ASSIGNMENT 1

Group #3

Contents

Solution 1
Source Code 3
Benchmark 6
    Conclusion: 7
Results 7

Solution

Divide the numbers of the given range according to the following equation:

$$n_p = \left\lfloor \frac{n}{P} \right\rfloor + \begin{cases} 1 & \text{if } p < \text{mod}(n, P), \\ 0 & \text{else,} \end{cases} \tag{10.4}$$

Then we can get the index of the range for every processor.

```
// calculate how many numbers one processor need to handle
long long int local_n = floor(maximum / num_procs);
long long int remainder = fmod(maximum, num_procs);

// determine the range index for a processor
long long start = my_rank * local_n + MIN(my_rank, remainder);
long long end = (my_rank + 1) * local_n + MIN((my_rank + 1), remainder);
```

And next we use the index to make a cycle for every processor to calculate their own largest prime gap. Don't forget to get the gap between the last prime of the current processor and the first prime of the next processor.

```
// cycle for finding the largest gap of the processor by comparing all gaps
while (i < end) {
  // calculate the gap
  mpz_sub(gap, curr_prime, pre_prime);
  temp_gap = mpz_get_d(gap);

  // judge if this is the largest gap for this processor
  if (temp_gap > largest_gap) {
    mpz_set(gmp_prime1, pre_prime);
    mpz_set(gmp_prime2, curr_prime);
    largest_gap = temp_gap;
  }

  // continue to find the next gap
  mpz_set(pre_prime, curr_prime);
  mpz_nextprime(curr_prime, pre_prime);
  i = mpz_get_d(curr_prime) + 1;
}

// Now :
// pre_prime --- the last prime of this processor
// curr_prime --- the first prime for the next processor
// We also need to consider the gap between these two primes
// to simplify the communication amoung processors
mpz_sub(gap, curr_prime, pre_prime);
temp_gap = mpz_get_d(gap);

// judge if it is the largest
if (temp_gap > largest_gap) {
  mpz_set(gmp_prime1, pre_prime);
  mpz_set(gmp_prime2, curr_prime);
  largest_gap = temp_gap;
}
```

Finally, every other processors send the results to processor 0 who will find the largest gap by comparing every largest gap of every processor.

```
// if processor 0, wait to receive the messages which is the largest gap of every other processor
if (my_rank == 0) {
  double proc_gap;
  long long prime_1 = mpz_get_d(gmp_prime1);
  long long prime_2 = mpz_get_d(gmp_prime2);
  long long proc_prime_1 = 0;
  long long proc_prime_2 = 0;

  // receive method
  for (source = 1; source < num_procs; source++) {
    // use different tags to distinguish different messages
    // 0 for gap    1 for the 1st prime    2 for the 2nd prime
    MPI_Recv(&proc_gap, 1, MPI_DOUBLE, source, 0, MPI_COMM_WORLD, &status);
    MPI_Recv(&proc_prime_1, 1, MPI_LONG_LONG, source, 1, MPI_COMM_WORLD, &status);
    MPI_Recv(&proc_prime_2, 1, MPI_LONG_LONG, source, 2, MPI_COMM_WORLD, &status);
    // determine which is the largest
    if (proc_gap > largest_gap) {
      prime_1 = proc_prime_1;
      prime_2 = proc_prime_2;
      largest_gap = proc_gap;
    }
  }

  // print the result
  printf("Largest Primes Gap: %.0f \t between: %.0lli and %.0lli\n", largest_gap, prime_1, prime_2);
}
// else send the messages to processor 0
else {
  long long prime_1 = mpz_get_d(gmp_prime1);
  long long prime_2 = mpz_get_d(gmp_prime2);

  MPI_Send(&largest_gap, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
  MPI_Send(&prime_1, 1, MPI_LONG_LONG, 0, 1, MPI_COMM_WORLD);
  MPI_Send(&prime_2, 1, MPI_LONG_LONG, 0, 2, MPI_COMM_WORLD);
}
```

Source Code

Dear professor, please be careful when copying the code to run on your own computer. It may have some syntax errors because of the copying operation. But it will usually work I think.

```
#include <stdio.h>
#include <math.h>
#include <gmp.h>
#include "mpi.h"

#define MIN(a,b) (a < b ? a : b)

int main(int argc, char** argv) {
    int my_rank;
    int num_procs;
    int source;
    int dest;
    long long maximum = 1000000000;
    double largest_gap = 0;
    double temp_gap = 0;
    MPI_Status   status;

    // initialize MPI
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
    MPI_Barrier(MPI_COMM_WORLD);
    double elapsed_time = -MPI_Wtime();

    // calculate how many numbers one processor need to handle
    long long int local_n = floor(maximum / num_procs);
    long long int remainder = fmod(maximum, num_procs);

    // determine the range index for a processor
    long long start = my_rank * local_n + MIN(my_rank, remainder);
    long long end = (my_rank + 1) * local_n + MIN((my_rank + 1), remainder);

    // initialize mpz types to store "the two primes" in every processor
    mpz_t gmp_prime1;
    mpz_init_set_d(gmp_prime1, 2);
    mpz_t gmp_prime2;
    mpz_init_set_d(gmp_prime2, 2);
```

```
// cast the long long type to mpz_t type for better use of GMP
mpz_t gmp_index;
mpz_init_set_d(gmp_index, start);

mpz_t gap;
mpz_init(gap);
mpz_t pre_prime;
mpz_init(pre_prime);
mpz_t curr_prime;
mpz_init(curr_prime);

// find the first prime for the processor
mpz_nextprime(pre_prime, gmp_index);
mpz_set(curr_prime, pre_prime);
long long i;
i = mpz_get_d(pre_prime) + 1;

// cycle for finding the largest gap of the processor by comparing all gaps
while (i < end) {
    // calculate the gap
    mpz_sub(gap, curr_prime, pre_prime);
    temp_gap = mpz_get_d(gap);

    // judge if this is the largest gap for this processor
    if (temp_gap > largest_gap) {
        mpz_set(gmp_prime1, pre_prime);
        mpz_set(gmp_prime2, curr_prime);
        largest_gap = temp_gap;
    }

    // continue to find the next gap
    mpz_set(pre_prime, curr_prime);
    mpz_nextprime(curr_prime, pre_prime);
    i = mpz_get_d(curr_prime) + 1;
}

// Now :
// pre_prime --- the last prime of this processor
// curr_prime --- the first prime for the next processor
// We also need to consider the gap between these two primes
// to simplify the communication amoung processors
mpz_sub(gap, curr_prime, pre_prime);
temp_gap = mpz_get_d(gap);
```

```
    // judge if it is the largest
    if (temp_gap > largest_gap) {
        mpz_set(gmp_prime1, pre_prime);
        mpz_set(gmp_prime2, curr_prime);
        largest_gap = temp_gap;
    }

    // if processor 0, wait to receive the messages which is the largest gap of every other processor
    if (my_rank == 0) {
        double proc_gap;
        long long prime_1 = mpz_get_d(gmp_prime1);
        long long prime_2 = mpz_get_d(gmp_prime2);
        long long proc_prime_1 = 0;
        long long proc_prime_2 = 0;

        // receive method
        for (source = 1; source < num_procs; source++) {
            // use different tags to distinguish different messages
            // 0 for gap       1 for the 1st prime       2 for the 2nd prime
            MPI_Recv(&proc_gap, 1, MPI_DOUBLE, source, 0, MPI_COMM_WORLD, &status);
            MPI_Recv(&proc_prime_1, 1, MPI_LONG_LONG, source, 1, MPI_COMM_WORLD, &status);
            MPI_Recv(&proc_prime_2, 1, MPI_LONG_LONG, source, 2, MPI_COMM_WORLD, &status);
            // determine which is the largest
            if (proc_gap > largest_gap) {
                prime_1 = proc_prime_1;
                prime_2 = proc_prime_2;
                largest_gap = proc_gap;
            }
        }

        // print the result
        printf("Largest Primes Gap: %.0f \t between: %.0lli and %.0lli\n", largest_gap, prime_1, prime_2);
    }
    // else send the messages to processor 0
    else {
        long long prime_1 = mpz_get_d(gmp_prime1);
        long long prime_2 = mpz_get_d(gmp_prime2);
```

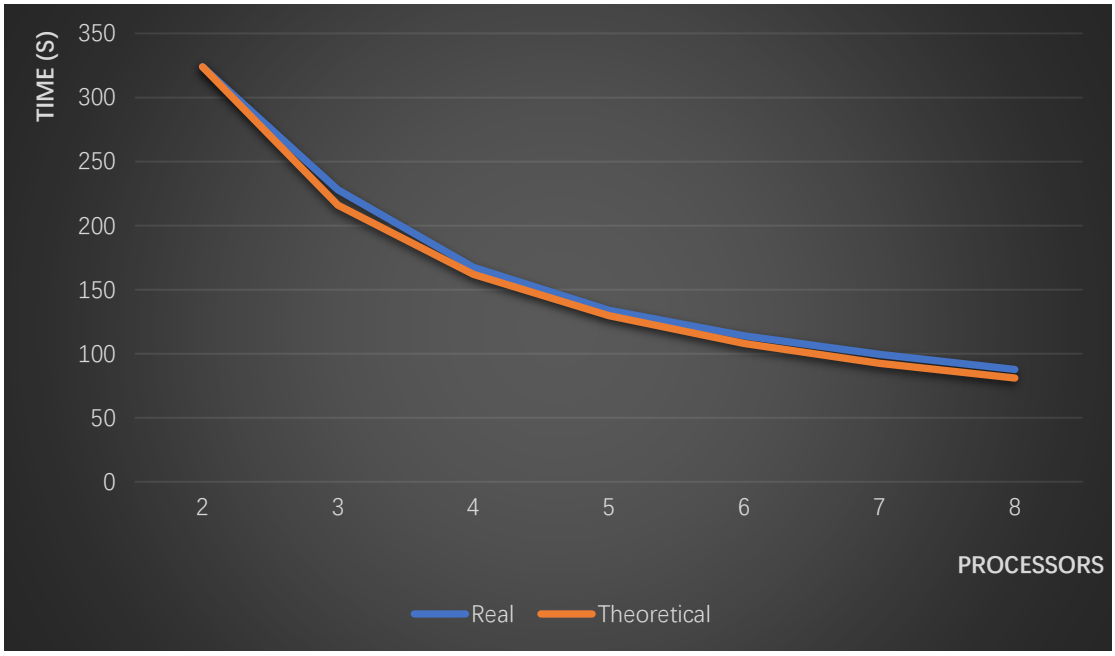
```
            MPI_Send(&largest_gap, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
            MPI_Send(&prime_1, 1, MPI_LONG_LONG, 0, 1, MPI_COMM_WORLD);
            MPI_Send(&prime_2, 1, MPI_LONG_LONG, 0, 2, MPI_COMM_WORLD);
        }

        // get the elapsed time for benchmarking
        elapsed_time += MPI_Wtime();
        printf("The elapsed time is: %lf\n", elapsed_time);

        MPI_Finalize();
}
```

Benchmark

These values are derived on Oct 4, 2019. (About 12:00am) Sometimes, the values change according to different time when we run the program. But the trend would not change.



#Processors	Real Time (s)	Theoretical Time (s)
2	323.92	323.92
3	228.03	215.95
4	167.35	161.96
5	133.88	129.57
6	114.06	107.97
7	99.37	92.55
8	87.67	80.98

Conclusion:

As the graph shows, the execution time decreases as we add processors. And it's a "perfect" speed improvement theoretically. But in reality, due to the reduction operations, the theoretical and real execution time are different and the difference grows with adding processors.

Results

For range [1, 10^9], the largest gap is 282 between 436273009 and 436273291

```
FINAL LARGEST GAP: 282   between: 436273009 and 436273291
```

For range [1, 10^12], the largest gap is:

Unfortunately, It took too long to find it.