CS 385 Lab 1

Revised Wed. 2/4/09 after lab.

In this lab, you'll be coding sequential versions of two "embarrassingly parallel" algorithms. Then you'll partition them, in preparation for parallelizing them in the future.

You'll turn in two C++ source files:

- *yourlastname*L1-seq.cpp
- *yourlastname*L1-par.cpp

Turn files in by copying them to ~srivoire/cs385/submit/ . You can verify the submission by visiting
http://rivoire.cs.sonoma.edu/cs385/lab1sub.txt

Building L1-seq

Generating and verifying the data

Both algorithms will use integer arrays as their inputs. You need to provide the infrastructure to randomly generate these arrays and to print them.

Include the following in your program:

- The preprocessor directive
  #define N 10
  (or any number you like). This is the array size.

- A function
  void GenerateArray(int array[], int size)
  The function needs to have exactly this name and these argument types, because I may replace it with something else when testing your code.

  This function will initialize the elements of array[] with random numbers. (It's OK to constrain the range of these random numbers in some way if it makes your life easier, but your code should all work for any array values.)

- A function
  void PrintArray(int array[], int size)
  This function needs to print the elements of your array in a format clear enough to allow you to test your code. Again, please use this function name and these argument types.

Test these functions by declaring an array in your main program, calling GenerateArray to initialize it, and calling PrintArray to print it.

IncrementAll

Write a function to produce a new array. The elements of this new array will be produced by adding 1 to the corresponding element of the old array, e.g.

```
new[0] = old[0] + 1;
```

It doesn't matter whether you return the output array from IncrementAll or whether you pass it to IncrementAll as a parameter. The important things are:

1. The output array doesn't overwrite the input array
2. The output array is accessible from main

Verify this function by printing the old and new arrays.

FindMaxElement

Write a function to find the value of the largest array element. Verify this function as well.

Building L1-par

Copy your L1-seq file to a new file (L1-par). You don't have to finish this file in lab.

In this part of the assignment, you will think about how to divide work among multiple processor cores. You'll simulate this by calling each function multiple times, once for each processor.

First, include a new #define statement:
```
#define P 8
```
This statement defines the number of processors. You should make no assumptions about the value of P, other than that it's less than N. 8 is a reasonable value to start with.

NOTE: The value of N=10 that we used for testing is ridiculously small. You should write your code with the assumption that N will usually be much larger than P (by a factor of 100 or more). Your code should be **correct** as long as N is larger than P. Your code should be **efficient** for the case where N is much larger than P.

IncrementAll

First, modify IncrementAll so that it will produce the correct output array if called in the following loop (the loop is in main).

```
for (int i=0; i<P; i++) {
        // Call IncAll – the interface is up to you
}
```

The idea is to envision each iteration of the loop being farmed out to a different processor, so that one processor handles the i=0 case, another processor handles i=1, and so on.

Each call to IncAll should thus result in N/P elements getting incremented (and don't forget to deal with the case where N is not a multiple of P). At the end of the for-loop, the resulting array should be exactly the same as the one from the sequential version.

FindMaxElement

You should use a similar method to parallelize FindMaxElement to the extent possible. Part of the task will not be fully parallelizable; it's OK to do that part sequentially inside main.