Prof. Jingke Li (lij@pdx.edu); Classes/Labs: TR 1400-1550 @FAB 150, 88-10; Office Hours: TR 1300-1400 @FAB 120-06

Lab 3: Programming with OpenMP

1 Hello World (`hello-omp.c`)

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 through two difference channels: (1) environment variables, (2) in-program directives. Run the program again to verify. If these two channels give conflicting directives, *e.g.* one says 4 and the other says 6, find out which one prevails.

3. Modify the print statement so that you can see the thread and CPU core info with each message, *e.g.*

   ```
   Hollo world! -- thread 0 on core 3
   ```

2 Loop Nest (`loop-omp*.c`)

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 two OpenMP programs in `loop-omp1.c` and `loop-omp2.c` are attempts at parallelizing each of the two loops in the double loop-nest. Compile these programs and run each multiple times. Do they always work? If not, figure out the problem and fix it.

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

3 Recursive Routines (`rec-omp*.c`)

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 you see expected parallelism? If not, can you figure out why.

2. Write a third OpenMP program in `rec-omp3.c` to parallelize the recursive function so that full parallelism is realized.

4 Lock Ownership (`lock-omp.c`)

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.

5 Nested Parallelism (`nested-omp.c`)

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?

6 Task Parallelism (`bank-omp.c`)

The file `bank.c` contains a sequential program performing simple deposit and withdraw operations on a bank account. If run, all deposit actions occur before any withdraw action.

1. Insert OpenMP directives to the code so that the deposit and withdraw actions will be concurrent. Try two different versions: (a) with the `sections` directive, and (b) with the `task` directive. Verify that the resulting program for each case behaves as expected.
2. Is there a need for synchronization? How would you handle it? Implement your solution.

7 Submission

Write a short report to summarize your experience with this lab. Zip your write-up with your `bank-omp.c` program into a single zip file, and upload it through the “File Upload” tab in the “Lab 3” folder on Canvas. The submission deadline is the end of tomorrow (Friday).