Benjamin Kaiser
10-15-16
CSC 410 - Parallel Programming
Dr. Christer Karlsson
Assignment 1, Part 1 - Sieve

Description of the Program

The program is a very simple program. All that it does is run the sieve of Eratosthenes with three different versions of parallelization. The sieve of Eratosthenes is an algorithm which checks for prime numbers. The details are described later in the document.

Algorithms and Libraries

This program only uses one algorithm called the sieve of Eratosthenes. This algorithm checks the first number in the list and then moves through the list deleting/marking all multiples of this number as not false. It then goes to the next number that is not marked as non-prime and marks all multiples in the list of this number as non prime.

The OpenMP library is a C/C++ preprocessor library that tells the compiler to optimize the code to be able to run on multiple threads. Other than that, the only libraries used in the program are iomanip and iostream for the printing statements.

I also used the make compilation system which allows me to simply create a Makefile and give it the source files and targets that I wish. Then I can use simple commands to compile the program instead of long strings of compiler commands and flags.

Program Structure

The program consists of just a main function. In this main function are three different implementations of the above described algorithm. One is just a serial implementation which runs on a single core.

The second is this exact same code except that the OpenMP pre-processor command has been added to compile this code so that it has the ability to run in parallel. This particular implementation uses the static method of scheduling.

The third is again the same loop as the serial version except that the OpenMP pre-processor command has been added to compile this code so that it has the ability to run in parallel. This particular implementation uses the dynamic method of scheduling.

Before and after each for loop, a start and end time for each implementation is set. The start time for each implementation is then subtracted from the end time of each implementation. Since this time is in seconds, it is then multiplied by 1000 to convert it to milliseconds.

Compilation and Usage

To compile this program, there are two different ways to do it. They both use the Makefile.

1) The first way is to simply navigate to the folder that the source code is in and type "make". This will run two compilation statements, one for this program and one for the other program included as part of this assignment.
2) The second way to compile the program is to navigate to the folder that the source code resides and type "make prime". This will only one compilation command and that is the command for just this portion of the assignment.

To use this program, the user must simply navigate to the folder where the binary file is and type either "prime <number to check to>" or "./prime <number to check to>". Both of these will run the program. Output will then be printed to the screen.

Testing and Verification Process

Testing for the Prime portion of the assignment was done as follows. The 3 different implementations were put into a loop which was run 100 times using 1,000,000 as the input. The printing of answers was commented out in order to do this. I then visually inspected the printing of each timing statement and noted patterns. The following results were noted:

The serial version of the code took around 24 ms to complete. This was very consistent throughout the run except for the first run which took around 60 ms.

Static scheduling stayed consistent around 8 ms except for the first run which took around 15 ms.

Dynamic scheduling stayed consistent around 25 ms even for the first run.

In order to check with printing and without iteration which may change results because the compiler fixes things for the iteration. After running a few iterations manually this way, I noticed that the serial code runs around 58 ms, the static runs around 19 ms and the dynamic runs around 28-29 ms.

Timing was done using the omp_get_wtime() function which is provided by omp.h. This was also used on the serial portion of the code. The start and end times for each loop were recorded and then the start time was subtracted from the end time. This occurred in seconds and they were then multiplied by 1000 to convert into milliseconds.

Submitted Documents

Included with this assignment are several deliverables. These are listed as follows.

- Description documentation for the Circuit Satisfiability portion of the assignment
- Description documentation for the Prime portion of the assignment
- Circuitsat.cpp - source code for the Circuit Satisfiability portion of the assignment
- Sieve.cpp - source code for the Prime portion of the assignment
- Makefile - the make file which allows for typing “make” and compiling both portion of the assignments but also specific targets for each half of the assignment.