Ashwin Nellimuttath
SID: 862393744

Assignment 3 - Report

To run the program simply add the files sieve1/2/3.c to tardis. Run the command sbatch submit.sh in the script file. At the end of all the running process, run ./data.sh from the script folder. We can see the final result in the result folder.

Results

sieve0:
The total number of prime: 455052511, total time: 27.612252, total node 32
The total number of prime: 455052511, total time: 26.391304, total node 64
The total number of prime: 455052511, total time: 7.038469, total node 128
The total number of prime: 455052511, total time: 6.405855, total node 256
sieve1:
The total number of prime: 455052511, total time: 14.509519, total node 32
The total number of prime: 455052511, total time: 7.391080, total node 64
The total number of prime: 455052511, total time: 6.524982, total node 128
The total number of prime: 455052511, total time: 3.259400, total node 256
sieve2:
The total number of prime: 455052511, total time: 14.474002, total node 32
The total number of prime: 455052511, total time: 7.298960, total node 64
The total number of prime: 455052511, total time: 6.139179, total node 128
The total number of prime: 455052511, total time: 3.239146, total node 256
sieve3:
The total number of prime: 455052511, total time: 14.359772, total node 32
The total number of prime: 455052511, total time: 7.265453, total node 64
The total number of prime: 455052511, total time: 6.027105, total node 128
The total number of prime: 455052511, total time: 3.227752, total node 256

Part 1: Modify the parallel Sieve of Eratosthenes program in class so that the program does NOT set aside memory for even integers.

Ans). In the part one of the program, to optimize the Sieve of Eratosthenes, we do not need to store even numbers in the array since the only even prime number is 2. We can **add +1 to the final count** to add the count of 2.
We change the algorithm such that only odd integers are represented, halves the amount of storage required and doubles the speed at which multiples of a particular prime are marked. With this change the estimated execution time of the sequential algorithm becomes half
Sieve0 is the timings when we run the algorithm without optimization
sieve0:
The total number of prime: 455052511, total time: 27.803920, total node 32

The total number of prime: 455052511, total time: 26.365304, total node 64
The total number of prime: 455052511, total time: 7.004961, total node 128
The total number of prime: 455052511, total time: 6.377673, total node 256

Sieve1 is the timing of the algorithm when we run the algorithm after using only odd numbers **sieve1**:
The total number of prime: 455052511, total time: 14.465093, total node 32
The total number of prime: 455052511, total time: 7.369440, total node 64
The total number of prime: 455052511, total time: 6.440130, total node 128
The total number of prime: 455052511, total time: 3.260411, total node 256

The improved algorithm is about half the time required for the original algorithm.

```
prime = 3;
 do {
     if (prime * prime > low_value)
         first = prime * prime - low_value;
     else {
         if (!(low_value % prime)) first = 0;
         else first = prime - (low_value % prime);
     }
     for (i = first; i < size; i += prime){
        temp = i + low_value;
         if((temp)%2==0){
         } else {
            marked[(i)/2]=1;
     }

     }
     if (!id) {
         while (marked[++index]);
         prime = index*2 + 3;
         // prime = index + 2;
     }
     if (p > 1) MPI_Bcast(&prime, 1, MPI_INT, 0, MPI_COMM_WORLD);
 } while (prime * prime <= n);
```

Here we can see we set the first prime number as 3 instead of two. Also we are only marking if the number is not divisible by 2.
At the end we add global_count + 1 to handle the count of 2

Part 2: Modify the parallel Sieve of Eratosthenes program in Part 1 so that each process of the program finds its own sieving primes via local computations instead of broadcasts.

In the original algorithm we find the new value of prime in the first process and then broadcasting that to the rest of the processes. During the whole algorithm this process is repeated many times. So in this optimization techniques instead of broadcasting the value of prime to every other process we calculate the next value of prime in each process.
So in this case each task will have its own copy of the array containing all primes. So therefore we can eliminate the broadcast step.

```
        long long int first_0 = prime*prime - low_value_first;
        for(i = first_0; i < (sqrt(n)); i += prime){
                 temp = i+low_value_first;
                 if((temp)%2==0){
                    }else{
                       marked_prime[(i)/2]=1;
                    }
        }
```

sieve1:
The total number of prime: 455052511, total time: 14.465093, total node 32
The total number of prime: 455052511, total time: 7.369440, total node 64
The total number of prime: 455052511, total time: 6.440130, total node 128
The total number of prime: 455052511, total time: 3.260411, total node 256

Sieve2 - After eliminating the broadcast step.
sieve2:
The total number of prime: 455052511, total time: 14.459497, total node 32
The total number of prime: 455052511, total time: 7.297940, total node 64
The total number of prime: 455052511, total time: 6.073918, total node 128
The total number of prime: 455052511, total time: 2.678513, total node 256

Here we can see that there is some improvement in the timing of sieve2.

Part 3: Modify the parallel Sieve of Eratosthenes program in Part2 so that the program can have effective uses of caches.
Ans.)
Here or each value of prime we are marking the array. So in each loop the array is marked. Here we can make use of cache since we might put same array values. So according to the cache spatial locality some of the nearby elements can also enter the cache lines. SO while accessing the array elements we can have cache hits.

Here in this optimisation instead of looping for each value of prime we fill the cache with a section of the larger subarray. So before the next loop we can mark all cache elements at once so we can have some sort of spatial locality.
sieve3:
The total number of prime: 455052511, total time: 14.359772, total node 32
The total number of prime: 455052511, total time: 7.265453, total node 64
The total number of prime: 455052511, total time: 6.027105, total node 128
The total number of prime: 455052511, total time: 3.227752, total node 256

```
            register long long int B = 10000;

            for(i = 0; i< size; i+=B){
                for (j = first; j < MIN(size,i+B); j += prime){
                    temp = j+low_value;
                    if((temp)%2==0){
                    }else{
                        marked[(j)/2]=1;
                    }
                 }
                for (j = first2; j < MIN(size,i+B); j += prime2){
                    temp = j+low_value;
                    if((temp)%2==0){
                    }else{
                        marked[(j)/2]=1;
                    }
                }
            }
```

In our example we make use of 2 primes. 3 and 5.
Instead of marking the array in each loop here we mark the primes in one single big subarray loop. Also here for each process we maintain the marked_prime array.That is for both prime1 and prime2 cases.

```
          long long int first_0 = prime * prime - low_value_first;
          for (i = first_0; i < (sqrt(n)); i += prime)
          {
              temp = i + low_value_first;
              if ((temp) % 2 == 0)
              {
              }
```

```
            else
            {
                marked_prime[(i) / 2] = 1;
            }
        }
        first_0 = prime2*prime2 - low_value_first;
    for(i = first_0; i < (sqrt(n)); i += prime2){
        temp = i+low_value_first;
        if((temp)%2==0){
        }else{
            marked_prime[(i)/2]=1;
        }
    }
```

In the end we update both prime1 and prime2

```
                while (marked_prime[++index]);
                    prime = index*2 + 3;
                    if (prime2*prime2>n){
                    prime2 = prime;
                    }
                    prime2 = index*2 + 3;
```