Parallel Programming Exercise 6 – 8

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 to measure two key network performance metrics between two processes:

1. Latency (λ): The time required for a zero-size (or minimal size) message to travel from one process to another.
2. Bandwidth (B): The maximum rate at which data can be transferred between the two processes, measured in bytes per second.

Step1. Process Setup: Use MPI to initialize two processes, Rank 0 and Rank 1. This back-and-forth process will simulate "ping-pong" communication, where Rank 0 can record the round-trip time.

Step2. Message Size Variation: Use a range of message sizes, starting from a minimal size (1 byte) up to a larger size (1 MB), doubling the size each iteration. For each size, measure the round-trip time over several trials to improve accuracy by averaging the results.

Step3. Timing Measurement: For each message size, Rank0 will record the time at the start and end of each round-trip, allowing us to compute the average round-trip time.

Step4. Latency and Bandwidth Estimation: Using the gathered data points, apply linear regression to estimate.

2 Theoretical Analysis Model

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

The time complexity(T) for a single round-trip message exchange in the ping-pong test can be expressed as: $T(n) = 2 (\lambda + n / B)$

3 Performance Benchmark

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

Setup and Validation: The program initializes MPI and checks if exactly two processes (p == 2) are involved, as this test is designed to measure the communication

overhead between just two nodes. If not, an error message is printed, and the program terminates. This requirement ensures that the timing measurements focus solely on the two-way communication without additional MPI nodes interfering.

Ping-Pong Communication Test :

1. Data Structure for Results: Arrays times and message_size store the average round-trip times and message sizes for different message lengths, allowing the program to gather data across a range of message sizes.
2. Varying Message Size: The program iterates through a range of message sizes, doubling the size each iteration from 1 byte up to 1 MB. This logarithmic scaling of message sizes helps to observe how bandwidth and latency vary with message size.
3. Communication and Timing: For each message size, the two processes perform a "ping-pong" exchange (MPI_Send and MPI_Recv), where:

    - Process 0 sends a message to Process 1.
    - Process 1 immediately returns the message.
    - Process 0 calculates the time taken for a round-trip.

Trial Averaging: This round-trip time measurement is repeated NUM_TRIALS times per message size to obtain an accurate average. The averaging helps smooth out any anomalies in individual timing measurements.

Latency and Bandwidth Estimation:

Latency (lambda) is estimated as the y-intercept, representing the minimal time required for a round-trip message of effectively zero length. And bandwidth (B) is derived from the slope, representing how much time it takes to transfer additional bytes as the message size increases.

Linear Regression for Bandwidth and Latency Estimation:

1. The program calculates a linear regression on the collected data points (message size vs. average time). It uses this model to approximate the network's latency and bandwidth:
    - **Latency (λ)**: Intercept of the line, indicating the time overhead when message size approaches zero.
    - **Bandwidth (B)**: Inverse of the slope, giving bytes per second as message size increases.
2. This design assumes a linear relationship between message size and time, suitable for measuring overhead and scalability of communication costs in small to medium-sized messages.

The performance benchmarks for the network latency and bandwidth are as follows:

1. Estimated Latency (λ) is 9.177276e+02 seconds: This represents the estimated round-trip time for a message of effectively zero length. Here, λ is around 917.7276 seconds, which is an unusually high latency. Typically, in MPI applications, latency is expected to be in the range of microseconds or milliseconds. This high value suggests there may be an issue in the configuration, such as improper timing or network setup.
2. Estimated Bandwidth (B) is 9.930001e+06 bytes/seconds: The estimated bandwidth is approximately 9.93 MB/s. This is a reasonable value for a network with modest performance. However, if this is on a high-performance network like InfiniBand or a Gigabit Ethernet, it may indicate that the bandwidth is lower than expected.

4 Conclusion and Discussion

(Discuss the following issues of your program

1. How can you improve your program further more

   Adding a "warm-up" phase at the beginning of the test, where several initial exchanges are conducted but not timed, allows the MPI library and hardware to reach stable states before measurement. This could improve accuracy by eliminating outliers due to setup delays or initial cache misses.

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

   Any contention on the network, such as multiple processes competing for the same resources, will degrade performance. Minimizing communication or optimizing it for fewer exchanges can reduce this impact.

   Caching can significantly speed up communication, especially when message sizes fit within typical cache lines (e.g., 64 bytes). If the data fits well into L1 or L2 cache, access times are much faster than main memory. Reusing buffers and keeping frequently used data in cache can improve performance. If each process reuses the same message buffer without reallocation, the data is more likely to remain in cache, reducing latency and improving bandwidth.

)

Appendix(optional):

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