COS 590 / APC 590
Computational Methods & Their Applications Across Disciplines

Homework #1 (DUE March 26, 2004 by 5PM)

A serial N-body code using the leapfrog method is provided at:

/usr/home/stevenk/590/homework/leapfrog2.c (C version)
/usr/home/stevenk/590/homework/leapfrog2.cpp (C++ version)

This code can be compiled with the commands:

gcc -o leapfrog2 -lm leapfrog2.c (C version)
g++ -o leapfrog2 leapfrog2.cpp (C++ version)

To run the code, type: './leapfrog2 >> data.out' This will write the particle trajectories to the file 'data.out'

1. Compile and run this code, and plot the results.
   Instructions suggesting one way to plot can be found at:
   http://www.artcompsci.org/vol_1/v1_web/node6.html
   Sections 3.3 and 3.5

2. The program reads in a data file to set the initial particle positions. A data file (data.in) containing random positions for 1000 particles is in the same directory as the code. You can change the number of particles simulated simply by changing the value of 'n' in the program code. Try running the simulation for higher and higher values of n.

3. Write a parallel version of this code using MPI. A fixed set of particles should be assigned to each process, and you should do a brute-force calculation of the forces. To help get you started, a framework MPI code is provided at: /usr/home/stevenk/590/homework/hello.c

4. Compare the results (i.e., particle trajectories and energy) of the parallel version with the serial versions (use n = 10 and 4 CPUs)

5. Think about ways in which you can reduce the amount of communication as much as possible, as well as structure the communication for good performance. Implement these methods.

6. Plot speedup versus CPU count curves for a few different values of n (i.e., the number of particles). How does speedup change with number of particles and number of CPUs? Explain briefly why the speedup curves have the shape they do. NOTE: All speedup curves should include at least 1, 2, 4 and 8 CPUs, and n = 10, 100, 200 as well as the highest value of n your code can handle. A brief guide on how to time your parallel code is below.

7. The data file data.2.in provides a different particle distribution for up to 1000 particles. Describe how the distribution of particles in these files differs from the one used above (hint: plot them both). How do you expect the performance of parallel hierarchical N-body methods to compare with parallel brute-force methods for the two different particle distributions? Do you expect

speedup results for parallel hierarchical N-body methods to behave the same way across the two distributions as the parallel brute-force methods?

8. EXTRA CREDIT: use non-blocking sends, and overlapping computation with communication, to achieve greater efficiency.

Please Submit by Email to Ben Phillips (benp@princeton.edu):

?? A copy of your parallel code (before extra credit changes), which should be derived from leapfrog2.cpp
?? Standard output for the runs completed in question #4. For example,

```
Initial total energy E_in = -0.866025
Final total energy E_out = -0.866025
absolute energy error: E_out - E_in = 2.72254e-10
absolute energy error: (E_out - E_in)/E_in = -3.14372e-10
```

?? Raw data files (e.g., data.out) and plots showing the particle trajectories for the runs completed in question #4.
?? Speedup curves from question #6.
?? Speedup curve for at least one value of n from new distribution in question #7.
?? A short write-up of the answers to questions 5, 6 and 7 above.
?? EXTRA CREDIT: A copy of your optimized parallel code, as well as a graph of speedup curves for particle distribution data.in, comparing basic (synchronous) communication with overlapped communication, for n = 200 as well as for the highest value of n that your code can handle.

The PC cluster

To access the PC cluster, you first have to ssh into your CS account. Then you can ssh into one of the cluster nodes: cluster-038, cluster-039 … cluster-046. These are the nodes should be used for developing your code. To restrict your MPI program to these nodes, you should copy the file /home/stevenk/590/homework/machines.txt to your directory and use the machine file option with mpirun. For example: 'mpirun –machinefile machines.txt –np 4 leapfrog2P'

Timing your MPI code

All codes should have timing functions embedded. These will help the user to understand where the time is being spent, pointing out where to concentrate optimization efforts, and ultimately lead to more efficient use of computer time.

Below are the basic MPI timing routines. These return an absolute time in seconds since the program started. Only the relative time should be used, so bracket the area of code of interest and take the difference in time as illustrated below. After summing the time contributions of all compute and communication intensive sections of the code, simply dump an analysis of the timing results at the end of each run.

In Fortran…

```
    DOUBLE PRECISION t0, t_code
    t_code = 0.0d0

    t0 = MPI_Wtime()

c   ... bracket the CPU or communication intensive code of interest ...

    t_code = t_code + ( t0 - MPI_Wtime() )
```

In C…

```
    double t0, t_code=0, MPI_Wtime();
    t0 = MPI_Wtime();

/*  ... bracket the CPU or communication intensive code of interest ...*/

    t_code += ( t0 - MPI_Wtime() );
```