Term-project #2

Due: Apr. 13, 2010

- Please prepare a progress report regarding your term project. The report is basically an extension to your proposal, and should minimally include:
  - Title
  - Objective: Full description of the problem you want to solve.
  - ***Methods and Algorithms:***
    - ***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.***
    - ***Draw a flowchart for your serial program.***
  - Application: potential applications of your term project
  - Expected outcome: the outputs that you expect to produce
  - Reference: list references that you found to be related to your term project.
  - Milestones: dates of delivery of your project

1

Recall assignment #2

This program will 1) randomly user-specified number of spheres, and 2) compute distances between all these spheres.

You are to write programs to:

1) Get a user-specified N, D, Seed from command line arguments.
2) After specifying random seed (Seed) using srand(), write a function to generate N random spheres (3 coordinate components + radius), centers of these spheres should be in the region of [-D:D, -D:D, -D:D] and each sphere is numbered incrementally starting from 0. (of course, you need to use dynamic arrays to store all the data!)
3) Write a function to compute and store distances between all spheres, and the distances should be stored in a dynamic 2D arrays to facilitate distance queries. The 2D array may look like the figure in the next slide.
4) **For each sphere, list spheres that collide with it (distances between two spheres are less than their sum of radius).**

All the arrays mentioned (storing spheres & distances) should be dynamic arrays.

2

For this assignment, use the most efficient data layout & compilation flags that you found in assignment #2

3

0. Serial assignment #2

(00.cpp) Serial version

1) Get a user-specified N, D, Seed from command line arguments.
2) After specifying random seed (Seed) using srand(), write a function to generate N random spheres (3 coordinate components + radius), centers of these spheres should be in the region of [-D:D, -D:D, -D:D] and each sphere is numbered incrementally starting from 0. (of course, you need to use dynamic arrays to store all the data!)
3) Measure T1: Write a function to compute and store distances between all spheres, and the distances should be stored in a dynamic 2D arrays to facilitate distance queries. The 2D array may look like the figure in the next slide.
4) Measure T2: compute max, min, and average distances between all balls.
5) Measure T3: For each sphere, count the number of spheres that collides (Q).
6) Output 1, T1, T2, T3, max, min, average distances and Q

4

1. MPI parallelization #1

(01.cpp) Parallelization of your assignment #2, version 1.

Parallelize 00.cpp by using MPI in the following manner:

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** all points generated to all processes.
3. Measure T1: Compute distance by dividing based on the outer loop when calculating distances as the following:

```
for(i=0;i<N;i++) {
    for(j=i+1;j<N;j++) {
        // some code here
    }
}
```

```
// where myEnd-myBegin ~ n/size,
// and size is the number of processes
// obtained from MPI_Comm_size(...)
for(i=myBegin;i<myEnd;i++) {
    for(j=i+1;j<N;j++) {
        // some code here
    }
}
```

1. MPI parallelization #1

4. Measure T2: Find max., min., and average distances between all balls in parallel.
5. Measure T3: Also, each process should locally find the number of collisions between balls, and use MPI_Reduce to find the total number of collisions (Q).
6. Output NP, T1, T2, T3, max, min, and average distances and Q

2. Discussion with parallel version #1

Please answer and discuss the following questions:

1. How big the problem can you can run with the serial version? What controls the problem size that you can run?
2. Proof your parallel version #1 is correct by checking your answers from the parallel version, and they should be identical to your serial version even with random numbers involved as long as you use the same seed! Do you get the same answer from your parallel version regardless the number of processors (NP) that you use? If not, check your code!
3. With the correct parallelization, what is the biggest problem size that you can run?
4. With N=500, 1000, 5000, 10000, and 20000, run your code with 2, 4, 6, 8, 10, 12, and 14 processors. Based on the output data, calculate and plot your 15 speedup curves (NP vs. T1, NP vs. T2 , NP vs. T3 with five different N) and 15 parallel efficiency curves in two separate charts.
5. Discuss your parallel efficiency. Do you get good parallel efficiency? Why or why not? How do they vary with your N and NP? Please separately discuss T1, T2, and T3!

3. MPI parallelization #2

(02.cpp) Parallelization of your assignment #2, version 2.

Parallelize 00.cpp by using MPI in the following manner:

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** all points generated to all processes.
3. Measure T1: Compute distance by dividing based on the outer loop when calculating distances as the following:

```
for(i=0;i<N;i++) {
    for(j=i+1;j<N;j++) {
        // some code here
    }
}
```

```
// size is the number of processes
// obtained from MPI_Comm_size(...)
for(i=0;i<N;i+=size) {
    for(j=i+1;j<N;j++) {
        // some code here
    }
}
```

3. MPI parallelization #2

4. Measure T2: Find max., min., and average distances between all balls in parallel.
5. Measure T3: Also, each process should locally find the number of collisions between balls, and use MPI_Reduce to find the total number of collisions (Q).
6. Output NP, T1, T2, T3, max, min, and average distances and Q

2. Discussion with parallel version #2

Please answer and discuss the following questions:

1. Again, proof your parallel version #2 is correct regardless the number of processors or # of balls that you run your code.
2. With N=500, 1000, 5000, 10000, and 20000, run your code with 2, 4, 6, 8, 10, 12, and 14 processors. Based on the output data, calculate and plot your 15 speedup curves (NP vs. T1, NP vs. T2 , NP vs. T3 with five different N) and 15 parallel efficiency curves in two separate charts.
3. Discuss your parallel efficiency. Do you get good parallel efficiency? Why or why not? How do they vary with your N and NP?
4. How does parallel version #2 compare to parallel version #1? Does #2 has better efficiency than #1? What is the reason one version is more efficient than the other version? Again, please separately discuss T1, T2, and T3.