Term project – 5.10.2011

1

Term Project #3

- Due: **05/10/2011**
- Presentation: A 10-minute presentation is expected on 05/10/2011 to show people your progress and how hard (?!) you have worked on your term project.
- Content: Please continue update your progress report regarding your term project for this course. The report is basically an extension to your last progress report, and should at least include:
  - Title
  - Objective: Full description of the problem you want to solve.
  - Method and Algorithms: need to update to reflect your serial implementation
    - Clearly list how do you solve the problem. If the solution method comes from some references you found, refer to the reference clearly.
    - Make sure that you can convince everyone that you understand the problem, and you can solve the problem by these methods/algorithms.
    - Do some sample calculation will be helpful.

Term Project, Phase 3

- Application: potential applications of your term project
- ***Serial Implementation***
  - *Flowchart: A flowchart of your overall program.*
  - *Input of your program: If your program takes an input file, document your input file format; if your program interactively get some input from users, document these inputs.*
  - *Execution time: Create some example problems and run them. Report its execution time.*
  - *Output of your program: Show outputs of your program.*
  - *Implementation: your code should be in working condition in our group cluster system, with a makefile to build executables from your source code. Let me know where you put your programs and makefile.*
- Reference: list references that you found to be related to your term project.
- Milestones: dates of delivery of your project

Assignment #5 – 4.26.2011

4

1. MPI parallelization #3

(01.cpp) Parallelization of your assignment #4, version 3.

We will continue parallelizing 00.cpp in the last assignment by using MPI and MPI_Send/MPI_Recv in the following manner (assume there are P processes):

1. For rank=0, allocate enough memory to store all points, and then generate N points. (step 1 – 2 in 00.cpp)
2. Rank=0 broadcasts the number of points to all processes, and then ~~broadcasts~~ scatters sphere data generated to all processes. Therefore, each process gets roughly N/P spheres. We call these spheres set A.
3. Start Measuring T1
4. Compute distances between all spheres in set A, and finds out the max, min, sum of distances in set A. Also, count number of pairs of spheres that collides in set A.

1. MPI parallelization #3

5. Start a loop running for P-1 times
6. Each process sends its set A to its next process;
7. Each process receives a set of spheres from its previous process, and we call them set B. (Note that you may need to do something to avoid deadlocks)
8. Compute distances of spheres between set A and set B, and updates the max, min, and sum of distances. Also, update the summation of distances.
9. Go back to step 5 until the loop ends (iterative for P-1 times)
10. Use MPI_Reduce to find the global max, min, summation of distances, and number of pairs of colliding spheres Q.
11. End of measuring T1
12. Output P, T1, max, min, and average distances and Q.
13. Note that we have simplified the timing into T1, which should be equivalent to T1+T2+T3 in you previous assignment.

2. Discussion with parallel version #3

Please answer and discuss the following questions:

1. Before you answer the following questions, prove that your parallel version #3 obtains identical results as your serial and parallel versions #1 and #2. (note that my steps in 1 is not necessarily complete!)
2. How big the problem can you can run with the parallel version? What controls the problem size that you can run?
3. With N=500, 1000, 5000, 10000, 20000, and (the biggest case that you can run) run your code with 2, 4, 6, 8, 10, 12, and 14 processors. Based on the output data, calculate and plot your speedup curves (NP vs. T1 with six or more different N) and corresponding parallel efficiency curves in two separate charts.
4. Discuss your parallel efficiency. Do you get good parallel efficiency? Why or why not? How do they vary with your N and NP?
5. Compare parallel efficiency and speedup with your previous assignment. Is version #3 better? Or worse?

3. MPI parallelization #4

(03.cpp) Parallelization of your assignment #4, version 4.

We will continue parallelizing 00.cpp in the last assignment by using MPI and MPI_Isend/MPI_Irecv in the following manner (assume there are P processes):

1. For rank=0, allocate enough memory to store all points, and then generate N points. (step 1 – 2 in 00.cpp)
2. Rank=0 broadcasts the number of points to all processes, and then ~~broadcasts~~ scatters sphere data generated to all processes. Therefore, each process gets roughly N/P spheres. We call these spheres set A.
3. Start Measuring T1
4. Compute distances between all spheres in set A, and finds out the max, min, sum of distances in set A. Also, count number of pairs of spheres that collides in set A.

3. MPI parallelization #4

5. Start a loop running for P-1 times
6. Each process sends its set A to its next process;
7. Each process receives a set of spheres from its previous process, and we call them set B. (Note that you may need to do something to avoid deadlocks)
8. Compute distances of spheres between set A and set B, and updates the max, min, and sum of distances. Also, update the summation of distances.
9. Go back to step 5 until the loop ends (iterative for P-1 times)
10. Use MPI_Reduce to find the global max, min, summation of distances, and number of pairs of colliding spheres Q.
11. End of measuring T1
12. Output P, T1, max, min, and average distances and Q.
13. Note that we have simplified the timing into T1, which should be equivalent to T1+T2+T3 in you previous assignment.

4. Discussion with parallel version #4

Please answer and discuss the following questions:

1. Before you answer the following questions, prove that your parallel version #3 obtains identical results as your serial and parallel versions #1 and #2. (note that my steps in 1 or 3 is not necessarily complete!)
2. How big the problem can you can run with the parallel version? What controls the problem size that you can run?
3. With N=500, 1000, 5000, 10000, 20000, and (the biggest case that you can run) run your code with 2, 4, 6, 8, 10, 12, and 14 processors. Based on the output data, calculate and plot your speedup curves (NP vs. T1 with six or more different N) and corresponding parallel efficiency curves in two separate charts.
4. Discuss your parallel efficiency. Do you get good parallel efficiency? Why or why not? How do they vary with your N and NP?
5. Compare parallel efficiency and speedup with your previous assignment. Is version #4 better? Or worse?