Prof. Jingke Li (FAB 120-06, lij@pdx.edu); Class: TTh 10:00-11:20 @ UTS 205; Lab: W 12:30-13:50, 14:00-15:20.

Assignment 2: Programming with OpenMP
(Due Wednesday, 2/15/17)

This assignment is to practice multi-threaded programming using OpenMP (version 3.0). Download the file `assign2.zip` to your CS Linux lab account and unzip it. You'll see two program files, `mmul.c` and `qsort.c`, for matrix-multiplication and quicksort, respectively. Your task is to convert these programs into OpenMP programs.

For this assignment, both CS415 and CS515 students will do the same programming work. However, CS515 students will need to include an extra piece in their summary report (see below). This assignment carries a total of 10 points.

Matrix Multiplication

The file `mmul.c` contains a simple matrix multiplication program, for multiplying two specific 8×8 matrices. Your tasks are:

1. Write three versions of OpenMP programs, `mmul-omp1.c`, `mmul-omp2.c`, and `mmul-omp3.c` to parallelize (1) the outer loop, (2) the inner loop, and (3) both loops, respectively of the multiplication loop-nest. (There is no need to parallelize the initialization loop.)
2. In all these programs, insert a `printf` statement in the loop-nest to print out the element's index `[i,j]` and the corresponding thread's id.
3. Insert proper statements to collect work-distribution statistics, *i.e.* the number of array elements computed by each thread. At the end of the program, print out the result in the following form:

   ```
   [ 0]: 6, [ 1]: 6, [ 2]: 6, [ 3]: 6, [ 4]: 5, [ 5]: 5,
   [ 6]: 5, [ 7]: 5, [ 8]: 5, [ 9]: 5, [10]: 5, [11]: 5,
   Total: 12 threads, 64 elements
   ```

   where the numbers inside the brackets are thread ids, and the numbers outside are the array element counts. (Space formatting is not required.)

   Note: Other than OpenMP directives and the above inserted statements, you should not change the original program.
4. Use OpenMP's environment variable to configure the number of threads to 4, 8, and 12. Run the three OpenMP programs under each configuration and save the statistics results.

Quicksort

The file `qsort.c` contains a sequential quicksort program. (This is the same version as you've seen in Assignment 1.) It takes a command-line argument, `N`, representing the size of the array.

Your task is to convert this program into an OpenMP program by inserting proper directives. Call the new program `qsort-omp.c`. You should not change any existing statement in the program, except for the following:

- The `qsort-omp.c` has a different user interface from `qsort.c` — it could take an additional second command-line argument, `numThreads`, representing the number of threads to use. If this argument is

not provided, a default value of 1 is used. Modify the input section of the program to accommodate this change.

- Insert a `printf` statement in the function `bubblesort()` to report the array range being sorted and the corresponding thread's id. Insert another `printf` statement in the function `quicksort()` to report the middle element returned by the `partition()` routine and the corresponding thread's id. The following is a sample print out:

```
Thread 1 found middle [3]
Thread 0 sorted [0, 2]
Thread 1 found middle [39]
Thread 3 found middle [7]
Thread 2 sorted [4, 6]
...
```

- Like in the matrix multiplication program, insert proper statements to collect work-distribution statistics, *i.e.* the number of array elements sorted by each thread. (Don't forget the middle elements!) At the end of the program, print out the result in the same format, *i.e.*

```
[ 0]:13, [ 1]:11, [ 2]:12, [ 3]: 4,
Total: 4 threads, 40 elements
```

Run your `qsort-omp` program with `N = 50` and number of threads set to 4, 8, and 12, respectively. Save the outputs from these runs.

Report

Create a report to include

- The statistics results from the three OpenMP matrix multiplication programs under the three configurations. (Don't include other print outs from these programs, since it would make the report too long.)
- The complete print-outs from the OpenMP quicksort program under the three configurations.

[**CS515 Students**] Include an analysis over the results in the report. Discuss whether you can observe full parallelism and balanced workload in each program's run. If there is anything unusual or unexpected, speculate what the cause might be. You could include other discussions or comments that you see fit.

Submission

Make a `zip` file containing your four OpenMP programs and the report. Use the Dropbox on the D2L site to submit your assignment file.