Muhammed Muktar
Project Report Multithreaded Application
March 3, 2024

The project design includes five functions:

checkFileOper: A helper function used to check file operation errors

dispatcher: A go routine function that reads into a datafile given a set N byte(s) or less to read into. The dispatcher will then create a job struct that consist of the path name, the start of the job in the file, and the length of the job that was read. The job struct is then inserted into a queue that will then be read by the worker later on. Once the dispatcher finishes it will indicate to close the queue of jobs so that the workers will know not to wait for any more jobs.

worker: A go routine function that reads into the queue of jobs. It will sleep for a random amount of time, then proceed as follows. The worker reads into the job's file path, reads into the start of the job in the file, and then read a set C byte(s) or less into the job. In every C byte(s) the worker will read 8 bytes into the job (unsigned 86 bit in little endian order) and determine if those 8 bytes are a prime number. The number of primes found will be tracked and once done it create a result struct with the job struct and the number of primes and insert the results into a result queue.

consolidator: A go routine function that will read into the queue of results given by the workers. The function will accumulate the total number of primes that all the workers have counter and send that result back to the main function.

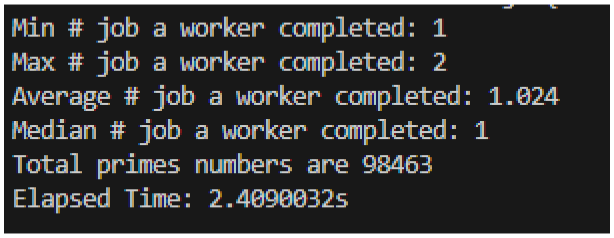
main: The main function handles the creation of all go routines: a dispatcher, a consolidator, and M worker threads. Once all the routines are finished the result from the consolidator thread will be given to the main.

To synchronize the worker threads a wait group is used. The main function will add 1 to the wait group for every worker routine created, each worker will then indicate that are done to remove themselves from the wait group once there are no more jobs in the job queue (which is made known when the dispatcher finishes). Once all worker routines are finished the result queue will close as there are no more jobs to work on (which lets the consolidator know when to stop looking for results). The program will eventually end when the consolidator thread finishes reading the results from the result queue.

Using a constant value for N = 64KB and C = 1KB

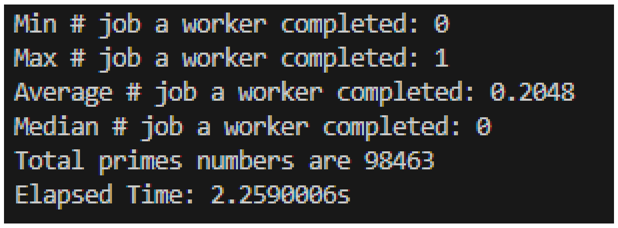
1. Largest number of workers M supports: 10000, at this stage I am not noticing any more improvements in the program with more workers.

   Using M workers (1000), N = 64KB, C = 1KB, and testFile64KB.dat of size 64KB. We see the result:

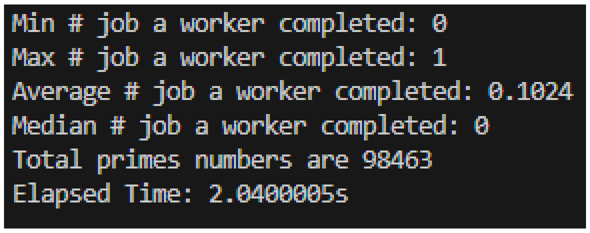
   ```
   Min # job a worker completed: 1
   Max # job a worker completed: 2
   Average # job a worker completed: 1.024
   Median # job a worker completed: 1
   Total primes numbers are 98463
   Elapsed Time: 2.4090032s
   ```

   Using M workers (5000), N = 64KB, C = 1KB, and testFile64KB.dat of size 64KB. We see the result:

   ```
   Min # job a worker completed: 0
   Max # job a worker completed: 1
   Average # job a worker completed: 0.2048
   Median # job a worker completed: 0
   Total primes numbers are 98463
   Elapsed Time: 2.2590006s
   ```

   Using all M workers (10000), N = 64KB, C = 1KB, and testFile64KB.dat of size 64KB. We see the result:

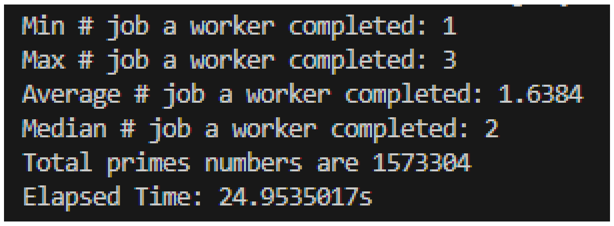
   

   ```
   Min # job a worker completed: 0
   Max # job a worker completed: 1
   Average # job a worker completed: 0.1024
   Median # job a worker completed: 0
   Total primes numbers are 98463
   Elapsed Time: 2.0400005s
   ```

   Once we go over 10000 workers, we can notice minimal improvements and moments were there are many workers achieving no work.

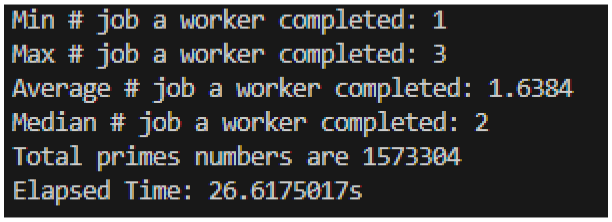
2. Using the largest number of M workers (10000), N = 64KB, C = 1KB, and a random data file of 1GB

   Run # 1 Results:



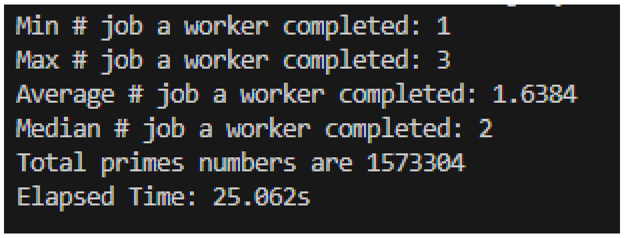
```
Min # job a worker completed: 1
Max # job a worker completed: 3
Average # job a worker completed: 1.6384
Median # job a worker completed: 2
Total primes numbers are 1573304
Elapsed Time: 24.9535017s
```

   Run # 2 Results:



```
Min # job a worker completed: 1
Max # job a worker completed: 3
Average # job a worker completed: 1.6384
Median # job a worker completed: 2
Total primes numbers are 1573304
Elapsed Time: 26.6175017s
```

   Run # 3 Results:

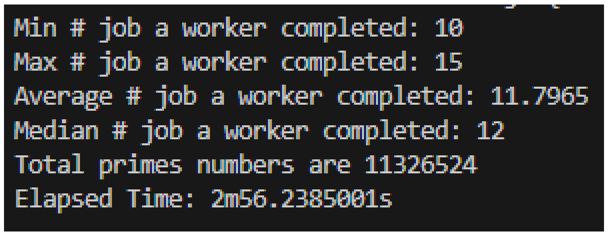


```
Min # job a worker completed: 1
Max # job a worker completed: 3
Average # job a worker completed: 1.6384
Median # job a worker completed: 2
Total primes numbers are 1573304
Elapsed Time: 25.062s
```

   From these results we have an average elapsed time of 25.54s for a random data of 1 GB.

3. Using the largest number of M workers (10000), N = 64KB, C = 1KB, and a random data file of 7.2GB. After running the program, I am able to observe the largest data file I can process which is around the size of 7.2 GB.

   Below are the results of that run:



```
Min # job a worker completed: 10
Max # job a worker completed: 15
Average # job a worker completed: 11.7965
Median # job a worker completed: 12
Total primes numbers are 11326524
Elapsed Time: 2m56.2385001s
```

4. The elapsed time seems to change, as M changes from 1 to the maximum value (10000) from Q1. In my observation, there was a significant improvement to elapsed time, as the number of workers increases the elapsed time decreases. The elapsed time seemed to also be affected when varying the values of N. I observed as N increases the elapsed time decreases, however for the C parameter, I noticed no effect on the elapsed time from Q3.