Parallel Computing Lab Documentation

Release 1.0

Lei Huang

Apr 22, 2022

Contents:

It is the codig repository for COMP 4073 Parallel Computing in the Computer Science Department, PVAMU. The goal of these coding labs is to study parallel computing programming usig OpenMP. In each lab, there is a sequential C++ code, and the student needs to write a corresponding parallel program using OpenMP.

CHAPTER 1

Prerequisites

You need Linux or Mac OS to build and run these labs.

- Linux (Ubuntu)
- GNU compiler (gcc and g++)
- make
- git
- editor (Atom, Notepad++, nano, vim, emacs, gedit, etc.)

1.1 Parallel Computing Labs

1.1.1 Parallel Computing Lab 1

The lab 1 is to practice the first parallel construct in OpenMP that creates a parallel region in a C++ code. It is a parallel version of Hello World.:

```
cd labs/lab1
make
make run
```

You will observe that the parallel version hello_omp.cpp prints Hello World with random order. It is because a parallel code runs with multiple threads with an arbitrary execution order.

1.1.2 Parallel Computing Lab 2

The lab 2 is to practice how to write a SPMD (Single Instruction Multiple Data) parallel program in OpenMP. The lab computes the sum of a billion numbers.:

```
cd labs/lab2
make
make run
```

You need to revise the loop_omp.cpp file by adding OpenMP parallel region inside.

In this lab, the total number of loop iterations is max=1,000,000,000. You need to split the work and distribute them to all available threads as fair as possible. Think about the workload is 1 billion and the total number of threads, each thread should perform

```
long length =  max / (the number of threads);
```

OpenMP provides the runtime functions to tell you how many threads you have, and assign a unique thread number to each thread.

```
int threads = omp_get_num_threads();
int myID = omp_get_thread_num();
```

Based on the number of threads and the each thread ID, you can compute the range of loop each thread needs to perform:

```
long start = myID * length;
long end = (myID+1) * length;
```

You need to take care of the non-divisible case, in which the last thread will do more or less work depending on the division of loop iterations and the number of threads. Now each thread will do a partial loop from start to end:

```
for (long i=start; i<end; i++)
    ....
```

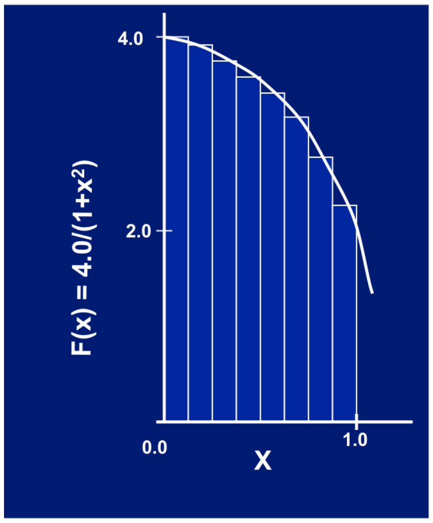
At the end, you need to summarize all threads' sum to get the final result. Keep in mind that each thread has its own partial sum at the end. You need to consider how to store them separately in memory. It may be a race condition if you summarize them in parallel without a protection.

1.1.3 Parallel Computing Lab 3

The lab 3 is to create a parallel program using OpenMP to calculate PI.:

```
cd labs/lab3
make
make run
```

You need to revise the pi_omp.cpp file by adding OpenMP parallel region inside.



You need to revise the program using three different ways to parallelize the code.

1. SPMD: calcuate the workload for each thread based on the thread ID.
2. OMP for with synchronization: use *omp for* to paralleize the loop, and then use the critical section to get the final result.
3. Reduction: use OpenMP reduciton to get the fastest andthe least revised code.

1.1.4 Parallel Computing Lab 4

The lab 4 is to create an OpenMP program to calculate themandelbrot set.:

```
cd labs/lab4
make
make run
```

You need to revise the mandel_omp.cpp file by adding OpenMP parallel region inside.

The correct answer should be 1.510659.

1.2 Parallel Computing Homework

1.2.1 Parallel Computing Homeowrk 1

A prime pair or twin prime is a prime number that has a prime gap of two, in other words, the difference between the two prime numbers are two, for example the twin prime pair (41, 43). You need to write an OpenMP program to find the total number of prime pairs between 2 and 50,000,000. Your grade will be not only determined by the correctness of the total number, but also depends on your program performance. Your program should print the number of prime pairs and the total execution time of your program. Report the speedup of your program using 1, 2, 4, and 8 threads respectively.

Build your code:

```
cd homework/hw1
make
```

Run your sequential version:

```
./prime
```

Run your OpenMP parallel version:

```
./prime_omp
```

Run both of them:

```
make run
```

You need to revise the prime.cpp and prime_omp.cpp files respectively to create a sequential program and an OpenMP parallel program to complete the homework.

The following is the grading percentage for evaluating your program.:

```
Correctness: 60%
Performance: 40%
```

Please submit all of your program source codes and a short report for performance observation.