High-Performance Computing

Find the largest prime number gap

Anand Kamble
Department of Scientific Computing
Florida State University

1 Introduction

This assignment focuses on developing an OpenMP program to find the largest prime number gap between two large positive integers, n and m. Two approaches are presented: one utilizing an array for simplicity, and an advanced option aiming to reduce memory consumption. The emphasis is on parallel processing efficiency, with performance analysis involving varying core configurations. The report concludes by summarizing findings and discussing implementation trade-offs.

2 Implementation

This program primarily has two parts, first responsible for finding all the prime numbers in the given range. The second part finds the largest gap between the two numbers. The boolean array stored if a number is prime or not is shared and used in both parts.

2.1 Part I: Finding the prime numbers

A parallel loop is implemented to iterate through each number in the specified range. The function `isPrime()` checks the primality of each number, and the results are stored in the boolean array `is_prime`. The dynamic scheduling strategy (`schedule(dynamic)`) is adopted to enhance load balancing among threads, ensuring efficient parallel execution.

Below is the function along with the OpenMP directives:

```
#pragma omp parallel for schedule(dynamic) shared(is_prime)
   num_threads(threads)
    for (int i = n; i < m; i++)
    {
        if (isPrime(i))
        {
            is_prime[i - n] = true;
        }
        else
        {
            is_prime[i - n] = false;
        }
    }
```

After the execution of this phase, we obtain a boolean array where true values are positioned at indices corresponding to prime numbers. For instance, consider the range from 11 to 20. The resulting boolean array would look like:

```
[True, False, True, False, False, False, True, False, True, False]
//[11  , 12   , 13  , 14   , 15   , 16   , 17  , 18   , 19  , 20]
```

2.2 Part II: Finding the Largest Prime Gap

Having successfully marked the prime numbers in the given range, the second part of the program focuses on identifying the largest prime gap between two numbers within this range. The primary components of this phase include:

2.2.1 Custom Reduction for Maximum Prime Gap

To efficiently find the maximum prime gap, a custom reduction operation is declared. The structure `MaxPrimeGap` is utilized to store information about the maximum gap, including its size and the index where it occurs. The reduction operation ensures consistent and accurate updating of the maximum prime gap across multiple threads. [1]

```c
// Structure to store information about the maximum prime gap
struct MaxPrimeGap
{
    long long int max;
    long long int index;
};

// Declare a custom reduction for finding the maximum prime gap
#pragma omp declare reduction(primeMax : struct MaxPrimeGap : omp_out
   = (omp_in.max > omp_out.max) ? omp_in : omp_out)
```

2.2.2 Parallel Loop for Maximum Prime Gap Calculation

Another parallel loop is implemented to traverse the range of marked prime numbers. Within this loop, the program identifies consecutive prime numbers and calculates the gap between them. If a larger gap is found, the information is updated using the custom reduction operation. The loop breaks when the next prime number is encountered, ensuring accurate gap calculations without redundancy.

```c
// Parallel loop to find the maximum prime gap
#pragma omp parallel for reduction(primeMax : max_diff)
   num_threads(threads) schedule(dynamic)
for (int i = n; i < m; i++)
{
    if (is_prime[i - n])
    {
        // Loop to find the next prime number and calculate the gap
        for (int j = i + 1; j < m; j++)
        {
            if (is_prime[j - n])
            {
                if (j - i > max_diff.max)
                {
                    max_diff.max = j - i;
                    max_diff.index = i;
                }
                break;
            }
        }
    }
}
```

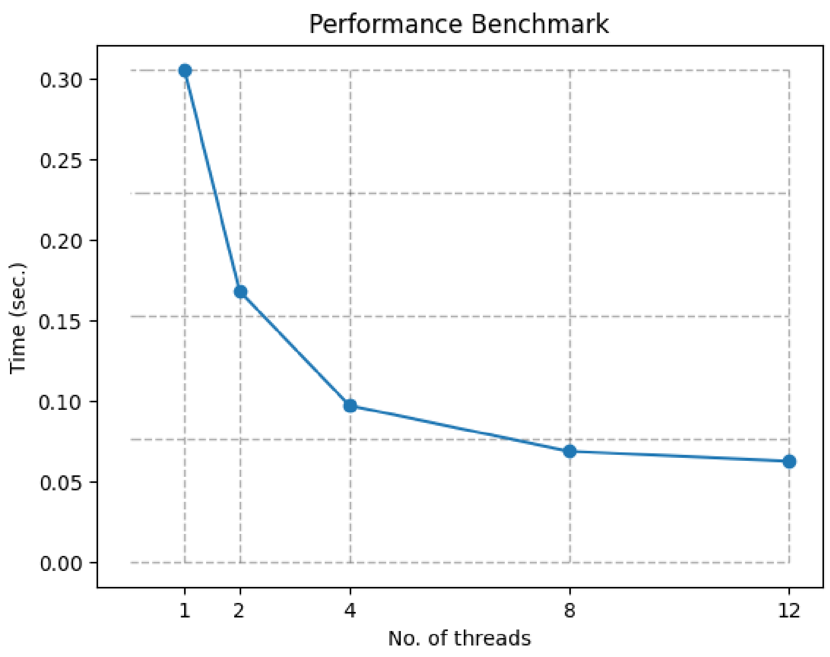
3 Errors and Debugging

While developing this program, the major problem faced was how to find the indices of the largest gap, since using `max` education clause only gives the maximum value but not the index on which it occurred.

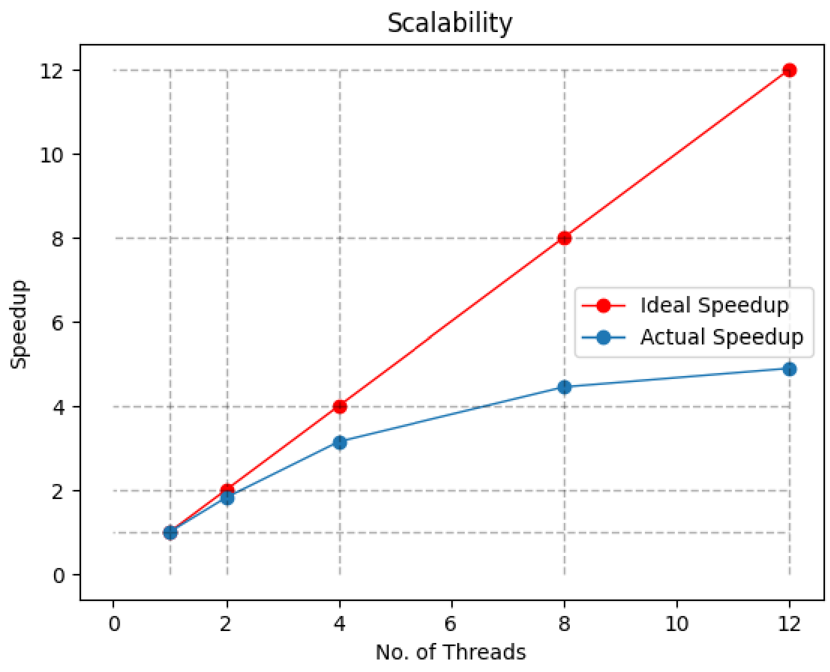
To solve this I created a structure for storing the maximum gap and its index and declared a custom OpenMP reduction. [2]

4 Results

This program was benchmarked across various thread counts, ranging from 1 to 12.



The following figure shows the scalability of the program compared to the ideal speedup.



5 Execution

Please use the command `‘make run‘` to compile the program and execute it. The source file and make file are provided in the zip bundle.

6 Future Enhancements

Currently, the program has two separate loops, instead of this we can merge the function for finding if the number is prime or not along with the loop to find the next prime number and the gap in between those. This way we won't require an additional array to store the boolean values which might improve the memory efficiency.

References

[1] OpenMP Architecture Review Board. *2.19.5.7 declare reduction Directive* , *https://www.openmp.org/spec-html/5.0/openmpsu107.html#x140-6020002.19.5.7*. November 2018.

[2] dreamcrash. *Reducing the max value and saving its index* , *https://stackoverflow.com/a/66668941/22647897*. 17 Mar, 2021.

[3] ChatGPT. *OpenMP Program: Prime Gap* , *https://chat.openai.com/share/c89567f5-f564-4936-b984-f02430911caf*. 5 Feb, 2017.