Boğaziçi University
CMPE 478 - Parallel Processing
Spring 2020
Prime Number Generator
Can Özturan

Baran Deniz Korkmaz - 2015400183

April 2, 2020

Contents

1 Introduction

1.1 Definition

In the project, we are asked to implement an OpenMP program which generates prime numbers up to a predefined number M. The program will be executed by 1,2,4 and 8 threads respectively, and the performance measures will be analyzed.

1.2 Solution

The prime number generation algorithm is predetermined and provided by the instructor. The primary objective is to convert the algorithm into a parallelizable program structure consisting of loops in terms of OpenMP specifications.

The program finds the prime numbers up to $\sqrt{M}$ sequentially. The remaining interval is divided among the threads as subportions according to OpenMP specifications, therefore the prime numbers in the interval [$\sqrt{M}$,M] is found parallel.

2 Compilation and Execution

2.1 Compilation

The program can be compiled in two ways which will be described below.

1. Make command can be used to compile by using the makefile provided.

```
>> make
```

2. The compilation commands can be given manually.

```
>> g++ -c 2015400183.cpp -o 2015400183.o -fopenmp
>> g++ 2015400183.o -o 2015400183 -fopenmp
```

2.2 Execution

The program takes 3 arguments of which 1 is optional described below: 1- M: The number which determines the upper bound for interval. 2- Chunk Size: The number which determines the chunk size. 3- –primes: This optional argument will enable program printing the calculated primes as standard output.

```
>> ./2015400183 [M] [CHUNK_SIZE] [--primes (Optional)]
```

IMPORTANT NOTES:
1- Make sure that M is greater than or equal to 10.
2- The last argument is optional.
3- You can also view README.MD file to find necessary information about compilation execution.

3 Input and Output

3.1 Input

As stated in the Section 2, the program takes 3 arguments of which 1 is optional. Below you can find 2 examples which also describes the general logic behind the input format:

1. The following run finds the prime numbers up to 10 millions where OpenMP uses chunk size of 100 as loop scheduling.

   ```
   >> ./2015400183 10000000 100
   ```

2. The following run prints the prime numbers found in addition to the previous one.

   ```
   >> ./2015400183 10000000 100 --primes
   ```

NOTE: In case of an invalid argument passing, the program will notify the user as follows:

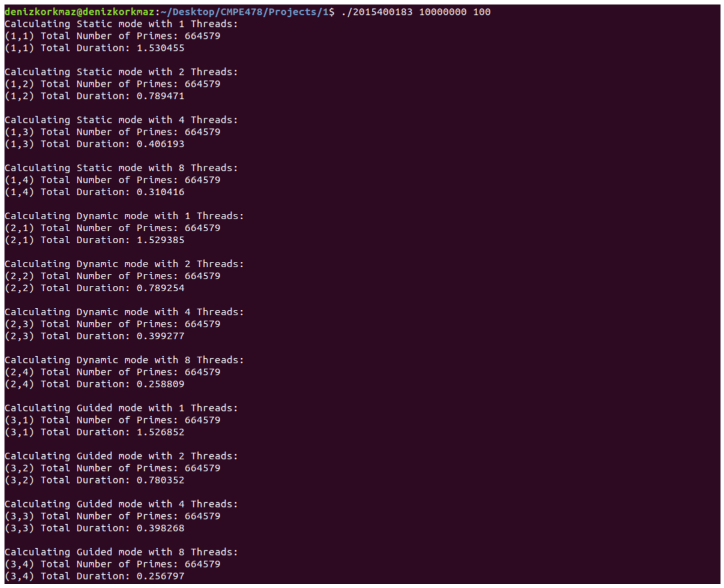
```
>> Please enter valid arguments: [M (>=10)] [CHUNK_SIZE] (Optional: [--primes])
```

3.2 Output

The program output is designed in a way that the user will be able to observe the progress of program. Therefore the user will be notified by the statements declaring which stage of the program is currently in progress within terminal.

After the program execution is completed, the analysis of performance measures will be written into the file named "2015400183.csv".
IMPORTANT NOTE: You should not have any other file named "2015400183.csv" in the directory you are running the program.

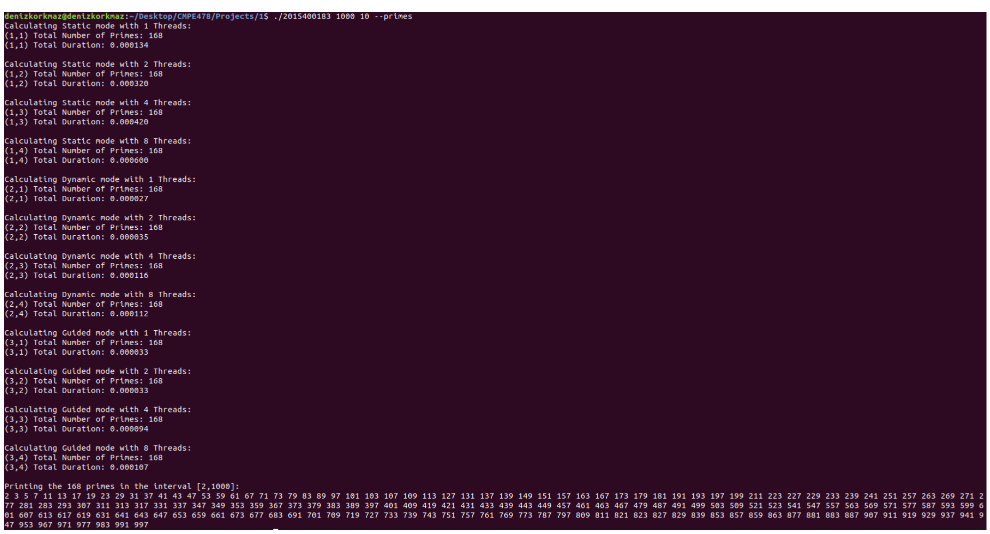
Below, you can find the general format for the output which is shown by running the first input stated in the previous section.



```
denizkorkmaz@denizkorkmaz:~/Desktop/CMPE478/Projects/1$ ./2015400183 10000000 100
Calculating Static mode with 1 Threads:
(1,1) Total Number of Primes: 664579
(1,1) Total Duration: 1.530455

Calculating Static mode with 2 Threads:
(1,2) Total Number of Primes: 664579
(1,2) Total Duration: 0.789471

Calculating Static mode with 4 Threads:
(1,3) Total Number of Primes: 664579
(1,3) Total Duration: 0.406193

Calculating Static mode with 8 Threads:
(1,4) Total Number of Primes: 664579
(1,4) Total Duration: 0.310416

Calculating Dynamic mode with 1 Threads:
(2,1) Total Number of Primes: 664579
(2,1) Total Duration: 1.529385

Calculating Dynamic mode with 2 Threads:
(2,2) Total Number of Primes: 664579
(2,2) Total Duration: 0.789254

Calculating Dynamic mode with 4 Threads:
(2,3) Total Number of Primes: 664579
(2,3) Total Duration: 0.399277

Calculating Dynamic mode with 8 Threads:
(2,4) Total Number of Primes: 664579
(2,4) Total Duration: 0.258809

Calculating Guided mode with 1 Threads:
(3,1) Total Number of Primes: 664579
(3,1) Total Duration: 1.526852

Calculating Guided mode with 2 Threads:
(3,2) Total Number of Primes: 664579
(3,2) Total Duration: 0.780352

Calculating Guided mode with 4 Threads:
(3,3) Total Number of Primes: 664579
(3,3) Total Duration: 0.398268

Calculating Guided mode with 8 Threads:
(3,4) Total Number of Primes: 664579
(3,4) Total Duration: 0.256797
```

As stated previously, the third optional argument enables program printing the calculated primes as standard output. Below, an example screenshot is attached after the run of following command:

```
>> ./2015400183 1000 10
```



```
denizkorkmaz@denizkorkmaz:~/Desktop/CMPE478/Projects/1$ ./2015400183 1000 10 --primes
Calculating Static mode with 1 Threads:
(1,1) Total Number of Primes: 168
(1,1) Total Duration: 0.000134

Calculating Static mode with 2 Threads:
(1,2) Total Number of Primes: 168
(1,2) Total Duration: 0.000320

Calculating Static mode with 4 Threads:
(1,3) Total Number of Primes: 168
(1,3) Total Duration: 0.000420

Calculating Static mode with 8 Threads:
(1,4) Total Number of Primes: 168
(1,4) Total Duration: 0.000600

Calculating Dynamic mode with 1 Threads:
(2,1) Total Number of Primes: 168
(2,1) Total Duration: 0.000027

Calculating Dynamic mode with 2 Threads:
(2,2) Total Number of Primes: 168
(2,2) Total Duration: 0.000035

Calculating Dynamic mode with 4 Threads:
(2,3) Total Number of Primes: 168
(2,3) Total Duration: 0.000116

Calculating Dynamic mode with 8 Threads:
(2,4) Total Number of Primes: 168
(2,4) Total Duration: 0.000112

Calculating Guided mode with 1 Threads:
(3,1) Total Number of Primes: 168
(3,1) Total Duration: 0.000033

Calculating Guided mode with 2 Threads:
(3,2) Total Number of Primes: 168
(3,2) Total Duration: 0.000033

Calculating Guided mode with 4 Threads:
(3,3) Total Number of Primes: 168
(3,3) Total Duration: 0.000094

Calculating Guided mode with 8 Threads:
(3,4) Total Number of Primes: 168
(3,4) Total Duration: 0.000107

Printing the 168 primes in the interval [2,1000]:
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199 211 223 227 229 233 239 241 251 257 263 269 271 2
77 281 283 293 307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409 419 421 431 433 439 443 449 457 461 463 467 479 487 491 499 503 509 521 523 541 547 557 563 569 571 577 587 593 599 6
01 607 613 617 619 631 641 643 647 653 659 661 673 677 683 691 701 709 719 727 733 739 743 751 757 761 769 773 787 797 809 811 821 823 827 829 839 853 857 859 863 877 881 883 887 907 911 919 929 937 941 9
47 953 967 971 977 983 991 997
```

4 Program Structure

As stated in Section 1.2, the program sequentially finds the prime numbers up to $\sqrt{M}$, and parallel computation will take place in the remaining interval $[\sqrt{M}, M]$.

The program consists of two helper functions which handles sequential and parallel computations respectively. This structures enables a simple analysis for performance measures, since we can simply compute the run-time durations of these functions.

Indeed, the sequential computation will be called just once per program which keeps the sequential computation time constant for every iterations within the program.

Finally, the performance measures will be written into the file "2015400183.csv"

5 Conclusion

Prime number generator algorithms presents a basis in computational science. Parallel computations of such algorithms conduces building fundamental skills for parallel processing.

Previously in CMPE 300, I have been assigned into a parallel processing project using OpenMPI - distributed memory parallelism. In this project, I think that I have gained an insight into the parallel computation of prime number generator algorithms using OpenMP - shared memory parallelism which is on the other side of the coin.