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.

Lab 3: Programming with OpenMP

Download and unzip the file `lab3.zip` from D2L. You'll see a `lab3` directory with some program files.

Exercise 1: Hello World

The file `hello-omp.c` contains a simple OpenMP version of a hello-world program.

1. Take a look at the program. How many copies of "Hello world!" do you think the program will print when executed? Compile and run the program. Is your guess right?

   ```
   linux> make hello-omp
   linux> ./hello-omp
   ```

2. Try to change the number of "Hello world!" printouts without changing the program itself. Verify your result by running the program again.

3. Now make changes within the program to achieve the same effect. Very your result by compiling and running the modified program.

4. Find out which of the above two thread-controlling approaches has the priority. (*i.e.* If you give conflicting directives through these approaches, which one will prevail?)

5. Finally, add a call to `omp_get_thread_num()` to the `printf` routine, so that you can see where each of the "Hello world!" is printed from.

Exercise 2: Loop Nest

The file `loop.c` contains a program with a simple double loop. Take a look of the program, then compile and run it. Pay attention to the result value; you want the parallel versions to produce the same result.

1. The OpenMP program in `loop-omp1.c` is an attempt at parallelizing the outer loop of the double loop-nest. Try to compile and run it. Does it work? If not, figure out the problem and fix it.

2. Similarly, the OpenMP program in `loop-omp2.c` is an attempt at parallelizing the inner loop of the double loop-nest. Try to compile and run it. Does it work? If not, figure out the problem and fix it.

3. Write a third OpenMP program in `loop-omp3.c` to parallelize *both* loops in the double loop-nest.

Exercise 3: Recursive Routine

The file `rec.c` contains a program with a simple recursive function. Take a look of the program, then compile and run it. Remember its output.

1. Two OpenMP programs, `rec-omp1.c` and `rec-omp2.c`, try to parallelize the recursive function. Try to compile and run them. Do they work? If not, can you figure out why.

2. Write a third OpenMP program in `rec-omp3.c` to parallelize the recursive function so that each array element is printed out by a different thread.

Exercise 4: Lock Ownership

The file `lock-omp.c` contains the code from a slide of this week's lecture. Try yourself to confirm that indeed a lock locked by a thread A can be unlocked by a different thread B.

Exercise 5: Nested Parallelism

The file `nested-omp.c` contains a program with a case of nested parallel regions.

1. Compile and run the program. How many lines of output do you see? What do you think happened to each of the two `parallel` directives?
2. Now set the environment variable `OMP_NESTED` to true. Run the program again. Do you see a different output?