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

Description of the Program

The program is a very simple program. All that it does is simulate a circuit and this circuit is the one that Dr. Karlsson put in the assignment document. The program runs many different inputs through this circuit (all of the possible inputs) using three different versions of parallelization and records and prints timing information for all three versions.

Algorithms and Libraries

This program uses one algorithm and that is the one provided by Dr. Karlsson. It essentially takes an input of a short integer or any integer that fits within 16 bits. It then evaluates the circuit given those bits and if the input satisfies the circuit, it prints out the inputs and the id of the thread.

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 two functions. These are main and check_circuit. Check_circuit was written by Dr. Karlsson (it was provided by him at least). Main is the function that I wrote and it consists of three for loops which loop from 0 to one less than 65,536 which is what 2^16 is. Inside of the loop, check_circuit is called and is passed the thread number that the current iteration of the for loop is running on as well as the current index between 0 and 65,536.

Check_circuit is the software model of the circuit which is contained in one long boolean condition in the if-statement. If the condition is satisfied, then the input that allowed the circuit to work is printed to the screen.

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 circuitsat". 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 "circuitsat" or "./circuitsat". Both of these will run the program. Output will then be printed to the screen.

Testing and Verification Process

Testing for the circuit satisfiability problem was done as follows. The entire 3 implementations were put into a loop which would run each one a 100 times. The output was print out for each of these 100 iterations. I then visually inspected each timing output and made observations. Please note that this loop was removed for the submission for grading.

The following results were noted:

The serial time seemed to settle right around 2.7 ms on average, however it did vary from about 2.7 ms to up to 12 ms.

The static scheduling time seemed to settle around .5 ms on average but there was much more wildl oscillation and it went from .5 to 12 ms.

The dynamic scheduling time seemed to settle around 1.4 ms on average but like the static scheduling, there was wild oscillation and it went from around 1.4 to 4 ms.

For 100 iterations, each of the loops printed out the exact same answers which would mean that the exact same answers occurred 300 times. I believe that this is a good indication that the results were accurate because no race conditions occurred and caused non-determinist results.

This 100 iteration on each loop was repeated 5 times with the same results on each. Based on this fact, I would say that parallelizing this code definitely had some speedup but I would say that dynamic scheduling seems to be the most stable as it didn't have as wide of a range as the static scheduling did. This makes sense as well since if system or other program processes took a thread, the program would not have to wait for those processes to complete their allocated time before continuing. Instead, it would just hand a different thread the part of the code that it was originally going to allocate to the first thread.

In addition, I ran the program several times on a single iteration to just see what it would do. It is interesting to note that this changed the time results significantly. The serial took between 4 and 12 ms on average while the static took between 4 and 12 ms on average as well. The static and the serial appear to be comparable on timing with the static scheduler sometimes taking longer. Dynamic scheduling for the most part was still faster and was running between 4 and 9 ms. It occasionally took longer than the static and serial but not often.

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.