Parallel Programming Exercise 4 – 7

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 task is to calculate the 1 + 2 + … + p using a parallel approach. Each process should be assigned a unique integer value to sum, and all processes will then perform a reduction operation to compute the total sum. Process 0 should display the result of the reduction and validate it using the formula p(p+1)/2.
Step1. Parallel Decomposition: Each process is assigned a unique integer for summation, with process id given the integer value id + 1.
Step2. Reduction Operation: We use MPI's MPI_Reduce function with the MPI_SUM operator to sum all local sums across processes. The result is accumulated on process 0, which serves as the designated "master" process.
Step3. Validation: To verify the result, process 0 calculates the sum using the mathematical formula p(p+1)/2, where p is the total number of processes. It then compares this result to the value obtained from the reduction.
Step4. Output: Process 0 prints both the result from the reduction and the result from the formula for comparison, with fflush(stdout);used after each printf to ensure that output is displayed immediately.

2 Theoretical Analysis Model

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

Local Computation: This operation has a constant time complexity, O(1).
Reduction Operation: Using MPI_Reduce has a time complexity of O(log p).
Formula Calculation: Process 0 computes the formula, which is an O(1) operation.
Total Time Complexity: O(log p).

High Efficiency at Lower Processor Counts: Efficiency is above 0.8 for 1–4 processors, indicating effective parallelization with minimal overhead.
Efficiency Drops at Higher Processor Counts: From 4 processors onwards, efficiency declines sharply, suggesting increasing overhead due to factors like communication

and synchronization among processes.
Scalability: The decreasing efficiency with more processors implies limited scalability. To maintain high efficiency with more processors, the workload would need to grow significantly, which might not be feasible for this problem size.

3 Performance Benchmark

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

Table 1. The execution time base on p = 1,000,000

Processors	1	2	3	4	5	6	7	8
Real execution time	0.000915	0.000481	0.000322	0.000335	0.000496	0.000478	0.000439	0.000618
Estimate execution time	0.000915	0.000457	0.000305	0.000228	0.000183	0.000152	0.000131	0.000114
Speedup	1	1.9022	2.8416	2.7313	1.8447	1.9142	2.0842	1.4805
Karp-flatt metrics		0.0257	0.0186	0.1162	0.3423	0.3563	0.3375	0.5505

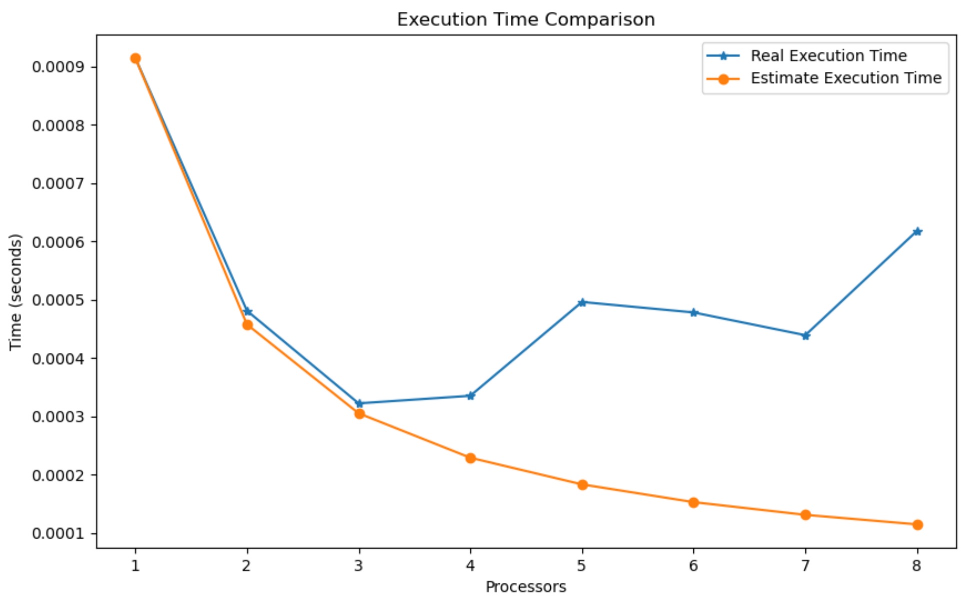

Figure 1. The performance of diagram

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 initially increases with more processors, reaching a peak at 3 processors with a speedup of approximately 2.8416. Beyond 3 processors, the speedup decreases, indicating that the overhead of parallelization outweighs the benefits of adding more processors. This suggests limited scalability in the program beyond 3 processors due to factors like communication and synchronization costs.

2. How can you improve your program further more

   MPI_Reduce involves communication among processes, which can become a bottleneck. For large numbers of processors, consider using non-blocking communication (e.g., MPI_Ireduce) or hierarchical reduction strategies to reduce latency.

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

   MPI communication, particularly in reduction operations, can introduce latency, especially as the number of processors increases. Communication overhead is often significant for fine-grained tasks where processes need to frequently exchange small amounts of data. And cache misses can slow down the program, especially if data is not accessed in a cache-friendly manner. For summing large ranges, if data is structured such that each process accesses memory sequentially (as it currently does in increments), cache performance is generally better. However, excessive communication can disrupt caching, especially if data has to be frequently exchanged between processors.

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

   The iso-efficiency metric measures how effectively the workload scales with an increasing number of processors, indicating the amount of additional work required to maintain a fixed level of efficiency as processors increase. Here, the Karp-Flatt metric values are low for processor counts up to 3, indicating minimal parallel overhead. However, from 4 processors onward, the values increase, suggesting rising communication or synchronization costs. This increase in overhead reflects suboptimal scaling, as shown by the speedup values that start to decrease after 3 processors. Based on these results, the program's iso-efficiency would imply that, beyond 3 processors, additional work or optimizations (e.g., minimizing communication or synchronization) would be necessary to achieve efficient scaling.

)

Appendix(optional):

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