# Assignment2 Report Part2 Primenumber

Name: Yahong Liu

SID: 440358628

## 1. Problem definition and Requirements

In the second part of this assignment, we need to use Pthreads to find the prime numbers within a range from 2 to n using the Sieve of Eratosthenes.

The users are asked to input two parameters: the largest integer in the range and the number of threads. In addition, the program need to output all the prime numbers to a file.

In this program, we need to consider the load balancing problem when splitting the computation.

## 2. Parallel algorithm design

### 2.1 Design

With the input parameters: the largest integer INT\_LARGEST and the number of threads NUM\_THREADS.

Firstly, we need to check that INT\_LARGEST is larger than or equal to 2 and NUM\_THREADS larger than or equal to 1.

Secondly, we need to create a global array for computation with the size (INT\_LARGEST+1). So that the array range is from 0 to INT\_LARGEST. Since even numbers except 2 are not prime numbers, we could eliminate them to reduce the computation. With the global array, we set all the odd numbers to 1 and even numbers to 0.

Thirdly, we will allocate the array for each thread equally. So that we will have minimum and maximum boundary for each thread as shown below.

For calculating minimum and maximum boundaries, first I assume each thread has an extra number. This will get out of the range of array. At this assumption step:

(1) Minimum boundary is: ;

(2) Maximum boundary is: .

However only if the thread meets the requirement: it will be allocating an extra number. Thus, we subtract the wrong allocation and will get the right boundary.

(1)

(2)

Fourthly, to apply the method of Eratosthenes, the checking base numbers are odd numbers from 3 to . Within the range of minimum and maximum boundary, we loop through the array with an index. As we have eliminated all the even numbers, the index is thereby odd number. If the index number is the multiples of base numbers, then set the value of this index in array to 0.

In the end, with each thread finished their own work in the global array, we need to set 2 to prime number as we have eliminated it in the previous stage for less computation. Then we will print the result into a file.

### 2.2 Pros and Cons of Algorithm

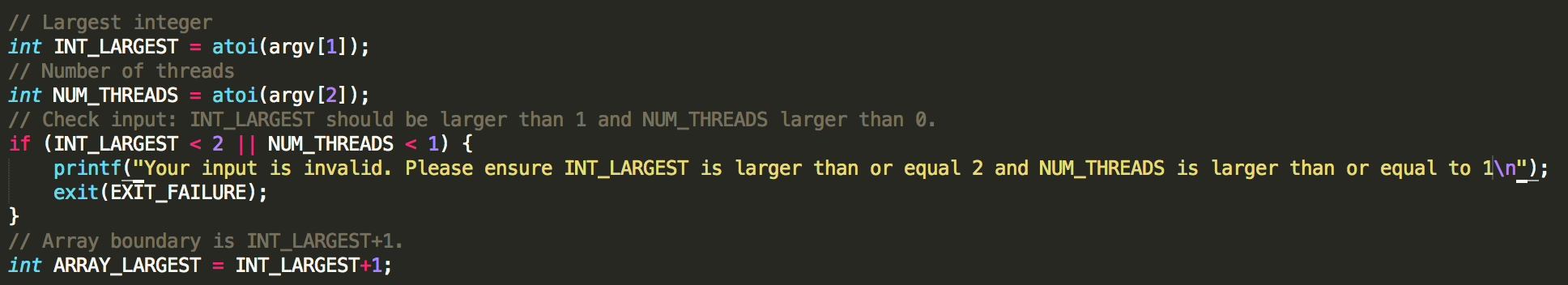
Since we have separated the global array to each thread, we do not need to set locks to get synchronization among these threads. Without locks, the program is easy to implement and the overhead of locks could be avoided. But we need to design the boundary and size of each thread very carefully, otherwise errors would appear.

## 3. Implementation and Testing

### 3.1 Implementation

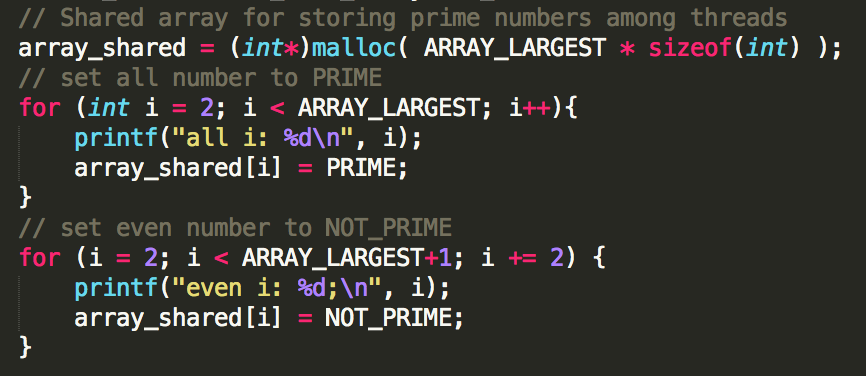
#### 3.1.1 Get Input Arguments

Get the input arguments and check if they are available.



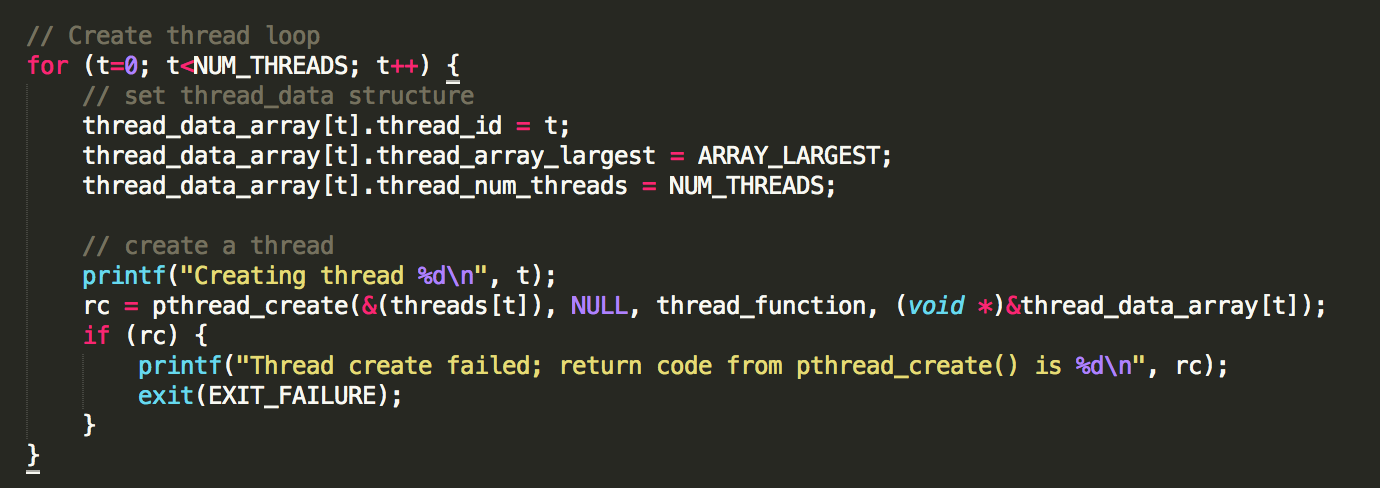
#### 3.1.2 Create Global Array and Set Even Numbers to 0

Create the global array and set all the even numbers to 0.



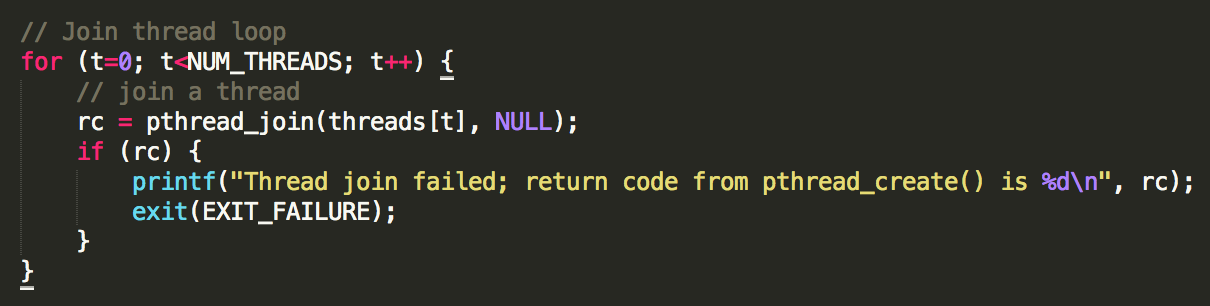
#### 3.1.3 Create Threads and Pass the Arguments

Create threads due to input NUM\_THREADS and pass the INT\_LARGEST and NUM\_THREADS to each thread.



#### 3.1.4 Join the Threads

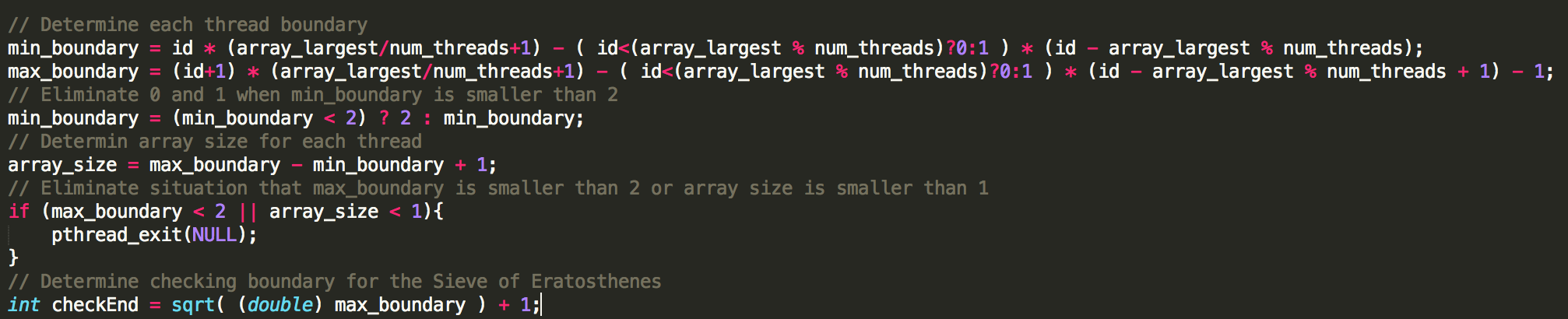
Join the threads for computation.



#### 3.1.5 Each Thread Tasks

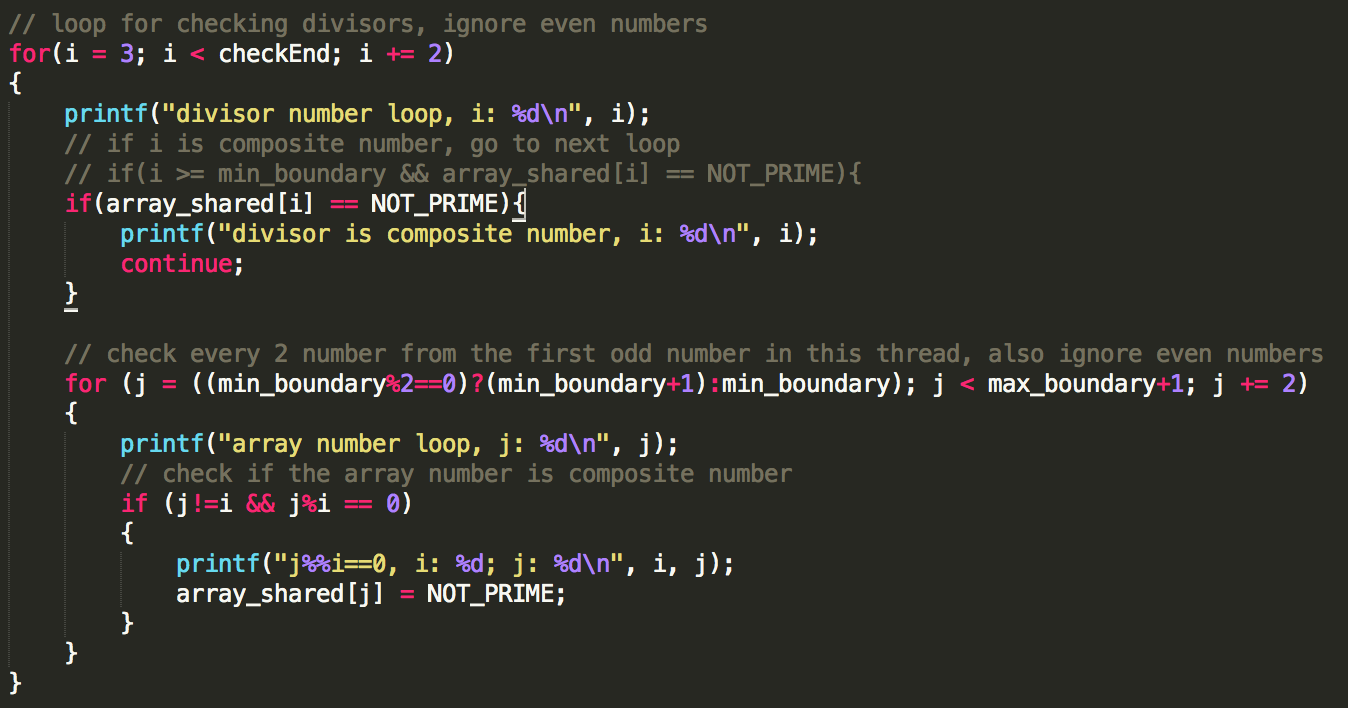
For the thread function, each thread will determine the boundary of itself. If the range is starting smaller than 2, then make the minimum boundary to 2. So that will get the size of computation inside this thread and we will get the largest checking base number.

In addition, if the NUM\_THREADS are larger than INT\_LARGEST, some of the range will be smaller than 1. We will exit these threads with this situation.



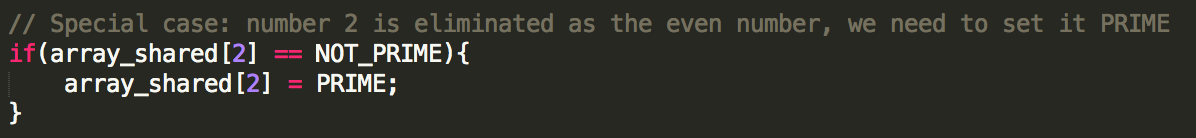
Then, we come to the computation of Sieve of Eratosthenes. Looping through the array from minimum to maximum boundary, we check each index if it is the multiple of base numbers from 3 to .

In addition, to further reduce the computation, we will check the base numbers. If any base numbers are composite number, we will eliminate these base numbers to reduce wasteful computation.



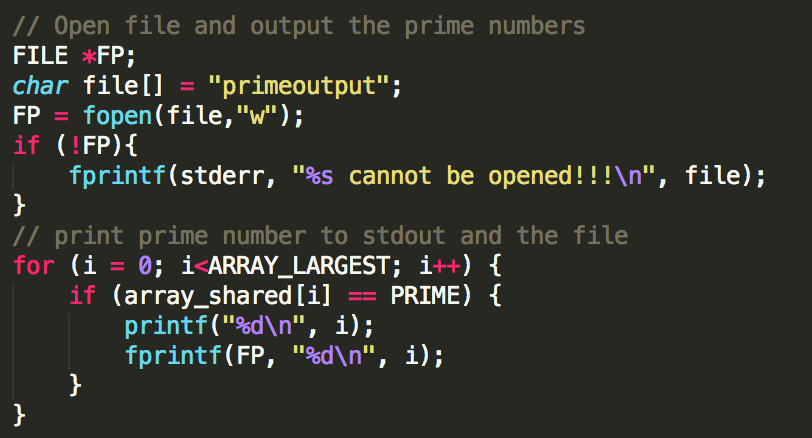
#### 3.1.6 Set 2 to Prime Number

Set 2 to prime number as we have eliminated it in the previous stage for less computation.



#### 3.1.7 Write Result to File

In the end, we write the result into a single file.



### 3.2 Testing

#### 3.2.1 INT\_LARGEST is smaller than 2 or NUM\_THREADS is smaller than 1

Use command to test and this will inform you to input right arguments

## ./primenumber\_yahongliu 2 0

## ./primenumber\_yahongliu 1 1

#### 3.2.2 Test INT\_LARGEST as 25 and NUM\_THREADS as 10

Use command to test this situation that the thread numbers are smaller than the largest integer.

## ./primenumber\_yahongliu 25 10

#### 3.2.3 Test INT\_LARGEST as 25 and NUM\_THREADS as 25

Use command to test the situation that these two arguments are the same.

## ./primenumber\_yahongliu 25 25

#### 3.2.4 Test INT\_LARGEST as 25 and NUM\_THREADS as 40

Use command to test the situation that thread numbers are larger than the largest integer.

## ./primenumber\_yahongliu 25 40

## 4. Manual

### 4.1 On Linux platform, extract tar file to current directory

## tar -xvf ./assignment2\_yahongliu.tar

### 4.2 Change the directory

## cd assignment2\_yahongliu/2\_primenumber

### 4.3 Run Makefile to compile the program

## make

### 4.4 Run the program with the largest integer and thread numbers

Input the parameters: largest integer, number of threads.

For example: largest number is 100 and 10 threads.

## ./primenumber\_yahongliu 100 10

### 4.5 Outputs

The outputs will be prime numbers in range of 2 to 100.