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

# Lab 4: More Shared-Memory Programming

In this lab, you are to practice instrumenting programs to make them suitable for performance studies. There are typically two types of instrumentation, (1) inserting timing routines, and (2) making program's parameters changeable at the execution time. (You've done the latter in Assignment 1.)

## 1 A Base Version

Take a look at the simple program in `sum.c`. It computes the sum of `sqrt(i*i)` for 1000 `i` values. Now make the following changes:

1. Make the parameter `N` a command-line argument, and remove the default value of 1000. If a user does not provide a value for `N`, the program should prompt the user, and exit:

```
linux> ./sum
Usage: ./sum N
linux> ./sum 1000
Sum of 100 sqrt(i*i) is 4950
```

2. Insert timing routines to measure the elapsed time of this program. You should leave the `printf` statement outside of the timing brackets. Also, report the timing in units of millisecond:

```
linux> ./sum 10000
Sum of 10000 sqrt(i*i) is 49995000
Seq-version elapsed time:     1.76 ms
```

   In last week's lecture, we introduced such routines for both C and C++ programs. See the provided files, `timing.c` and `timing2.cpp`, for more information. (*Hint:* For this program, the `gettimeofday()` routine is adequate.)

## 2 OpenMP Version

Convert the modified version of `sum.c` into an OpenMP program, and save it in `sum-omp.c`. Insert directives to parallelize the loop in the program. You should use the `reduction` clause to handle the summation.

Further modify the program to take an optional, second commend-line argument, `P`. Use `P` to set the number of threads to use in the parallel region. If a user does not provide a value for `P`, use a default value of 1:

```
linux> ./sum-omp
Usage: ./sum-omp N [P]
linux> ./sum-omp 10000
Sum of 10000 sqrt(i*i) is 49995000
OMP-version (1 threads) elapsed time:     1.48 ms
linux> ./sum-omp 10000 2
Sum of 10000 sqrt(i*i) is 49995000
OMP-version (2 threads) elapsed time:     1.14 ms
```

*Hint:* You may want to insert some print statements to check that there are indeed multiple threads working in the parallel region. Before timing measurement, you should remove these debugging statements or placed them inside guarded sections, which can be controlled by compiler flag setting (e.g. `gcc -DDEBUG`):

```
#ifdef DEBUG
    printf("some debugging message\n");
#endif
```

## 3 C++ Multi-Thread Version

Now convert the modified version of `sum.c` into a C++ multi-thread program, and save it in `sum-par.cpp`. Keep the same user parameter interface as that of the OpenMP version, i.e. requiring the user to provide a value for `N` and maybe a value for `P`.

There are multiple tasks involved in this conversion.

- You need to convert C header files and some statements (e.g. `printf`) into corresponding C++ ones.
- You need to create `P` worker threads, and evenly partition the workload (i.e. the loop range) over them. More specifically, the worker routine needs to use its thread id to compute the loop section that it's responsible for. (You should print out this info to verify it is correct.)
- You need to use some form of access protection for the global `sum` variable. (*Hint:* C++'s `atomic` variable is a good choice.)
- You need to use a different set of timing routines.

Here is a sample run of this program:

```
linux> ./sum-par
Usage: ./sum-par N [P]
linux> ./sum-par 1000 4
Worker[0] on range [0..250)
Worker[1] on range [250..500)
Worker[2] on range [500..750)
Worker[3] on range [750..1000)
Sum of 1000 sqrt(i*i) is 499500
C++-version (4 threads) elapsed time: 3.3878 ms
```

## 4 Timing Study

Collect timing data for different value combinations of `N` an `P` from these three programs. Tabulate and analyze the results.

- Do you observe any performance improvement from the parallel versions over the sequential version?
- Do you observe any speedup from each parallel version? If so, how good are the speedup trends?
- Do you observe performance differences between the two parallel versions?

For any observed behavior, try to explain the reason.

## 5 Submission

Write a short report covering your timing study. Submit it with your three programs through the "Lab 4" folder on Canvas. The submission deadline is the end of tomorrow (Friday).