Prof. Jingke Li (FAB 120-06, li@cs.pdx.edu); Classes: TTh 10:00-11:50 @ FAB 40-06; Office Hours: TTh 12:00-13:00.

Programming Project 2
(Due Thursday, 5/10/12)

This project is to practice multi-threaded programming with the use of OpenMP (version 3.0). On the D2L class website, you'll find two programs, `prime_seq.c` and `qsort_seq.c`. They implement the sequential prime-finding and quicksort algorithms, respectively. (`qsort_seq.c` is the same program you used in Project 1.) Your task is to insert OpenMP directives and runtime routines to make them run as parallel programs. Rename the modified programs to, `prime_omp.c` and `qsort_omp.c`. In addition, you need to collect timing results from these programs, and tune your programs to achieve some speedup.

Program Modification

For each program, make sure you fully understand the algorithm it implements. Study to see where parallelism may be introduced. The next step is to decide what OpenMP directives to use. Two quick hints: (1) you always need a `parallel` directive; and (2) you always need to specify the number of threads to use (either through the environment variable `OMP_NUM_THREADS` or through the `num_threads` clause).

Program Compilation and Execution

While you may develop your programs somewhere else, you need to compile and run them on the specific Sun multi-core server, `cepheus.cs.pdx.edu`. On this machine (and other CS solaris machines), there are two versions of `gcc`. It is important that you use the correct one: `/opt/csw/bin/gcc` (which is version 4.6.3). To compile an OpenMP program, use the `"-fopenmp"` flag.

Once you have compiled your modified programs, you need to test them to make sure they produce correct results and use concurrent threads. (*Hint:* Insert debugging/printing code to verify results and to echo a message from each thread with `tid` showing.)

Timing Measurements

Use the simple Unix timing routine `time` to conduct a rudimentary timing study, e.g.

```
solaris> time prime_omp 1000
0.02u 0.07s 0:00.52 17.3
```

The first two numbers are accumulative user and system CPU times (across all cores); the third number is the real (elapsed) time; and the last is the ratio of the combined CPU time and real time (as a percentage). For our timing study, you should use the real time.

For each program, collect timing results for different thread-number and array-size settings. For thread numbers, cover the following cases: 1, 2, 4, 8, 16, 32, 64, and 128. For array sizes, cover at least four large sizes. (For `prime`, include at least sizes 10^7 and 10^8; and for `qsort`, include at least sizes 10^6 and 10^7.) (*Hints:* Before making timing runs, you need to comment out all print statements in the program. Also, run a program multiple times for the parameter settings you want to collect timing data on. You may use the best of the runs to report timing.)

Compare timing results with that from the sequential programs. Do you see speedup or slowdown? If you don't see any speedup, you need to tune your programs to get better performance.

Comparison with Pthread Version (CS515 students only)

Compile and run the Pthread version of the quicksort program you wrote for Project 1. Collect timing data for the same thread-number and array-size settings as you did for your OpenMP version. Compare the results of the two versions.

Requirements

You must tune your programs to achieve some speedup against the sequential programs for large array sizes. Report timing results. Use nice tables to tabulate them. Summarize your project. Speculate reasons for the performance. Include any lessons you learned and any conclusions you want to draw. 40% of points are to be assigned to the summary.

What to Turn In

Make a `zip` or `tar` file containing your two OpenMP programs and the summary report. Use the Dropbox on the D2L site to submit your project file.