CS-E4690 Programming Parallel Supercomputers
Sheet 3 Report
Nguyen Xuan Binh 887799

Part 1: Communication-time measurement

Exercise instruction:

Implement a simple point-to-point communication involving just two MPI processes and measure the communication time as a function of data package size for three different combinations out of the spectrum of the MPI send/recv routines.
For time measurement, use MPI_Wtime. (Consider that the OpenMPI library doesn't provide synchronization of the starting time across processes. So, you have to organize a proper synchronization yourself.)

In particular, compare the case "both processes on the same node" with "processes on different nodes". The number of processes per node can be controlled by the SLURM option `--ntasks-per-node`.

Execution on a cluster is affected by random effects, hence the results will in general be subject to scatter. Hence, for obtaining reliable mean values, you have to repeat the measurement at an appropriate rate.

Document your work in a separate PDF document containing
- a short description of the code,
- a graphical representation of the measured latency/bandwidth - package size relation,
- a short discussion of the results.

Bonus task: Compare communication performance (Bytes per second communicated) with compute performance (FLOPS). FLOPS can be estimated by running and timing a long loop, e.g. adding two large arrays.

I. Short description of the code

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char** argv) {
    MPI_Init(&argc,&argv);

    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Ensure there are exactly two MPI processes
    if (world_size != 2) {
        fprintf(stderr, "World size must be two for %s\n", argv[0]);
        MPI_Abort(MPI_COMM_WORLD, 1);
    }

     int sizes[] = {1024, 10*1024, 100*1024, 1024*1024, 10*1024*1024}; //
1KB, 10KB, 100KB, 1MB, 10MB
    int num_sizes = 5;
    double start_time, end_time;
    char* buffer;

    for (int i = 0; i < num_sizes; i++) {
        int size = sizes[i];
        buffer = (char*) malloc(size);

        if (world_rank == 0) {
            // MPI Barrier for synchronization

            MPI_Barrier(MPI_COMM_WORLD);

            start_time = MPI_Wtime();

            MPI_Send(buffer, size, MPI_CHAR, 1, 0, MPI_COMM_WORLD);

            end_time = MPI_Wtime();
```

```
            printf("Send %d bytes took %f seconds\n", size, end_time -
start_time);

        } else if (world_rank == 1) {

            // MPI Barrier for synchronization

            MPI_Barrier(MPI_COMM_WORLD);

                MPI_Recv(buffer, size, MPI_CHAR, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
        }

        free(buffer);
    }

    MPI_Finalize();
}
```

Initialization: The MPI environment is initialized, and the rank and size (number of processes) of the communicator are fetched from MPI_COMM_WORLD.

Process Check: The program ensures that exactly two MPI processes are used. If not, it aborts execution, as the communication is designed for a P2P between two processes.

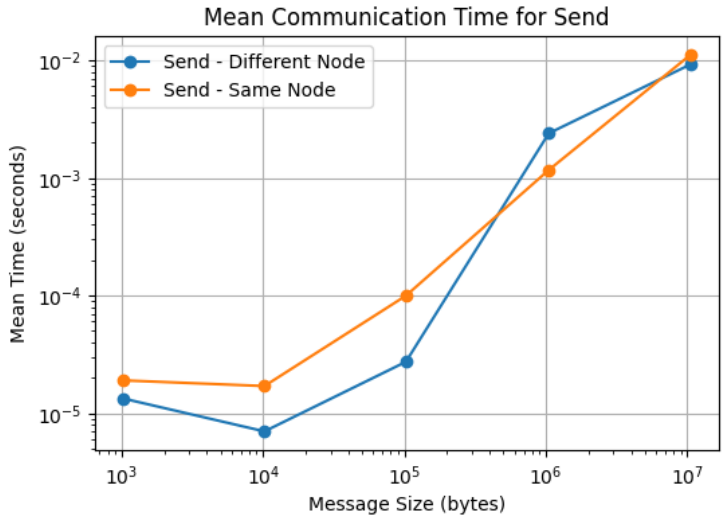
Time Loop: The program enters a loop over an array of each message size to measure the communication times. For each size, it does the following:

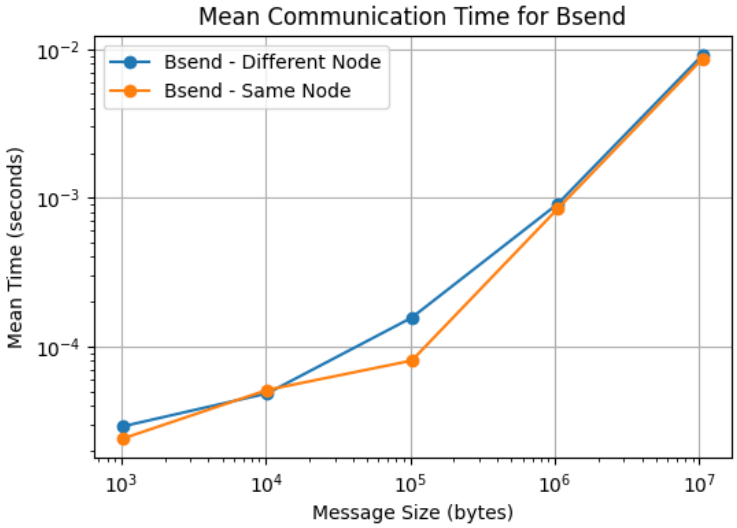
- Allocates a buffer for the message
- Uses MPI_Barrier to synchronize the sender and receiver, ensuring that both processes start at the same time before timing by MPI_Wtime begins.
- The sender process (rank 0) starts the timer, performs a buffered send operation, stops the timer, and reports the time taken to send the message.
- The receiver process (rank 1) waits for the message to arrive without timing its operation.
- After P2P communication, the program frees the allocated memory of the buffer

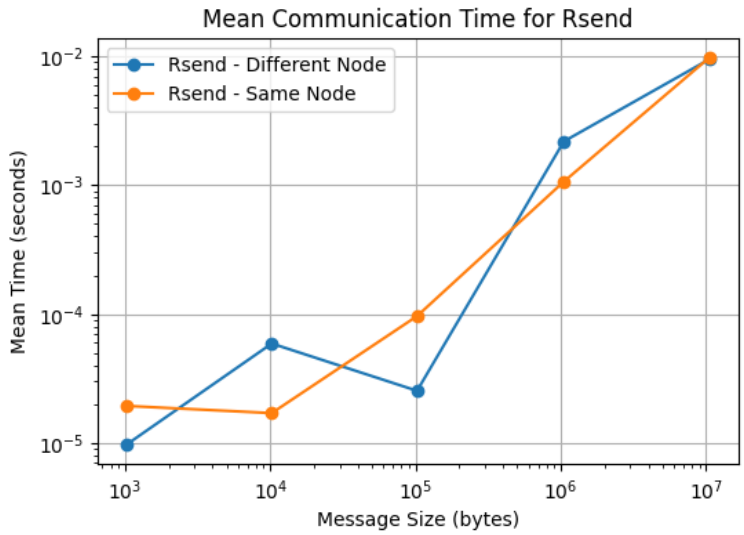
Finalization: The MPI environment is finalized

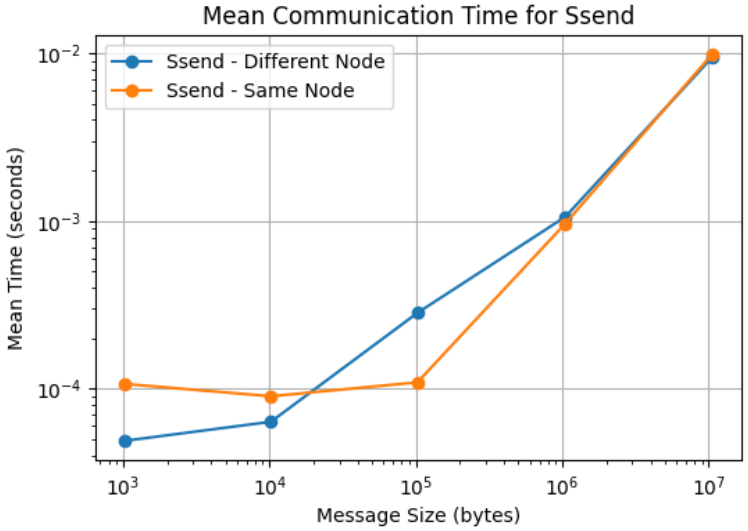
II. Graphical representations of the measured latency/bandwidth - package size relation

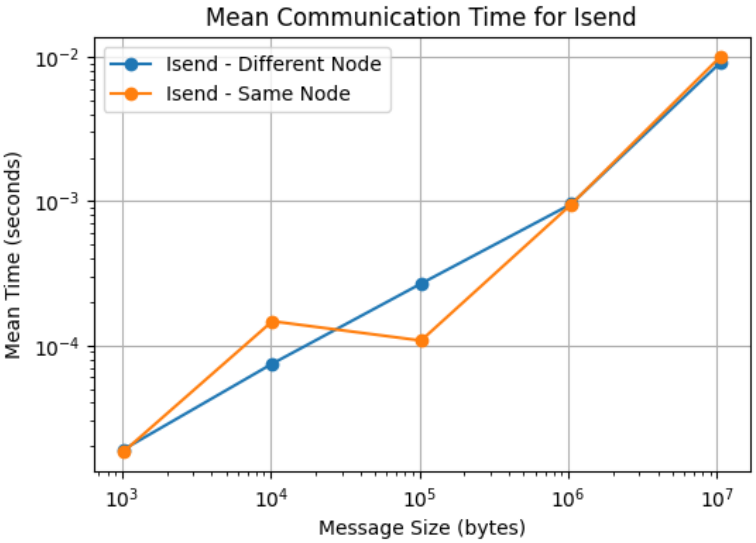
Here are the graphs that record the mean latency time for each MPI send type. They are tested using 2 tasks on 2 different nodes (each node 1 task) and 2 tasks on 1 same node. Here are the result figures plotted on the log scale for both time and package size






Mean Communication Time for Rsend
Rsend - Different Node
Rsend - Same Node
Mean Time (seconds)
Message Size (bytes)


Mean Communication Time for Ssend
Ssend - Different Node
Ssend - Same Node
Mean Time (seconds)
Message Size (bytes)


Mean Communication Time for Isend
Isend - Different Node
Isend - Same Node
Mean Time (seconds)
Message Size (bytes)

III. Short discussion of the results

1. Scaling with message size: For all send types, as the message size increases, the mean communication time also increases. This is expected due to the larger amount of data being transmitted over the network.

2. Different node communication time: Communication times for different nodes are more volatile compared to same node communication. This variability can be due to the increased complexity of network traffic and potential contention when communicating across nodes. Communication within the same node likely benefits from shared memory transports, which are typically faster than network transports used for different node communication.

3. Buffered send behavior: MPI_Bsend shows a relatively consistent pattern of increased time with message size, similar to MPI_Send, suggesting that the buffering mechanism efficiently handles the messages.

4. Ready send performance: MPI_Rsend may show lower times for smaller message sizes, potentially due to requiring the receiver to be ready, thus reducing handshake overhead.

5. Synchronous send overhead: MPI_Ssend may have increased communication time, especially as message size increases, due to the synchronous communication requiring acknowledgment before the send operation completes.

6. Non-blocking efficiency: MPI_Isend together with MPI_Irecv shows a little competitive performance, showing that the non-blocking allows for efficient overlap of communication, particularly for larger messages.