. This is a requirement to measure the speedup and efficiency of the parallel version.
- Since this homework requires large-scale data, you need to start the assignment early. Large jobs may spend up to 24 hours in the scheduler's queue before running. Therefore, it is crucial to finish the coding and debugging days before the deadline, so you allow enough time for the runs to complete.
- Be responsible in using the common pool of service units in our allocation. In particular, do **not** submit a job for more than 5 minutes if you are using more than 256 cores.
- Consider using interactive job submissions for developing and debugging your code:
`idev -A TG-CIE140012`
- To gather the experimental results, in particular the large-scale data, use the provided batch script:
`sbatch submit.batch`

Particle Interaction

A very common pattern in scientific simulations is to solve a n-body problem. Such problem involves a system of particles interacting with each other via *van der Waals* forces, using a (over)simplified model, in a 2D world. The calculation for the force f between particles $\langle x_1, y_1 \rangle$ and $\langle x_2, y_2 \rangle$ is given by the following formulas:

$$\begin{aligned} f &= \frac{A}{r^6} + \frac{B}{r^{12}} & r_x &= x_1 - x_2 & r_y &= y_1 - y_2 \\ r &= \sqrt{r_x^2 + r_y^2} & f_x &= \frac{f * r_x}{r} & f_y &= \frac{f * r_y}{r} \end{aligned}$$

where A and B are constants.

There will be an odd number of processors. Each processor will own a subset of the total particles. Each particle has a $\langle x, y \rangle$ position, mass, and force associated with it. The MPI program calculates the force on each particle. The processors will be arranged in a ring such that it will only receive messages from the previous processor and send to the next processor (with the exception of the last step). The particles are initialized with random locations and forces unless those values are contained in a provided text file.

If an input file is provided, rank 0 will be the only one reading the particle information. You must use collective communication operations to distribute the particles among the processors. Similarly, after the simulation is finished, rank 0 will collect all particles and output their information. Again, collective communication operations must be used to collect particles.

The simulation algorithm is composed of the following steps:

1. Each processor sends its particles to the next processor (location, mass, and running total of forces so far).
2. Upon receiving a set of particles, a processor calculates the resulting forces from the local particles to the remote particles. You must call function `compute_interaction` with `locals` and `remotes` (in that order).
3. Each processor updates total force on local particles and remote particles.
4. Each processor sends the remote particles, along with the calculated forces, to the next processor.
5. Repeat previous steps for $(p - 1)/2$ stages.
6. Each processor sends remote particles back to their original processor.
7. Each processor calculates the forces resulting from its own particles. You must call functions `merge` and then `compute_self_interaction` (in that order).
8. Each processor finishes execution.

Figure ?? presents an overview of the algorithm. The system contains 5 processors in this case and each processor is represented by a circle with a distinctive color. Messages carrying particles are rectangles with a vertical line representing the particles being transferred in a message. Horizontal lines in a message represent the effect of other particles whose effect has been calculated. For instance, the message in Figure ?? going from the red to the green processor is carrying the purple particles and has collected the effect of interacting with the red particles.

The format of messages requires the following information for each particle to be passed:

- Location: x and y .

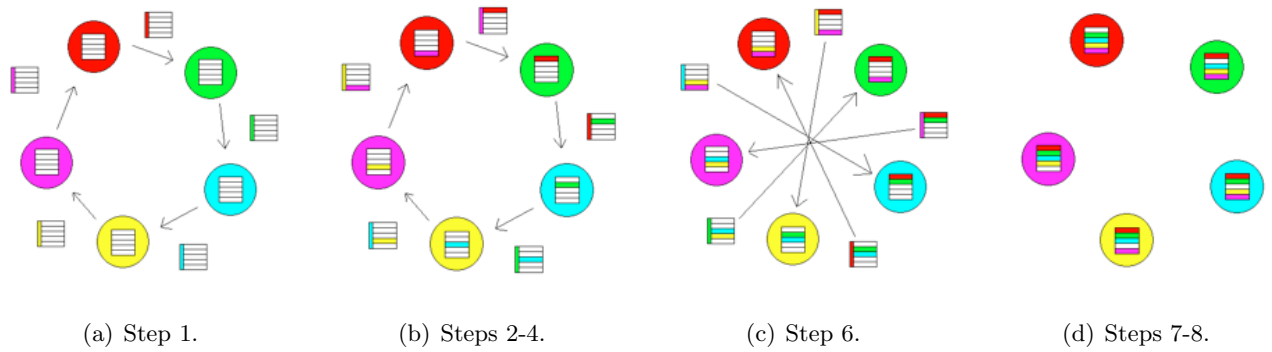


Figure 1: Overview of algorithm to compute particle interaction.

- Running sum of forces in x and y .
- Mass.

The program should take at least one command line parameter:

`./particles <N> [<file>]`

where N is the number of particles. The program should divide N as evenly as possible among the total number of processors.

Report

You should create a report with the following sections:

1. A general strategy of the parallelization effort. Why did you choose those MPI operations to parallelize the program?
2. A speedup analysis. Using an *interesting* value of N , the number of particles, report the speedup for a number of cores ranging from $\{63, 127, 255, 511, 1023\}$. Repeat each experiment at least 3 times and report the average value.
3. An efficiency analysis. Using an *interesting* values of N , number of particles, report the efficiency for a number of cores ranging from $\{63, 127, 255, 511, 1023\}$. Repeat each experiment at least 3 times and report the average value.
4. A description of the performance bottlenecks. What is preventing the program from getting higher speedup?

Acknowledgments

Derived from material in a class by Prof. Laxmikant V. Kalé at the University of Illinois at Urbana-Champaign. Images taken from original specification.