Funiba Neba

RW09258 – CMSC621 Project 1

1. Implementation
2. One word about Implementation includes the piece where it was suggested that a channel was set up between the dispatcher and consolidator so that the consolidator would know how many results it should be expecting. I was unable to figure out how to implement this in time, so I just used the option of using a goroutine with wg.Wait() then close(resultsQueue), to achieve the same effect.
3. Also note that for 1 worker you must put 0 as the input for M since the IDs are initialized at zero.
4. Also, each input must be separated by a space on the command line.
5. Project Report
6. What is the largest number of workers M that your implementation supports?
7. **Command prompt to generate 1MB data:** head -c 1048576 /dev/urandom > funibaNumbers.dat
8. **Command line arguments:** go run primes.go -M=#
9. The absolute number of workers my implementation supports is unclear. However, I started with a value of M=1, with the default values for N and C, and increased it all the way up to M=1,000,000. I didn’t keep going because it seems my program would continue to find the correct number of primes (3110) in a random 1MB data file regardless of M. But for sake of the following questions, I will say that my implementation supports M up to 1,000,000.
10. What is the least (expected) elapsed (wall) time for a random datafile of 1GB in your implementation?
11. The least elapsed time for my implementation to compute the primes in a 1GB random data file was 58.70 seconds. The inputs for time were M=10,000 ; N=65536 ; C=8192.
12. **Command line to generate 1 GB of data:** head -c 1073741824 /dev/urandom > funibaNumbers2.dat

**There are 3,094,190 primes in this file.**

1. **Command line arguments for iii:** go run primes.go -pathname=“funibaNumbers2.dat” -M=#
2. **Elapsed time (Default N & C):**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Workers | 1 | 1000 | 5000 | 10,000 | 50,000 | 100k | 500k | 1 mill |
| Time | 371.49 sec | 68.09 sec | 61.28 sec | 60.04 sec | 60.25 sec | 61.63 sec | 63.62 sec | 64.95 sec |

1. **Command line arguments for v:** go run primes.go -pathname=“funibaNumbers2.dat” -M=# -N=32768 -C=4096
2. **Elapsed time (N=32768, C=4096):**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Workers | 1 | 1000 | 5000 | 10,000 | 50,000 | 100k | 500k | 1 mill |
| Time | 376.63 sec | 64.03 sec | 60.26 sec | 58.96 sec | 59.34 sec | 58.79 sec | 62.70 sec | 63.07 sec |

1. **Command line arguments for vii:** go run primes.go -pathname=“funibaNumbers2.dat” -M=# -N=65536 -C=8192
2. **Elapsed time (N=65536, C=8192):**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Workers | 1 | 1000 | 5000 | 10,000 | 50,000 | 100k | 500k | 1 mill |
| Time | 374.35 sec | 62.60 sec | 58.86 sec | 58.70 sec | 59.50 sec | 58.66 sec | 62.79 sec | 62.23 sec |

1. **Command line arguments for ix:** go run primes.go -pathname=“funibaNumbers2.dat” -M=# -N=262144 -C=8192
2. **Elapsed time (N=262144, C=8192):**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Workers | 1 | 1000 | 5000 | 10,000 | 50,000 | 100k | 500k | 1 mill |
| Time | 372.95 sec | 62.32 sec | 59.71 sec | 59.10 sec | 59.57 sec | 61.43 sec | 61.37 sec | 62.00 sec |

1. **Command line arguments for xi:** go run primes.go -pathname=“funibaNumbers2.dat” -M=# -N=1048576 -C=8192
2. **Elapsed time (N=1048576, C=8192):**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Workers | 1 | 1000 | 5000 | 10,000 | 50,000 | 100k | 500k | 1 mill |
| Time | 360.09 sec | 61.89 sec | 59.97 sec | 58.73 sec | 60.54 sec | 64.88 sec | 70.65 sec | 70.70 sec |

1. **Command line arguments for xi:** go run primes.go -pathname=“funibaNumbers2.dat” -M=# -N=67108864 -C=8192
2. **Elapsed time (N=67108864, C=8192):**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Workers | 1 | 1000 | 5000 | 10,000 | 50,000 | 100k | 500k | 1 mill |
| Time | 370.15 sec | 75.54 sec | 65.51 sec | 64.26 sec | 66.10 sec | 67.57 sec | 70.54 sec | 69.45 sec |

1. What is the largest (random) datafile you can process within 3 mins (wall) elapsed time?

For this question, the values for N and C will be fixed at the value combination that produced the smallest elapsed time in question 2, **(N=65536, C=8192)**. This N and C combo gave 58.70 seconds with 10,000 workers and a 1GB file. I will test up to 3GB while changing the number of workers.

1. 1GB File:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Workers | 1 | 1000 | 5000 | 10,000 | 50,000 | 100k | 500k | 1 mill |
| Time | 374.35 sec | 62.60 sec | 58.86 sec | 58.70 sec | 59.50 sec | 58.66 sec | 62.79 sec | 62.23 sec |

1. 2GB File:

**Command line to generate 2 GB of data:** head -c 2147483648 /dev/urandom > funibaNumbers3.dat

**Command line to run primes.go with 2GB file:** go run primes.go -pathname=“funibaNumbers3.dat” -M=# -N=65536 -C=8192

**Number of Primes:** 6,195,902

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Workers | 1 | 1000 | 5000 | 10,000 | 50,000 | 100k | 500k | 1 mill |
| Time | -- | 119.25 | 112.11 sec | 111.01 sec | 114.49 sec | 114.20 sec | 118.32 sec | 120.54 sec |

1. 3GB File:

**Command line to generate 3 GB of data:** head -c 3221225472 /dev/urandom > funibaNumbers4.dat

**Command line to run primes.go with 2GB file:** go run primes.go -pathname=“funibaNumbers4.dat” -M=# -N=65536 -C=8192

**Number of Primes:** 9,295,058

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Workers | 1 | 1000 | 5000 | 10,000 | 50,000 | 100k | 500k | 1 mill |
| Time | -- | 181.34 sec | 171.36 sec | 169.74 sec | 170.33 sec | 170.67 sec | 174.69 sec | 174.34 sec |

1. Because I decided to increase the file size GB by GB the maximum size is 3GB. This means for the lowest time of 169.74 seconds, it took around 18.09 seconds to get through 1 MB of that 3GB. Using that to extrapolate, I could probably add on about 56% of another MB to 3GB, and using 10,000 workers with (N=65536, C=8192), successfully find the correct number of primes in less than or equal to 3 minutes.
2. How does the elapsed time change as M ranges from 1 to the maximum value in Q1, for the datafile and N, C parameter values in Q3?

For the constant N, C parameter values in Q3 the elapsed time followed the same parabolic pattern as the elapsed times in Q2. It would start out with 1 worker taking an obscene amount of time to calculate the number of primes, then the elapsed time would decrease as you increased M, then it would attain a local minimum at 10,000 workers. And then elapsed would gradually increase as you increase M to 1 million. Note, I did not include the times for 1 worker in Q3 because they would far exceed 3 minutes.