Parallel Programming Exercise 4 – 9

Author:	邱亮茗 (r12942159@ntu.edu.tw)
Student ID	R12942159
Department	Graduate Institute of Communication Engineering

(If you and your team member contribute equally, you can use (co-first author), after each name.)

1 Problem and Proposed Approach

(Brief your problem, and give your idea or concept of how you design your program.)

The program aims to find the largest gap between consecutive prime numbers under 1,000,000. Since calculating prime gaps is computationally intensive for large ranges, the task is distributed among multiple processors using MPI for parallel execution. Each processor computes a section of prime numbers and tracks the largest gap between consecutive primes in its range. The results from all processors are combined to determine the maximum gap across the entire range.

Step1. Odd-only Representation: Since even numbers (except 2) are not prime, the sieve only considers odd numbers, halving the memory requirement.
Step2. Partitioning the Range: The range up to 1,000,000 is split among processors. Each processor calculates primes within its own segment, while overlapping a portion of the range with adjacent processors to handle primes close to segment boundaries.
Step3. Parallel Sieve with MPI: Each processor marks non-primes in its segment based on smaller prime numbers. The main sieve loop calculates prime multiples and marks them as non-prime. And MPI_Bcast broadcasts the next prime from the root processor (id=0) to all processors to ensure synchronization of the sieve across segments.
Step5. Performance Measurement: The program runs multiple iterations (NUM_ITERATIONS) to average execution time, providing a more stable performance metric.

2 Theoretical Analysis Model

(Try to give the time complexity of the algorithm, and analyze your program with iso-efficiency metrics)

The main part of the algorithm is the Sieve of Eratosthenes, which has a time complexity of O(nloglogn), where n is the upper limit for the primes (1,000,000 in this case). Additionally, distributing the computation across p processors affects the complexity by dividing the work approximately evenly among them. Each processor handles O(n/p) work in marking non-primes within its subrange. The synchronization step where the root process broadcasts the prime number to others is O(logp), as MPI_Bcastgenerally operates in logarithmic time with respect to the number of processors.

Thus, under considering iterations, the time complexity of the algorithm is approximately: O(nloglogn / p + log p).

=>T_0(n, p) = plogp

=> M(n) = n; n >= Cplogp

=> M(f(p)) / p = Cplogp / p = Clogp

=> The system has good scalability

3 Performance Benchmark

(Give your idea or concept of how you design your program.)

Table 1. The execution time

Processors	1	2	3	4	5	6	7	8
Real execution time	0.003032	0.001627	0.001254	0.000934	0.001793	0.001794	0.001568	0.001414
Estimate execution time	0.003032	0.001516	0.001011	0.000758	0.000606	0.000505	0.000433	0.000379
Speedup	1	1.8635	2.4178	3.2462	1.6910	1.6900	1.9336	2.1442
Karp-flatt metrics		0.0732	0.1203	0.0774	0.4891	0.5101	0.4366	0.3901

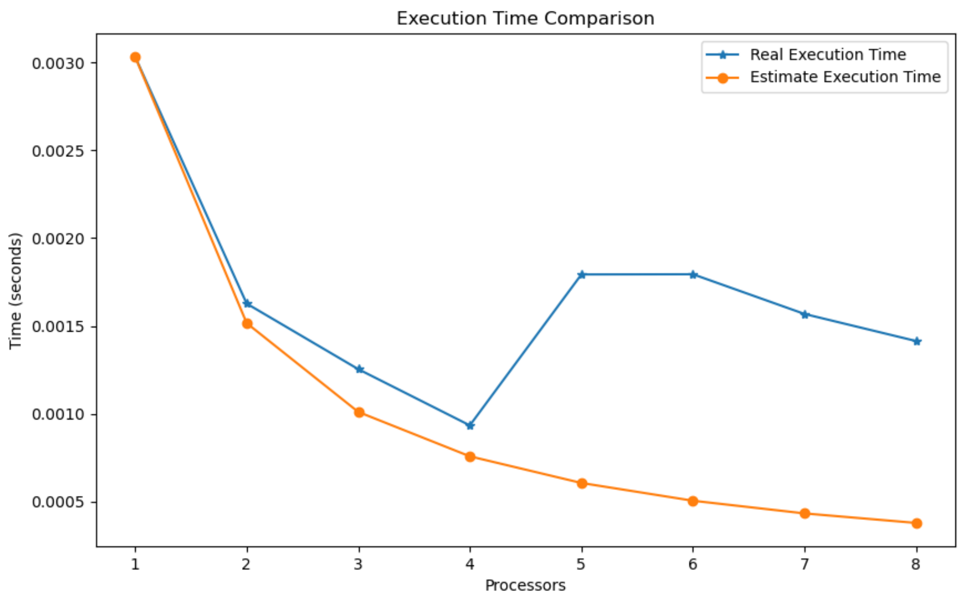

Figure 1. The performance of diagram

4 Conclusion and Discussion

(Discuss the following issues of your program

1. What is the speedup respect to the number of processors used?

   The speedup values indicate that efficiency improves with 2, 3, and 4 processors but then diminishes as more processors are added, likely due to increased communication overhead.
2. How can you improve your program further more

   By minimizing the number of MPI_Bcast calls, each processor can compute primes independently as much as possible, reducing the need for synchronization and communication overhead. And adjust the prime sieve to operate within cache-friendly segments. By structuring the sieve to fit within the cache size, you can reduce cache misses, which in turn improves memory access efficiency and overall program speed.
3. How does the communication and cache affect the performance of your program?

   Communication with more processors, communication costs rise, especially during synchronization steps like broadcasting the next prime or reducing the local gaps. This is particularly costly when the number of processors exceeds the optimal count for a given problem size. Excessive communication reduces parallel efficiency, as processors spend more time waiting for data rather than computing.

   Cache performance access patterns in the sieve algorithm heavily impact cache efficiency. If the algorithm frequently accesses non-contiguous memory, cache misses increase, slowing down the program. By optimizing the sieve to work on chunks that fit within the cache (e.g., through segmented sieving), cache utilization improves, enhancing performance.
4. How does the Karp-Flatt metrics and Iso-efficiency metrics reveal?

   Karp-Flatt Metric: This metric, which indicates the fraction of time spent on overhead, rises with more processors, especially after 4 processors, showing increased communication overhead. High Karp-Flatt values (0.4891 for 5 processors, 0.5101 for 6) suggest that communication dominates, limiting scalability.

   Iso-efficiency Metric: Iso-efficiency reflects the relationship between problem size, overhead, and processor count. As shown by the execution times and Karp-Flatt metric, this algorithm becomes inefficient with more than 4 processors for the problem size n=1,000,000. To maintain efficiency at higher processor counts, the problem size must increase proportionally.

)

Appendix(optional):

(If something else you want to append in this file, like picture of life game)