EECS 358 – Introduction to Parallel Computing
Homework 3 (100 points)

EECS 358
G. Memik

DUE: End of Day, Wednesday, June 8th

Problem 1 [20 pts.]

a) [10 pts.] In Slide 15.12 of the class notes, we have seen the runtime cost for Matrix Vector Multiplication on a hypercube and torus. Show step by step the analysis to find these cost estimates.

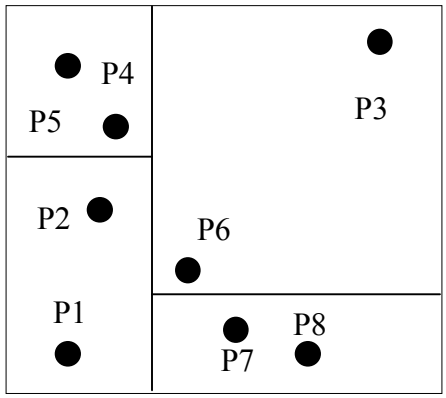
b) [10 pts.] Describe an efficient matrix transposition (Slides 15.7 – 15.10) for a ring communication structure with p processors and analyze its running time.

Problem 2 [80 pts.]

In this problem, you are asked to implement a parallel version of recursive bisection algorithm we have seen in the class. Particularly, you are asked to find a number of quadrants (this number will be given as an argument to the program) in a 2-dimensional coordinate system. A template file called *recursive_bisection.c* is provided in the tarball located here:

```
358smp.eecs.northwestern.edu:~memik/hw3.tar
```

The quadrants are rectangles of varying sizes. In the final result, each quadrant will contain equal number of points[1]. An example result is shown below:



Here, we have 4 quadrants of size 2. The coordinates for the points are generated by processor 0 and broadcast to all the processors using arrays x_axis and y_axis. The corresponding positions of this array are the x- and y-coordinates of the points. For example, point 997 is at the coordinate (x_axis[997], y_axis[997]). Note that there are a total of 524288 points in the

[1] Points that lie on the border of quadrants can be counted towards any of the bordering quadrants.

problem. Since the number of quadrants will be given as an argument, you can calculate the number of points in each quadrant.

Each partition (quadrant distribution) has a cost associated with it. This cost is the sum of all the Euclidian distances of points within a quadrant (or cluster to be precise). For example, for the quadrants in the above figure, the cost of the partitions is:

$$|\text{euc\_dist}(P5, P4)| + |\text{euc\_dist}(P3, P6)| + |\text{euc\_dist}(P1, P2)| + |\text{euc\_dist}(P7, P8)|$$

If there are more than 2 points in the quadrants, the cost is the sum of all distances from any point to any other. For example, if there are 3 points in a quadrant (P1, P2, P3), then the cost of that quadrant is

$$|\text{euc\_dist}(P1, P2)| + |\text{euc\_dist}(P1, P3)| + |\text{euc\_dist}(P2, P3)|$$

The partitions returned by your program should minimize this total cost (**points should be assigned to quadrants to yield a small total partition cost**).

Course Cluster: MPICH (*without* Condor)

Login to 358smp. Copy the corresponding files from the instructor's directory:

```
% cp ~memik/hw3.tar .
```

Untar the files

```
% tar –xvf hw3.tar
```

Set up the environment variables, without this you won't have access to compiler:

```
% source /etc/profile.d/modules.csh
% source /etc/profile.d/mpich-x86_64.csh
```

You need to do above step every time you log into the 358smp machine.

Now, you will use **mpicc** to compile your code:

```
% cd hw3
% mpicc –o executable recursive_bisection.c
```

Execute the binary created on 358smp, and it will run on course cluster 358-# (# = 1/2/3/4):

```
% mpiexec –launcher rsh -f machine_list_cluster -n 128
./executable 256
```

This will start the program (`executable`) on 128 processors (picked from `machine_list_cluster`) with an argument specifying `256` quadrants.

a) Implement the parallel recursive bisection algorithm. You are free to use whatever algorithm you like. You will be judged based on:

a) The quality of the output (valid quadrants, low partition cost, readability)
b) The general sequential/parallel run-time of your algorithm (including cost calculation)
c) Scalability (and readability) of your code

Your program must find quadrants, then print the total partition cost as well as the total execution time (including both time to find quadrants and time to compute partition cost) and the quadrant coordinates (you can print the coordinates of four corners of each quadrant).

In its current state, recursive_bisection generates random variables for two arrays (`X_axis` and `Y_axis`), broadcasts these values to the processors and calls the "`find_quadrants`" procedure. This is the procedure you will implement for this homework. Find_quadrants inputs and outputs are defined as follows:

Input:

`num_quadrants`: The number of quadrants that the procedure has to extract. This number is always a power of 2.

`X_axis`[]: The x-axis coordinates of the points

`Y_axis`[]: The y-axis coordinates of the points

Output:

The coordinates of num_quadrant quadrants

The cost of the quadrant distribution.

b) Run your program on the course cluster using MPICH with varying quadrant counts and thread counts. **Record** runtimes for **1, 2, 4, 8, 16, 32, 64 and 128 processors** on the cluster for **64, 128, and 256 quadrants**. You can change processor number by replacing N with **1, 2, 4, 8, 16, 32, 64 and 128:**

```
% mpiexec –launcher rsh -f machine_list_cluster -n N
./executable <arguments>
```

Upload on Canvas:

1. **A report describing your chosen algorithm, notes on the algorithm's scalability, and runtimes recorded for (b)**
2. **All of your code**

Good luck!