CS 385 Lab 3: Feb. 18, 2009

In this lab, you'll be using OpenMP to finish parallelizing your sequential code from Lab 1 and optimize the parallel performance of your code.

You'll turn in two files:

- *yourlastname*L3.cpp, your code
- *yourlastname*L3.txt, a writeup

Turn files in by copying them to ~srivoire/cs385/submit/ . You can verify the submission by visiting
http://rivoire.cs.sonoma.edu/cs385/lab3sub.txt

NOTE: Whenever you take a timing measurement, you should run the experiment **at least** 3 times and take the average time. You should also consider watching top to see if your program is getting its fair share of system resources. The other processes/users on the system and the OS's scheduling decisions will make your results inconsistent from one run to the next.

You should feel free to run the timing experiments on a home computer and report the results.

Parallelizing FindMax

Copy your code from Lab 2 to a new file. You will use OpenMP pragmas inside the FindMax function to parallelize this task.

Parallelizing this task is trickier than parallelizing IncAll because it requires communication between threads to determine the final answer. Here is one way to parallelize it:

```
#pragma omp parallel for
for (int i = 0; i < 1000; i++) {
     if( data[i]>largest ) {
#pragma omp critical(c)
          {
               if( data[i] > largest ) largest=data[i];
          }
     }
}
```

This approach has one major disadvantage; be sure that your code avoids this disadvantage.

In your writeup, answer this question (and number it):

Q1. What is the main disadvantage of this approach? Explain how your code is an improvement on this approach.

Make sure that your code works correctly for different values of `N` and `NUM_THREADS`.

Analyzing parallel performance

First, you need to insert timing routines into your code to measure the performance of `IncAll` and `FindMax`. Here's what you need to do:

1. Include the `<ctime>` library
2. Declare two new variables of type `time_t`, and one new variable of type `double`:
   ```
   time_t start, end;
   double diff;
   ```
3. Surround your `IncAll` code with
   ```
   time(&start);
   ```
   at the beginning, and
   ```
   time(&end);
   ```
   at the end. It doesn't matter exactly where you place these two calls, as long as they capture the incrementing operation and as little unrelated activity as possible.
4. Just after the second call, add the lines:
   ```
   diff = difftime (end,start);
   printf ("Increment ran in %f seconds.\n", diff);
   ```
5. Repeat steps 3 and 4 for your `FindMax` code.

Compare the performance of `IncAll` and `FindMax` with 1 thread vs. 2 threads. Try a wide range of values of `N` (remember that you may need to make your loop counters `long long` variables for very large `N`) and show the results in your writeup.

Q2. Show the results of this experiment, including the values of `N` you tried. Why do you think you got these results?

Now experiment with the directives you learned about in class, in particular:

- `shared/private/firstprivate`
- `schedule` (`static/dynamic/guided/runtime` and adjusting chunk size)

This is a helpful reference page:
https://computing.llnl.gov/tutorials/openMP/

Q3. Report the performance results of these experiments. Were you able to improve on the default settings? Explain your results.