Parallel Programming Exercise 4 – 12

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 goal is to compute an approximation of π using Simpson's Rule for numerical integration, where f(x) = 1 / (1 + x ** 2), and integration limits are a = 0 and b = 1 with n = 50 intervals are used for the approximation.

Step1. Work Distribution: Each process is assigned a part of the summation. Processes with different ranks compute terms based on their rank and stride by the number of total processes to ensure each interval is covered without overlap.
Step2. Each process computes a partial result, stored in a local variable (local_area). And we use MPI_Reduce to sum up all local_area values from each process and store the result in global_area on the root process (rank 0).
Step3. Result Calculation: Only the root process (rank 0) divides global_area by 3n to complete the Simpson's Rule calculation and then prints the approximation of π.

2 Theoretical Analysis Model

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

Function Calculation: Each interval requires computing f(x), which is an O(1) operation. For n intervals, the sequential time complexity is O(n), as each interval requires a single computation.
Parallel Execution: The work each processor performs is O(n / p).
Communication Cost: MPI_Reduce has a communication complexity of O(logp), assuming a binary tree structure for reduction.
The total parallel time complexity of the algorithm is: O(n / p + logp)
=>To(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 base on 1000000 iterations

Processors	1	2	3	4	5	6	7	8
Real execution time	0.136507	0.215039	0.234368	0.352094	0.696259	0.897269	1.053608	2.807326
Estimate execution time	0.136507	0.068253	0.045502	0.034126	0.027301	0.022751	0.019501	0.017063
Speedup	1	0.6348	0.5824	0.3877	0.1961	0.1521	0.1295	0.0486
Karp-flatt metrics		2.1506	2.0753	3.1057	6.1256	7.6876	8.8380	23.3604

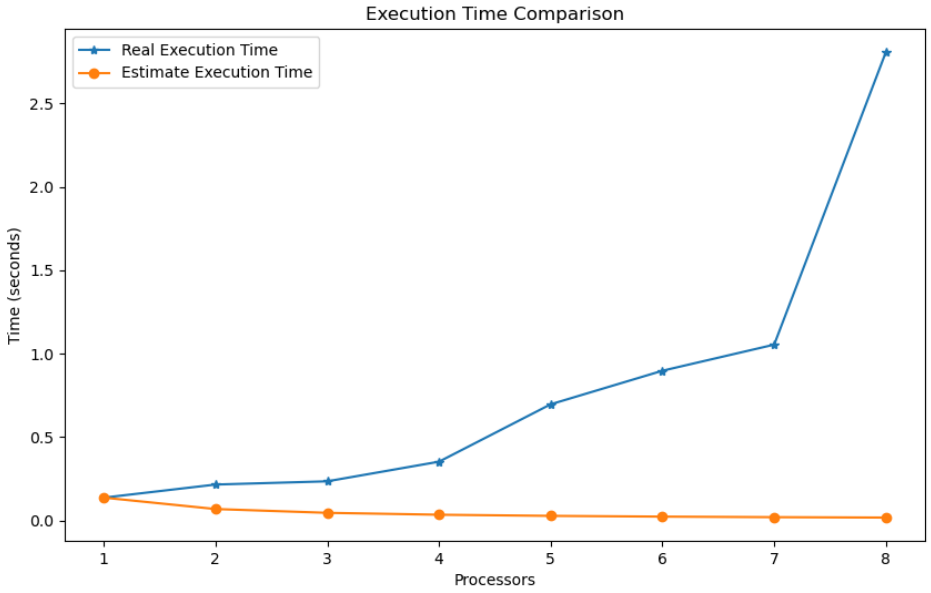

Figure 1. The performance of diagram

4 Conclusion and Discussion

(Discuss the following issues of your program

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

   As the number of processors increases, the speedup decreases significantly.

2. How can you improve your program further more

   The MPI_Reduce operation is necessary to collect results from all processors, but it introduces communication overhead, especially as the number of processors increases. In this case, we can avoid using parallel architectures because the small value of n results in communication overhead dominating the performance gains.

3. How does the communication and cache affect the performance of your program?

Communication overhead increases as the number of processors grows, leading to diminishing returns in speedup. Reducing communication by optimizing the use of MPI and minimizing synchronization points can help mitigate this effect.

Cache performance significantly affects the speed of memory-bound computations. Poor memory access patterns can lead to cache misses, increasing execution time. Improving data locality and memory access patterns can minimize these misses and improve performance.

In conclusion, both communication and cache optimization are key to enhancing the performance of my parallel program. Careful design of data distribution and communication strategies, as well as attention to memory access patterns, will help in maximizing the scalability and efficiency of the algorithm.

4. How does the Karp-Flatt metrics and Iso-efficiency metrics reveal?

   The Karp-Flatt metrics reveal inefficiency in the parallel program, especially as the number of processors increases. The program's real execution time increases much faster than the estimated execution time, showing that overheads, like communication and synchronization, are limiting scalability. And the sharp decline in speedup as the number of processors increases suggests that the problem size n is insufficient to sustain parallel efficiency. According to the iso-efficiency analysis, the program does not scale well because the problem size does not grow sufficiently with the number of processors to compensate for the increased communication costs.

)

Appendix(optional):

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