CS 385 Lab 2: Feb. 11, 2009

In this lab, you'll be using OpenMP to parallelize your sequential code from Lab 1 in two different ways.

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

- *yourlastname*L2-test.cpp
- *yourlastname*L2-task.cpp
- *yourlastname*L2-data.cpp

Turn files in by copying them to ~srivoire/cs385/submit/ . You can verify the submission by visiting
http://rivoire.cs.sonoma.edu/cs385/lab2sub.txt

Building L2-test

Please work in pairs for this part of the assignment in order to minimize resource contention on cwolf. In your initial comment, list the names of both students who worked on the assignment.

Before you parallelize your code from last time, you'll do a test run to be sure that

- The correct number of threads are being created
- Threads are being assigned to both of cwolf's processors.

Log in to cwolf, start a new C++ file, and add the following:

At the top of the file:
```
#include <omp.h>
#include <stdio.h>
#define NUM_THREADS 2
```

At the beginning of `main`*:*
```
int i;
omp_set_num_threads(NUM_THREADS);
```

Following those lines:
```
#pragma omp parallel for
for (i=0; i<20000; i++) {
     int j, k;
     for (j=0; j<20000; j++)
          for (k=0; k<20000; k++);
     printf("Thread id: %i (%i, %i, %i)\n",
          omp_get_thread_num(), i, j, k);
}
```

Note that there is a semicolon after the k loop! This is not a typo; the only purpose of the j and k nested loops is to slow your program down enough for you to observe it in action.

Compile your program using g++ with the –fopenmp flag. *Don't run it yet.*

Open up another Terminal window and log into cwolf. Enter the top command. Top shows you the processes that are currently taking the most CPU time. There are a few header lines and then a table; each line corresponds to a process. Notice the `%CPU` column about 2/3 of the way across the line. This shows the percentage of CPU time taken by each process. The maximum is actually not 100%; it's 100% multiplied by the number of processors. On cwolf, which has 2 processors, the maximum is 200%.

You can verify that cwolf has 2 processors by typing:

```
cat /proc/cpuinfo | grep processor | wc -l
```

Be sure that you have located the %CPU column before you execute your program.

You are going to verify that your program uses both CPUs. In order to do this, we should avoid having everybody execute their programs at once! Looking at top, make sure that nobody else is running a CPU-intensive program before you proceed.

When top shows that it's OK, execute your program and verify the following:

- Your program prints out thread IDs 0 and 1
- Top reports a CPU usage close to 200% for your program.

Once you have verified these two things, press Ctrl-C to exit your program.

Change `NUM_THREADS` to 4 and verify that your program prints out 4 thread IDs. Please press Ctrl-C as soon as you verify this. Then change the number of threads back to 2.

Now change your program as follows:

- Change the line declaring i to
  ```
  int i,j,k;
  ```
- Remove the declaration of j and k later in the program.

In your initial comment, state what the problem with this change is. Running the program and comparing the output to the previous output will help, but the problem may not be obvious. You may have to wait a little bit longer to see output from your program.

Correct the program while still declaring i, j, and k at the beginning of the program.

Your program is ready for submission if it meets the following criteria:

- Initial comment with your name and your partner's name, plus an explanation of what's wrong with the initial change to `int i, j, k`
- A correction to the problem with the initialization of j and k
- `NUM_THREADS` set equal to 2
- Spawns two threads and uses both processors

L2-task

Work on this part individually.

Copy your sequential code from Lab 1 to a new file. You will parallelize this code by assigning the increment task to one thread and the find-max task to another.

Add the following to your program:

- Include the OpenMP header file
- #define NUM_THREADS 2
- At the beginning of main, create a new array:
  bool spawned[NUM_THREADS]
- Initialize each element of this array to false.
- At the beginning of main, add
  omp_set_num_threads(NUM_THREADS);

Use the
#pragma omp parallel sections
and
#pragma omp section

directives to assign IncrementAll to one thread and FindMaxElement to the other. It's OK if this part of the program simply calls the functions you have already written.

Inside each parallel section, add the line:
spawned[omp_get_thread_num()] = true;

Be sure that the input array is declared and defined *before* the parallel code. Also be sure that the output array and the "max" variable are declared and defined before the parallel code.

You should not be printing anything out in the IncrementAll function, the FindMaxElement function, or in the parallel block of code.

After the parallel block of code, print out the following:

- The values of each element of spawned, separated by tabs
- The values of each element of the old array, separated by tabs
- The values of each element of the incremented array, separated by tabs
- The value of the maximum element of the array

You may use either cout or printf.

Verify that both elements of spawned are 1 and that the calculated values are correct, and then submit your program.

L2-data

Work on this part individually.

Modify your code to exploit data parallelism rather than task parallelism by using

```
#pragma omp parallel for
```

to parallelize the iterations of IncrementAll. Use the same spawned variable to make sure that two threads are spawned. After the parallel part of the program, print out the values of spawned and of the old and new arrays.

You don't need to use FindMax in this program; that comes later ☺

Make sure that your code works for varying numbers of threads.