Problem 1 [20 points total] Each of \( p \) processors starts out with the coordinates \((x, y)\) of a single point in the plane. Our goal is to compute the center of gravity \((cx, cy)\) of the \( p \) points, and the average distance \( \text{avgdist} \) from the center of gravity to the points. The values of \( cx \), \( cy \), and \( \text{avgdist} \) should end up on processor 0. Here are the formulas:

\[
\begin{align*}
    cx &= \frac{x[0] + \ldots + x[p-1]}{p} \\
    cy &= \frac{y[0] + \ldots + y[p-1]}{p} \\
    \text{dist}[i] &= \sqrt{(x[i] - cx)^2 + (y[i] - cy)^2} \\
    \text{avgdist} &= \frac{\text{dist}[0] + \ldots + \text{dist}[p-1]}{p}
\end{align*}
\]

For example, if \( p = 3 \) and the points are \((0, 0)\), \((1, 2)\), and \((2, 1)\), then \((cx, cy)\) is \((1, 1)\), the distances are \(\sqrt{2}\), 1, and 1, and \(\text{avgdist}\) comes out to be \((2+\sqrt{2})/3\) or about 1.14.

(1a) [10 points] Using pseudo-code, show how to do this in MPI using \texttt{send} and \texttt{recv}. You don’t have to write a complete syntactically correct program, just the computations and MPI calls.

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

Problem 2 [28 points] A vector \( x \) of \( n \) doubles is divided evenly among \( p \) processors, each processor having \( n/p \) elements of \( x \). We want to end up with the sum of all the elements of \( x \) on processor \( P0 \), using only sends and receives to communicate. Here are three algorithms:

Algorithm 1: Each processor sends all its elements of \( x \) to \( P0 \), which then adds them up.

Algorithm 2: Each processor adds up its own \( n/p \) elements, then sends the result to \( P0 \), which adds up those sums.

Algorithm 3: Each processor adds up its own \( n/p \) elements. Then each odd-numbered processor \( P(k) \) sends its element to its even-numbered left neighbor \( P(k-1) \), which adds the received element to its own element. Then each of \( P2, P6, P10, \ldots \) sends its sum to the “divisible-by-4” processor to its left (\( P2 \) to \( P0 \), \( P6 \) to \( P4 \), \( P10 \) to \( P8 \), and so forth), which adds the received maximum to its own sum. This repeats with each processor \( P(8k+4) \) sending to \( P(8k) \), then \( P(16k+8) \) sending to \( P(16k) \), and so on, until finally the only receiving processor is \( P0 \).

We count computation time in terms of additions, so the time for the sequential algorithm on one processor is just \( t_1 = n - 1 \). We will ignore the difference between \( n \) and \( n - 1 \), and say that \( t_1 = n \).

Fill in the following table with the computation time \( t_p \) on \( p \) processors, the speedup \( s \), and the communication volume \( v \), always as a function of both \( n \) and \( p \). You can compute \( t_p \) as the maximum time over all the processors. You can ignore differences of plus or minus one. Two entries are filled in to start you off.
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Parallel time $t_p$</th>
<th>Speedup $s$</th>
<th>Comm volume $v$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm 1</td>
<td>n</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Algorithm 2</td>
<td></td>
<td>p</td>
<td></td>
</tr>
<tr>
<td>Algorithm 3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Problem 3** (This problem was about programming assignment 3, which was different in 2009 than this year; in 2010 we’ll ask questions about the n-body assignment instead.)

**Problem 4** [10 points] You have a function called `accumulate` that computes the sum of elements in an array of size $n = 2^k$. The *serial version* of your code looks like the following:

```c
double accumulate(double * array, int n) {
    double sum = 0;
    for (int i = 0; i < n; i++) {
        sum += array[i];
    }
    return sum;
}
```

Suppose that you need to parallelize this function using cilk++ in order to get better performance on your multicore desktop. Your friend tells you that simply replacing the `for` loop with the `cilk_for` keyword would work. Do you agree with him/her? Explain why or why not.

**Problem 5** [15 points total] The Magic Dornick algorithm (which I just made up) has two steps. The first step takes time $n^2$ on one processor, but it is embarrassingly parallel. The second step takes time $100*n$ on one processor, and there is no known way to do it in parallel. Answer the following questions about the Magic Dornick algorithm (ignoring communication time).

(5a) [5 points] What is the work as a function of $n$?

(5b) [5 points] What is the span as a function of $n$?

(5c) [5 points] Suppose $n = 1000$. If we are willing to buy as many processors as we want, what is the best speedup we can achieve?