Parallel Programming Exercise 4 – 6

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.)

We need a parallel program that allows each process to independently print a message indicating its unique rank. The output will look like: “hello, world, from process <i>”, where <i> is rank of the process.

Step1. Initialize MPI Environment: Use MPI_Init to start the MPI environment, preparing all processes to communicate within the MPI communicator (MPI_COMM_WORLD).
Step2. Identify Process Rank: Use MPI_Comm_rank to retrieve the rank (or unique identifier) of each process within MPI_COMM_WORLD. This rank allows each process to distinguish itself when printing its message.
Step3. Print the Message: Each process will use printf to output its message.
The fflush(stdout); statement ensures that all output is immediately flushed to the screen, making it less likely to appear out of order.
Step4. Finalize MPI: Conclude with MPI_Finalize to shut down the MPI environment, allowing the program to release resources cleanly after all processes have finished their tasks.

2 Theoretical Analysis Model

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

The time complexity of this simple parallel program is O(1).

The iso-efficiency metric assesses how well a parallel program can scale by balancing computation with communication and overhead as more processors are added. But in this case, there’s no parallel program can scale by computation, so the problem size does not grow as p increases. This means it cannot benefit from increasing the number of processors to achieve higher efficiency or performance improvements.

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	0.000001	0.000001	0.000001	0.000001	0.000001	0.000001	0.000001	0.000001
Estimate execution time	0.000001	0.000001	0.000001	0.000001	0.000001	0.000001	0.000001	0.000001
Speedup	1	1	1	1	1	1	1	1
Karp-flatt metrics		1	1	1	1	1	1	1

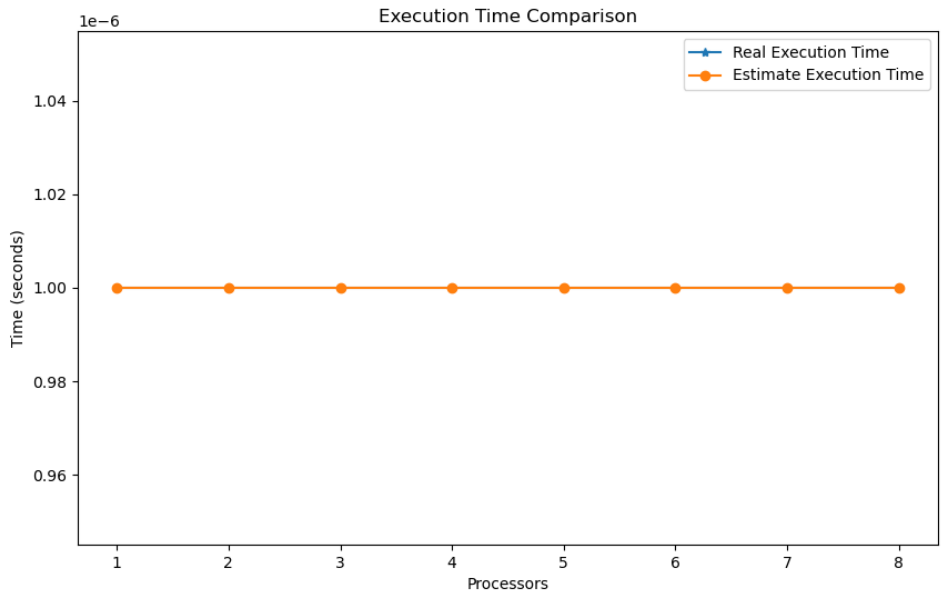

Figure 1. The performance of diagram

4 Conclusion and Discussion

(Discuss the following issues of your program

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

   The task is primarily sequential, as in this simple "hello, world" program where each process performs independent I/O, parallel execution provides no real benefit. Each process is essentially doing the same amount of work, and there's no parallel workload to benefit from multiple processors.

2. How can you improve your program further more

   Since each process performs I/O, it can cause contention and delay due to simultaneous calls to printf. Using MPI's MPI_Reduce to collect messages

into a single process and print them together could help minimize I/O contention.

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

   Each process prints independently, creating potential I/O bottlenecks if many processes run simultaneously. When processes access shared resources (like stdout), synchronization can increase wait times.

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

   No change in performance as processors increase, and the execution time doesn't decrease with more processors. This imply that the computation might be fully sequential. The iso-efficiency metric evaluates scalability by assessing how problem size should increase relative to processor count to maintain efficiency. If small tasks dominate execution time, you'll notice high iso-efficiency costs, meaning adding processors doesn't significantly improve performance. For simple programs like this, the iso-efficiency metric essentially does not apply here, as the program has a trivial computational load that does not benefit from parallelism.

)

Appendix(optional):

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