Naman Gangwani
nkg160030

CS 4301.001
March 11, 2018

Report for Pgm3

Problem Statement

Finding prime numbers can be a large task, especially when finding the prime numbers that go up to a really large number. Making use of multiple processes and cores can help in improving its performance. One way to specifically do this is by utilizing MPI to split work amongst multiple processes and the OpenMP API for C. MPI functions and OpenMP directives will be useful in improving upon a well-defined algorithm for finding prime numbers known as The Sieve of Eratosthenes.

Approach to Solution

MPI is the main tool utilized to assist in finding a solution to the problem, although OpenMP does help in parallelizing a single loop. Additionally, SLURM (Simple Linux Utility for Resource Management) and access to a Linux server on a supercomputer through an SSH may be essential. Since it is apparent that computing prime numbers up to a value of N that reaches 2^32 – 1 is quite resource intensive, the decision to split up the work amongst multiple processes and combining the solution from each process through MPI functions was made. Similarly, an OpenMP pragma directive was utilized in order to parallelize the loop that calculates primes within a given range. Combined together, MPI and OpenMP make a powerful team in finding prime numbers.

Solution Description

In this task, the code provided by Michael J. Quinn in *Parallel Programming in C with MPI and OpenMP* was modified merely by accounting for big integers through the usage of *long long* data types. Additionally, the OpenMP API was made use of on this program, and comments throughout the C source file and calculations for the Karp-Flatt Metric were added on.

When logged into a remotely hosted Linux server on a supercomputer through an SSH (with bridges.psc.edu being the recommended remote host), it is important to have the *Pgm3.c* and *Pgm3.sh* files in the current directory. When they are there, go into interact mode by typing in "interact" and pressing enter. From there, executing the bash script with the SLURM-specific command *sbatch* should be done with the command *sbatch Pgm3.sh*. A slurm file should be produced in the current directory from executing this bash script, and the content of that file will display the number of prime numbers between 0 and the inputted value from the command along with the statistics of the program that ran.

```
[namang@br018 ~]$ interact

A command prompt will appear when your session begins
"Ctrl+d" or "exit" will end your session

[namang@r001 ~]$ sbatch Pgm3.sh
Submitted batch job 2549148
[namang@r001 ~]$ ls
a.out  matmul.c  matmul.sh  Pgm3.c  Pgm3.sh  slurm-2285378.out  slurm-2549148.out  test.c
[namang@r001 ~]$ cat slurm-2549148.out
Compile
Execute
There are 105097565 prime numbers less than or equal to 2147483647
Total sequential execution time:   5.742020
Total parallel execution time:   0.717751
Karp-Flatt Metric:   0.125000
Done
[namang@r001 ~]$
```

The execution of the bash script and the results after calculating the prime numbers between 0 and $2^{31} - 1$.

When testing values of N for this program, it seemed as if Bridges was having trouble allocating enough resources for $2^{32} - 1$, however, $2^{31} - 1$ seemed to be working fine, as shown in the screenshot. Nevertheless, it is apparent that there is quite some improvement, as it was able to calculate the primes between 0 and $2^{31} - 1$ rather quickly (in approximately 6 seconds). Without parallelization and splitting of work amongst processes, the program would not even complete execution after 15 minutes. The Karp-Flatt metric value helps assure us that parallel overhead was accounted for throughout the MPI broadcasts and other sources of overhead such as process startup time and synchronization time, and the value from this execution of the program was important in assuring that the Sieve of Eratosthenes algorithm was as effective in constantly accounting for overhead as possible in an effort to measure speedup.