CS 140 Homework 4:
Cilk++ Galaxy Quest

Assigned February 10, 2009

Due by 11:59 pm Thursday, February 19

This assignment is a fairly easy computation just to exercise your ability to use Cilk.

You are the master of a fictitious galaxy that consists of n stars, each of which is represented by (x, y, z) coordinates in space. The stars are not evenly distributed, and each has a different mass m. Your first goal is to compute the center of mass (cx, cy, cz) of your galaxy as fast as possible. Then, you will calculate the average distance $avgdist$ from the center of mass to the stars. Here are the formulas:

$$cx = (x[0] \cdot m[0] + \cdots + x[n-1] \cdot m[n-1])/n \tag{1}$$

$$cy = (y[0] \cdot m[0] + \cdots + y[n-1] \cdot m[n-1])/n \tag{2}$$

$$cz = (z[0] \cdot m[0] + \cdots + z[n-1] \cdot m[n-1])/n \tag{3}$$

$$dist[i] = \left((x[i] - cx)^2 + (y[i] - cy)^2 + (z[i] - cz)^2\right)^{1/2} \tag{4}$$

$$avgdist = (dist[0] + \cdots + dist[n-1])/n \tag{5}$$

$$\tag{6}$$

Finally, your program should be able to return the indices of the k stars that are closest to the center of mass of the galaxy.

You will write a shared memory parallel program in Cilk++ in order to perform these computations. Remember that, as opposed to distributed memory computing in MPI, your data do not need to be physically distributed among processors; i.e., it is best to store your matrix or vector as a continuous chunk of memory. However, you should make sure that your implementation avoids data races. You can use the cilkscreen tool to check your program for any data races. You can view the first part of this problem as a matrix-vector multiplication operation. Here A is a 3-by-n matrix with each column representing the coordinates of a different star. Specifically, $A(0, i)$, $A(1, i)$, and $A(2, i)$ are the x, y, and z coordinates of the i-th star. Then m is an n-by-1 vector that contains the masses of the stars, with $m[i]$ being the mass of the i-th star. The matrix-vector multiplication $c = Am/n$ computes the center of mass of the galaxy and stores it in the 3-by-1 vector c. However, the simple parallelization where you treat the matrix-vector multiplication as three independent vector dot products $A(1, :)m$, $A(2, :)m$, and $A(3, :)m$ using a `cilk_for` statement will not scale to more than 3 processors. Therefore, you might want to parallelize the dot products themselves using a divide-and-conquer approach.

For the last part, where you return the closest k stars, it is all right to use an existing sorting implementation in Cilk++ (check the `$cilkpath/examples/` folder). Your program will be executed as:

```
>> ./galaxymaster n inputfile k
```

The first parameter n denotes the number of stars in the galaxy. The inputfile will contain a quadruple on each line, representing the x, y, z coordinates followed by the mass of the star, with line i containing information about the ith star.

```
line 1:    x1  y1  z1  mass1
line 2:    x2  y2  z2  mass2
line 3:    x3  y3  z3  mass3
. . .
```

Each value is a double precision floating point number. The length of the inputfile is assumed to be consistent with the first parameter. The last parameter k denotes the number of closest stars to the center of the galaxy that you need to return. Your program should print out the center of galaxy in one line, followed by the average distance in the second line. In the third line, you should print out k integer values for the indices of closest stars. An example is:

```
>> ./galaxymaster 100 stars.txt 5
Center of galaxy is:   (78.4, -12.5, 43.9)
Average distance to center is 4938.45
Closest 5 stars in order of distance are: 93, 45, 63 12, 32
Number of cores used: 4
Number of repeats: 100
Average time for computation: 0.21 seconds
```

You should run the program on the OnDemand cluster, using 1, 2, 3, and 4 cores. (Each single node of OnDemand has 4 cores, so you will just use one node.)

For the reported computation time, you should only measure computation, not the time to parse and read in the arrays. The time will probably be very small, so I suggest that your code actually time a (sequential) loop that repeats the code (but not the input and output) many times and divides by the number of runs.

Even after parallelizing the dot products using divide-and-conquer, there is a chance that your program will not enjoy linear speedup. What might be the reason? Do you think that the algorithm does not have sufficient parallelism, or do you think there is another bottleneck other than parallelism?

Hint: Using cilkscreen, you can empirically find out the parallelism of your code. However, you should remember that the tool estimates parallelism for the entire program, meaning that your possibly sequential input parsing will make your program look less parallel than it actually is. For an accurate parallelism evaluation using cilkscreen, you could just fill matrxix A and vector v with random dummy values on the fly, not doing any input, in an embarrassingly parallel way. You can use any random number generator. Report the parallelism report of cilkscreen on this variant (the one that doesn't read from the file) of your program for extra credit.

For more extra credit, try to find a way to list the k closest stars that is faster than sorting all of them by distance.

Turn in

- Your source code.
- A `Makefile` that compiles your code.
- A document that contains instructions for compiling and running your code, plus tables of your run time results on different numbers of cores (1-4), plus your cilkscreen results, plus a report that describes and interprets your results and any conclusions you draw from them, and also the names of all your team members.

You may do this assignment alone or in groups of two.