CS 140 Midterm 1 -- 8 February 2010

Name Perm#

Problem 1 [20 points total] In Lake Wobegon, all the women are strong, all the men are good-looking, and all the children are above average. Well, everyone can't be above average--but here we'll count how many are.

We have **n** kids and **p** processors. Each processor starts out with **n/p** elements of a vector **IQ** of the **n** kids' IQ values. Your goal is to compute the average of all **n** IQs (that is, their sum divided by **n**), and also to figure out how many of the **n** IQs are larger than average. The results, called **averageIQ** and **numHighIQ**, should end up on processor 0. For example, if the entries in **IQ[]** are 110, 90, 120, and 100, then the **averageIQ** is 420 / 4 = 105, and the **numHighIQ** is 2 (since two values, 110 and 120, are larger than average). A sequential algorithm to do this on one processor would be as follows. Note that **IQ[]** and **averageIQ** are doubles, not integers.

```
double sum = 0;
for (int i = 0; i < n; i++)
    sum += IQ[i];
double averageIQ = sum / n;
int numHighIQ = 0;
for (int i = 0; i < n; i++)
    if (IQ[i] > averageIQ) numHighIQ ++;
```

For this problem only, you don't have to worry about the efficiency of your code.

(1a) [10 points] Using pseudo-code, show how to do this in MPI using send and recv.

(1b) [10 points] Using pseudo-code, show how to do this in MPI using broadcast and reduce.

Name Perm#

Problem 2 [20 points total] This problem compares two different data layouts for matrix-vector multiplication on a message-passing machine. All **n** elements of a vector **x** are on processor 0. The elements of an **n**-by-**n** array **A** are divided evenly among **p** processors, with $\mathbf{n^2/p}$ elements per processor. The goal is to have all **n** elements of the product **A*x** end up on processor 0. For Algorithm 1, each processor has **n/p** rows of **A**. For Algorithm 2, each processor has a square block of **A** with **n/sqrt(p)** rows and **n/sqrt(p)** columns. Assume that **n** is divisible by **p**, and that **p** is a perfect square. You don't have to show the code for the two algorithms; just answer these questions.

(2a) [2½ points] Draw a clearly labeled diagram of the data layout for Algorithm 1.

(2b) [2½ points] Draw a clearly labeled diagram of the data layout for Algorithm 2.

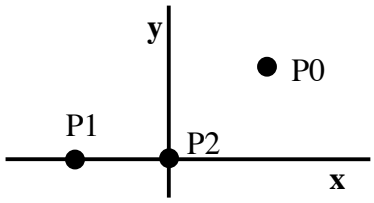
(2c) [15 points] Complete the following table with the parallel time $\mathbf{t_p}$, the span $\mathbf{t_\infty}$, and the total communication volume **v** for each algorithm. For **t**, we count only multiplication operations (which is why the work $\mathbf{t_1}$ is $\mathbf{n^2}$). You can omit lower-order terms in your answer, for example by writing $\mathbf{n^2}$ instead of something like $\mathbf{n^2 - n + 1}$.

	Work $\mathbf{t_1}$	**Parallel time** $\mathbf{t_p}$	**Span** $\mathbf{t_\infty}$	**Comm volume** **v**
Algorithm 1	$\mathbf{n^2}$	$\mathbf{n^2 / p}$		
Algorithm 2	$\mathbf{n^2}$			

Name Perm#

Problem 3 [20 points] Suppose you have **p** processors, **P(0)** through **P(p-1)**, each with local (double) variables **x**, **y**, and **d** (plus any other local variables you need). Each **(x, y)** represents a point in the plane, so each processor has one point. The goal is for each processor to set its own **d** to the shortest distance between its point and any other processor's point. For example, if there are three processors with points

P(0): x = 1, y = 1 **P(1): x = -1, y = 0** **P(2): x = 0, y = 0**



then **(1,1)**'s closest point is **(0,0)**, and **(-1,0)**'s closest point is **(0,0)**, and **(0,0)**'s closest point is **(-1,0)**, so the result should be

P(0): d = sqrt(2) **P(1): d = 1** **P(2): d = 1**

Write message-passing code (pseudocode is fine) to achieve this. Rules:

- Use *blocking* send and receive calls for all communication.
- Each processor **P(i)** should only send to / receive from its neighbors **P(i-1**) and **P(i+1)**, where we also include **P(p-1)** and **P(0)** as neighbors of each other.
- For full credit, your algorithm should use no more than **2p** rounds of message-passing, and should have parallel computation time $\mathbf{t_p = O(p)}$.

Hint: Send copies of all the processors' **(x, y)** values around a merry-go-round ring.

(Note: It's an interesting problem in computational geometry to do this in *less* than **O(p)** parallel time; but you don't have to do that for the exam problem.)

Name Perm#

Problem 4 [20 points total] You have a function called **findmax** that computes the largest element in an array of size $\mathbf{n = 2^k}$. The *serial version* of your code looks like the following:

```
double findmax(double * array, int n) {
        double max = array[0];
        for (int i = 1; i < n; i++)
                if (array[i] > max)  max = array[i];
        return max;
}
```

(4a) [10 points] Explain briefly why you can't parallelize this function in cilk++ by just changing the **for** loop to a **cilk_for**. Give a small example (say **n = 3** or **4**) of what can go wrong.

(4b) [10 points] (omitted from sample exam)

Problem 5 [20 points total] Short answer questions.

(5a) [10 points] What is the most important question in parallel algorithm design?

(5b) [10 points] A sequential program spends 99% of its time on a computation that could be done efficiently in parallel, and the other 1% on a computation that can't be parallelized at all. What can you say about maximum speedup for a parallel version of this program?