Report Exercise 1
High Performance Computing

Alessandro Della Siega

May 23, 2024

1 Introduction

The OpenMPI library implements several algorithms to perform collective blocking operations according to many different parameters. In order to evaluate the performance of those collective operations, the OSU Micro-Benchmark tool is used.

1.1 Computational architecture

In order to test the performance of the broadcast and reduce operations, we will use a high-performance computing cluster. The computational resources used to complete the exercise are the ones provided by the ORFEO cluster.

The ORFEO system architecture consists of different machines architecture, in this project the THIN partition is used. The THIN partition consist of 12 Intel nodes: two equipped with Xeon Gold 6154 and 10 equipped with Xeon Gold 6126 cpus.

1.2 OSU Micro-Benchmark

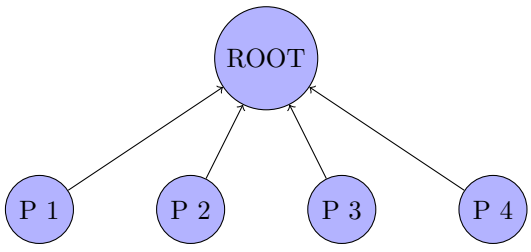
OMB includes benchmarks for various MPI blocking collective operations (MPI_Allgather, MPI_Alltoall, MPI_Allreduce, MPI_Barrier, MPI_Bcast, MPI_Gather, MPI_Reduce, MPI_Reduce_Scatter, MPI_Scatter and vector collectives). These benchmarks work in the following manner. Suppose users run the osu_bcast benchmark with N processes, the benchmark measures the min, max and the average latency of the MPI_Bcast collective operation across N processes, for various message lengths, over a large number of iterations. In our experiments, we will consider the average latency for each message length.

In this report, two operations are take into consideration: broadcast and reduce.

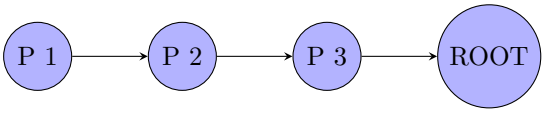
1.3 Broadcast

The broadcast operation `MPI_Bcast` is a one-to-all communication operation which allows a process (typically the root process) to distribute information across many processes. This operation is implemented in OpenMP using different algorithms. The algorithms that are considered in this analysis are:

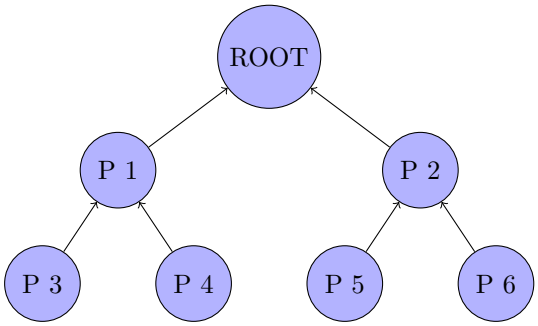
- **basic linear (1)**, the root process sends to all the other process the message;



- **chain (2)**, if we order the processes with respect to their rank, each process sends the message to the next process;



- **split binary tree (4)**, suppose that each process is a node of a binary tree and the root node is the root process, each parent node sends the message to its children. As the name of the algorithm suggests, the message is split in two halves before transmission and when each half of the message reaches its destination, it is necessary to merge the two halves.



1.4 Reduce

The reduce operation `MPI_Reduce` combines the elements provided in the input buffer of each process in the grouping the specified operation, and returns the combined value in the output buffer of the root process. The operations used in the OSU benchmark are sum, maximum, and minimum.

The algorithms that are considered in this analysis are:

- **basic linear (1)**, the root process receives the message from all the other process and reduces the messages;
- **chain (2)**, if we order the processes with respect to their rank, each process sends the message to the next process, each process receives the message from the previous process and reduces the messages;
- **binary (4)**, suppose that each process is a node of a binary tree and the root node is the root process. Each parent node receives the message from its children and reduces the received messages passing the result to its parent node.

2 Implementation

In this section we discuss how the OSU benchmark tool is used to evaluate the performance of the broadcast and reduce operations.

For each operation, we vary the following parameters:

- **algorithm**, the algorithm used to perform the operation;
- **message size**, the size of the message that is sent. It is measured in `MPI_CHAR` unit (1 byte). We will range from 2^1 byte to 2^{20}. We choose this range in order to saturate the osu benchmark capabilities. In fact the maximum message size that can be sent is $1MB$ which is equivalent to 2^{20} bytes.
- **number of processes**, the number of processes that are involved in the operation. This parameter will range from 2 to 48 by step 2. This choice is due to the fact that, according to the THIN architecture, the maximum number of processes that can be used is 48;
- `--map-by`, the mapping of the processes to the hardware. We will consider the following options: `socket`, `core`;

The result is the latency and it is measured in microseconds.

For each run we set 10^4 repetitions and we take the average of the time taken to perform the operation. We also set a warm-up phase of 10^4 repetitions, in order to avoid the cache overhead of the first runs.

3 Results

We have many parameters to take into account when we evaluate the performance of the broadcast and reduce operations. In order to make the analysis easier, we will fix some of them and vary the others.

3.1 Broadcast

By fixing the message size, we notice that the latency scales linearly with the number of processes.

As the size increases, Figure **??**, a substantial positive performance is observed for the binary tree algorithm. And it takes advantage of the fact that the message is divided into smaller parts and sent to the processes in a binary tree fashion. The linear algorithm, on the other hand, has a constant performance, as it sends the message to all processes one by one.

The mapping does not seem to have a significant impact on the performance of the broadcast operation. The performance is almost the same for all the mappings.

It is interesting to notice, as show in Figure 1, that there is a ramp up in the latency when we reach 16 processes and 32 processes. It is not clear why this happens.

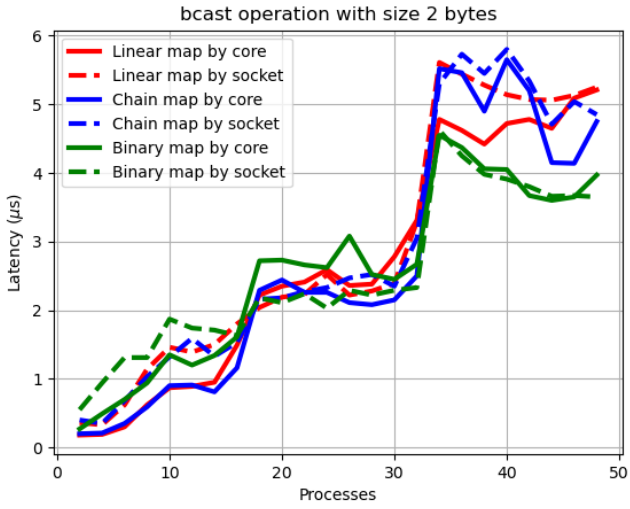


Figure 1: Broadcast 2 bytes fixed message size

Another kind of analysis that we can do is to fix the number of processes and vary the message size. In this case, we can see that the latency scales perfectly linearly with the message size. The binary tree algorithm has a better performance than the linear algorithm, as we can see in Figure **??**.

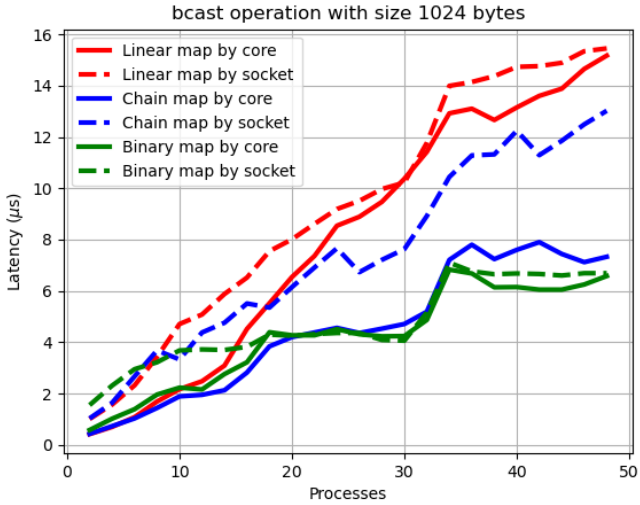


Figure 2: Broadcast 1024 bytes fixed message size

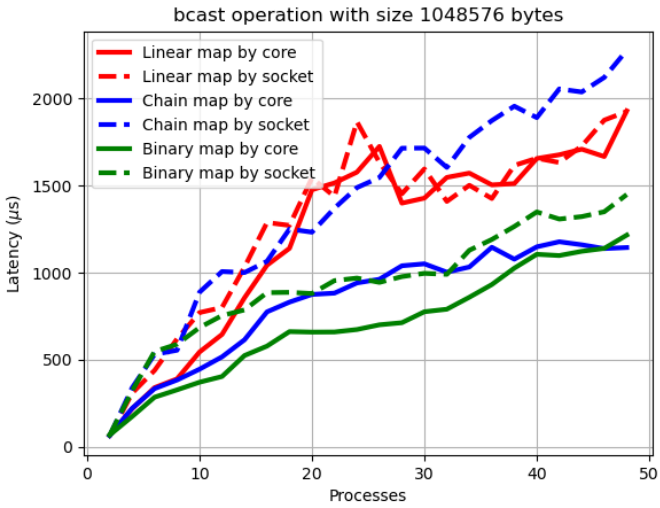


Figure 3: Broadcast 1 megabyte fixed message size

3.2 Reduce

Also in this case, by fixing the size of the message, the latency scales linearly with the number of processes. As noticed in the broadcast operation, the binary tree algorithm has a better performance than the linear algorithm.

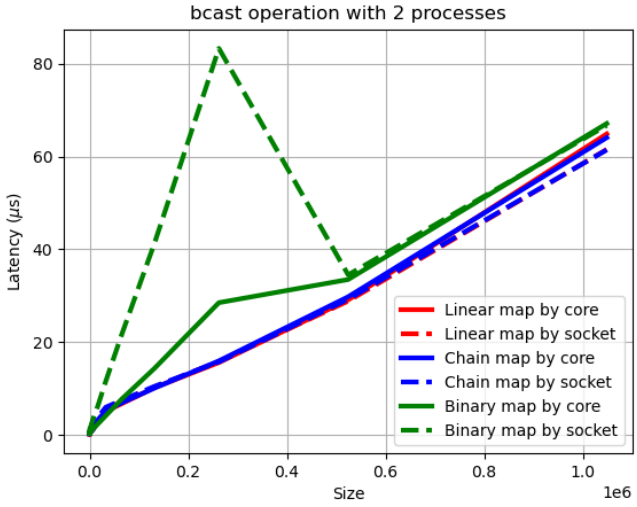


Figure 4: Broadcast with 2 processes

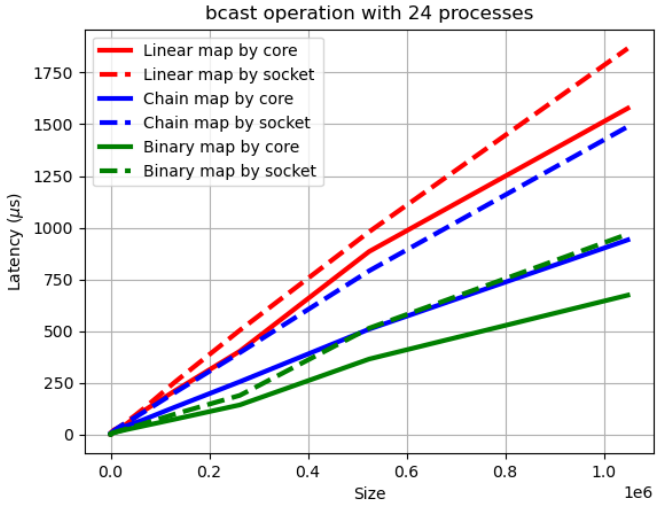


Figure 5: Broadcast with 24 processes

As the message size increases, the mapping has no significant impact on the performance of the reduce operation performed by linear and binary algorithm. However, the chain algorithm performs significantly better using map by node policy for the processes, 6.

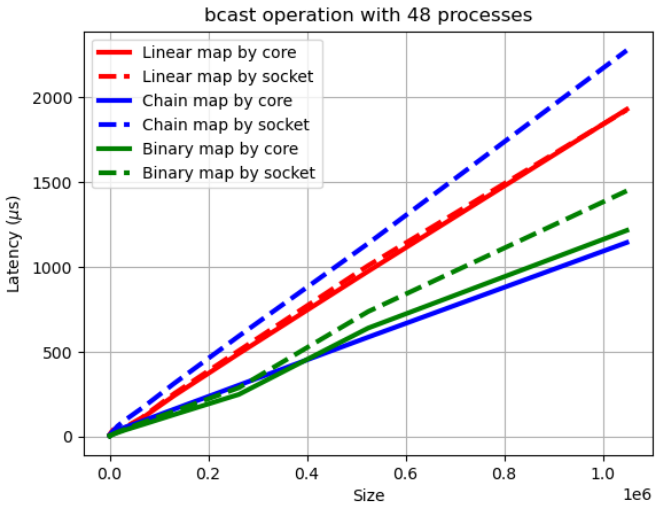


Figure 6: Broadcast with 48 processes

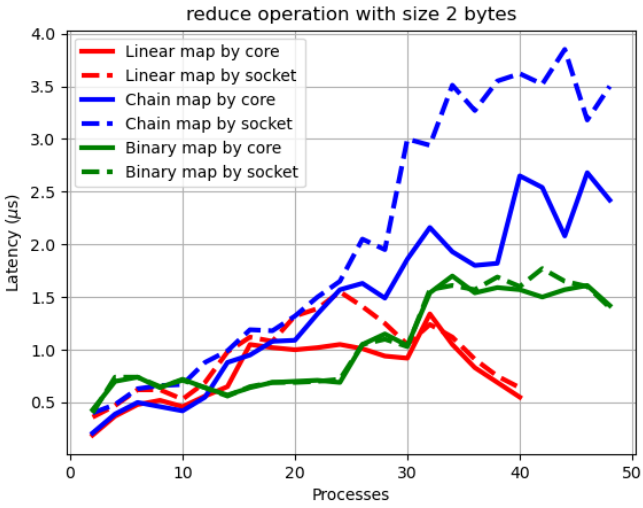


Figure 7: Reduce 2 bytes fixed message size

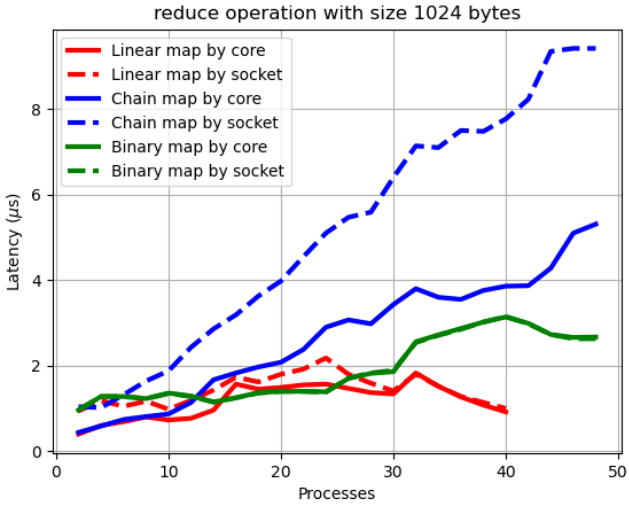


Figure 8: Reduce 1024 bytes fixed message size

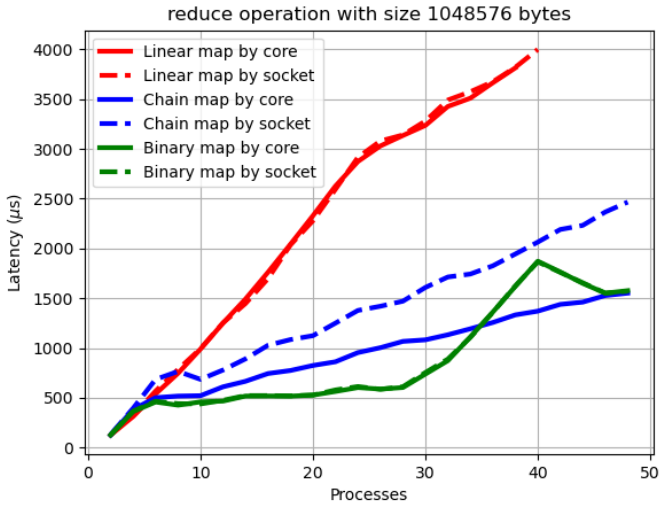


Figure 9: Reduce 1 megabyte fixed message size