Report exercise 1 HPC

Edoardo Zappia

May 2024

1 Introduction

The principal aim of this investigation is to assess the efficacy of the default implementations of broadcast and scatter operations within OpenMPI, with a specific emphasis on diverse algorithms, data dimensions, and process quantities. Our analysis scrutinizes these collective operations and associated algorithms through the employment of the OSU benchmark. In the domain of high-performance computing, a comprehensive comprehension of the effectiveness and efficiency of parallel processing libraries holds paramount importance. This report undertakes an evaluation of the collective operations offered by the OpenMPI library within the context of the ORFEO cluster. OpenMPI affords a spectrum of algorithms tailored for collective operations, thereby granting users the capability to optimize performance contingent upon a multitude of parameters. Within this study, particular attention is directed towards the broadcast and scatter operations, assessing the efficacy of select implementations thereof.

1.1 Algorithms

The algorithms for broadcast implementation analyzed are:

- Basic Linear: Root node sends individual messages linearly to all participating nodes.
- Chain: Each node, except for the root, receives data from its immediate predecessor and forwards it to its immediate successor.
- Binary Tree: Each internal process has two children with sequential data transmission from each node to both children.

The algorithms for scatter implementation analyzed are:

- Default
- Basic Linear: Root node sends individual messages linearly to all participating nodes.
- Binomial: Employs a binomial tree structure for scatter operations.

- Non-blocking Linear: Overlapping receive and send operations, ensuring efficient communication.

These algorithms offer diverse approaches to collective operations, optimizing data transmission based on their inherent structures and characteristics.

1.2 Mappings

- Node:
  Processes are assigned to nodes. All processes requested are distributed evenly across available nodes. Useful for larger parallel applications where processes need to communicate more within the same node than across nodes. This layout improves performance when the communication is minimized between nodes by keeping frequently communicating processes together.

- Socket:
  MPI processes are distributed across sockets (groups of cores in a processor). All processes are assigned to one socket until all its cores are used before switching to the next socket. Beneficial when each socket has its own memory controller to minimize memory contention.

- Core:
  Each MPI process is assigned to a specific core. All cores across nodes are used uniformly until all requested processes are mapped. Ideal when fine-grained parallelism is needed. However, if there are not enough cores available, processes may share the same core.

1.3 Hardware specifications

The evaluation encompassed variations in the number of processes, the size of the messages exchanged, and the mapping of computational resources. To conduct this assessment, we utilized the OSU MPI benchmark, a widely recognized tool for evaluating the performance of MPI implementations. The benchmark was executed on two entire THIN nodes of the ORFEO cluster. The THIN partition comprises 12 Intel nodes: two equipped with Xeon Gold 6154 and 10 equipped with Xeon Gold 6126 CPUs. The primary distinction in the hardware lies in the available random access memory (RAM). Each node contains 24 cores, resulting in a total of 48 cores being utilized for the evaluation.

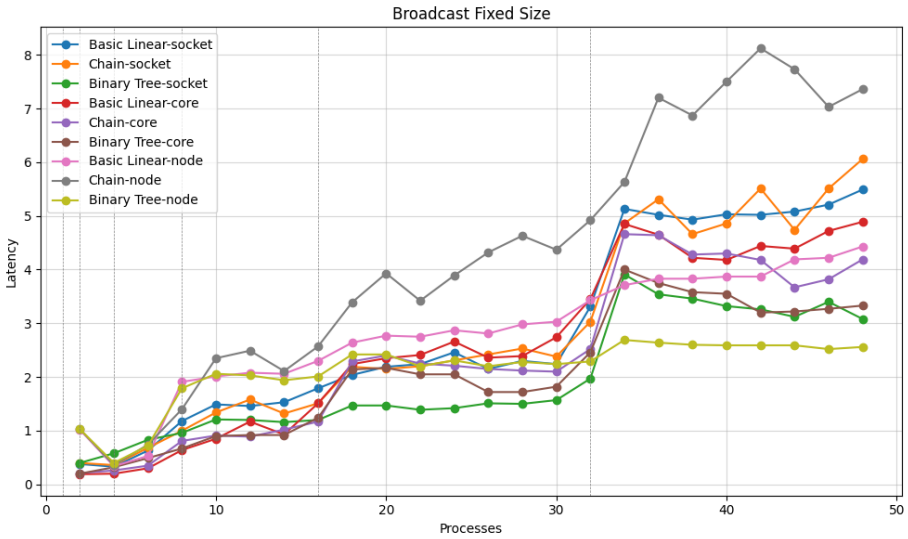
2 Analysis

In this section, we analyze the various experiments conducted and present the linear models utilized for further latency estimation analyses. Initially, we present the results obtained while keeping the size of the sent message constant at 1 byte, followed by those obtained by varying the size with the number of processes.

2.1 Latency with fixed size

The graphs presented here depict the trend of latency time for various MPI implementation algorithms as the number of MPI processes increases. The message size has been set to 1 byte so that it is entirely negligible.

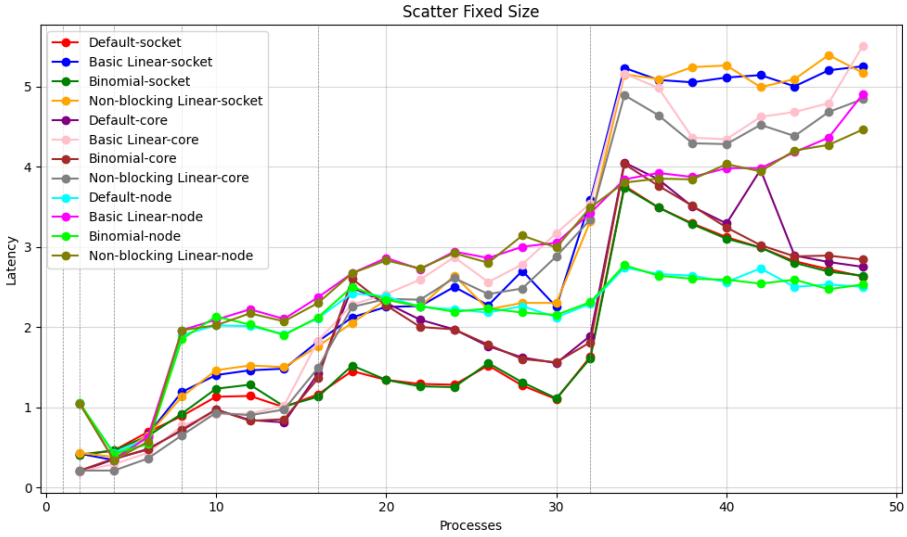
2.1.1 Broadcast



As evident from the graph, the various implementations and mappings have a significant impact on latency. However, a discernible pattern emerges: the Binary Tree implementation appears to outperform others when analyzing algorithms with the same mapping, consistent with theoretical expectations. The performance of the Basic Linear algorithm is comparable to that of the Chain with core and socket mapping. Conversely, when node mapping is employed, a distinct difference in implementation algorithms becomes apparent: the Chain performs worse, while the Binary Tree stands out as markedly superior.

All trends exhibit an anomaly; indeed, one might expect to observe "jumps" at 12 and 24 processes, given that at 12 cores, there is a transition from one socket to another, and at 24 cores, a transition from one node to another. However, from the graph, we can observe clear "jumps" at 16 and 32 cores instead. It thus appears that performance is influenced by factors beyond socket or node changes.

2.1.2 Scatter



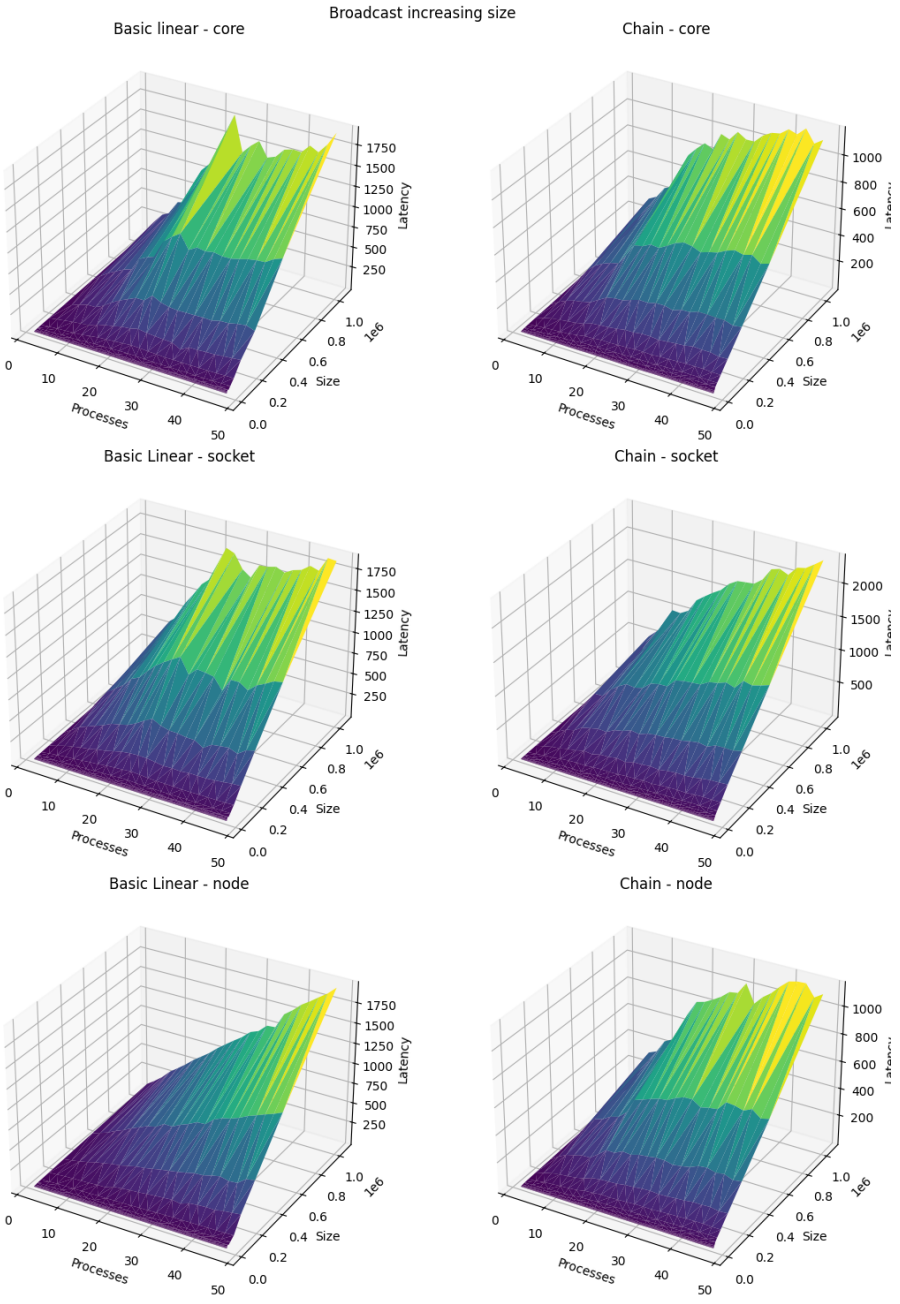
For the scatter operation as well, the various implementations exhibit differences in performance. Here too, a general pattern emerges: when the mapping is held constant, it is observed that the Default and Binomial algorithms produce latency curves that closely resemble each other, whereas the Basic Linear algorithm performs worse across all three different mappings. The Non-blocking Linear algorithm performs similarly to the Basic Linear.

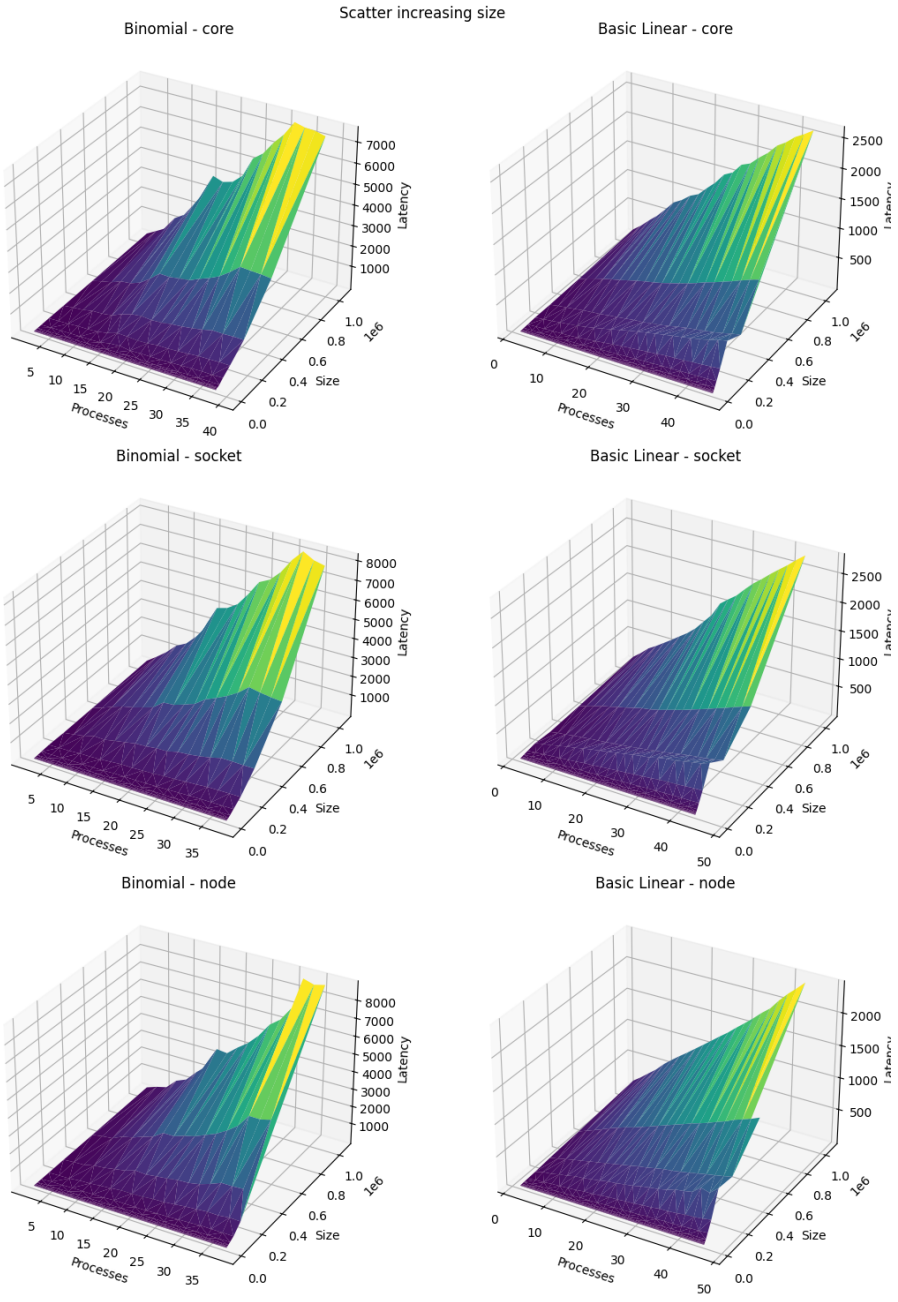
For a moderate number of processes, between 16 and 32, the best-performing algorithms are Binomial and Default with socket mapping. However, for a number of processes greater than 32, the top performers are Binomial and Default with node mapping. The trends of Default and Binomial with different mappings appear to converge as the number of processes increases.

Similarly to the broadcast operation, we can also observe similar "jumps" in the scatter operation. The potential explanation is analogous.

2.2 Latency with increasing size

To create an environment closer to real-world applications, we also examined the variation in latency of the different algorithms as the message size varied.


Broadcast increasing size
Basic linear - core
Chain - core
Basic Linear - socket
Chain - socket
Basic Linear - node
Chain - node
Processes
Size
Latency
1e6



As is easily anticipated, latency increases with message size. Furthermore, it is discernible from the graphs that the number of processes becomes pivotal when the size is larger, and that size significantly impacts performance.

3 Performance Models - Broadcast fixed size

An intriguing idea is to develop a performance model capable of describing the observed trends. Initially, we attempted to utilize the Hockney model, leveraging data collected from the OSU benchmark (using the tool located at /osu-micro-benchmarks-7.3/c/mpi/pt2pt/standard). Unfortunately, the model did not fit well with the empirical data. Therefore, I decided to develop two simple linear models and test them for the various algorithms. A more appropriate approach would be to develop a model for each algorithm individually.

The models developed are:

$$\text{Latency} = \text{const} + \beta_1 \times \text{Number\_Processes} + \beta_2 \times \text{Number\_Processes}^2 \quad (1)$$

$$\log(\text{Latency}) = \text{const} + \beta_1 \times \text{Number\_Processes} + \beta_2 \times \text{Number\_Processes}^2 \quad (2)$$

This analysis has been realized on Broadcast with node mapping.

Table 1: Fixed Size Model 1 - Broadcast Basic Linear

	Coefficient	Std. Error	P-value
Constant	-0.3867	0.274	0.173
Processes	0.1348	0.025	0.000
Processes^2	-0.0005	0.000	0.367
R-squared: 0.942			
Adj. R-squared: 0.937			

Table 2: Fixed Size Model 2 - Broadcast Basic Linear

	Coefficient	Std. Error	P-value
Constant	-1.9058	0.142	0.000
Processes	0.1702	0.013	0.000
Processes^2	-0.0021	0.000	0.000
R-squared: 0.959			
Adj. R-squared: 0.955			

Table 3: Fixed Size Model 1 - Broadcast Chain

	Coefficient	Std. Error	P-value
Constant	-0.3062	0.382	0.432
Processes	0.1259	0.035	0.002
Processes^2	-0.0006	0.001	0.429
R-squared: 0.867			
Adj. R-squared: 0.854			

Table 4: Fixed Size Model 2 - Broadcast Chain

	Coefficient	Std. Error	P-value
Constant	-1.6790	0.159	0.000
Processes	0.1492	0.015	0.000
Processes^2	-0.0018	0.000	0.000
R-squared: 0.939			
Adj. R-squared: 0.933			

Table 5: Fixed Size Model 1 - Broadcast Binary Tree

	Coefficient	Std. Error	P-value
Constant	-0.1865	0.311	0.556
Processes	0.1136	0.029	0.001
Processes^2	-0.0007	0.001	0.213
R-squared: 0.861			
Adj. R-squared: 0.847			

Table 6: Fixed Size Model 2 - Broadcast Binary Tree

	Coefficient	Std. Error	P-value
Constant	-1.5367	0.150	0.000
Processes	0.1350	0.014	0.000
Processes^2	-0.0017	0.000	0.000
R-squared: 0.932			
Adj. R-squared: 0.925			

3.1 Performance Models - Scatter increasing size

After attempting, similar to the broadcast operation, to utilize the Hockney model and obtaining unsatisfactory results, for the scatter operation, we endeavored to develop a model that also took into account the variation in message size. The developed model is

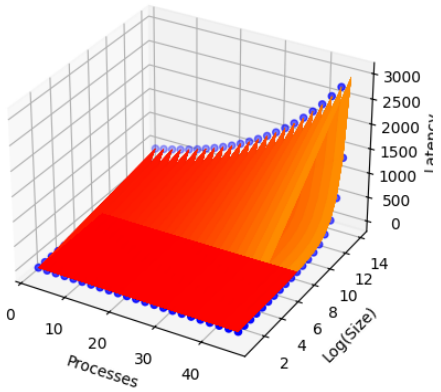
$$\log(\text{Latency}) = \beta_0 + \beta_1 \times \text{Number\_Processes} + \beta_2 \times \log(\text{Size}) + \beta_3 \times \log(\text{Size})^2 \quad (3)$$

The results obtained are

Table 7: Model increasing size - Scatter node

	Coefficient	**Std. Error**	**P-value**
Constant	0.0245	0.056	0.661
Processes	0.0436	0.001	0.000
Log_Size	-0.1875	0.016	0.000
Log_Size_Squared	0.0446	0.001	0.000

R-squared: 0.976
Adj. R-squared: 0.976



4 Conclusion

In summary, our assessment of broadcast and scatter operations on the ORFEO cluster underscores the nuanced interplay among communication patterns, hardware configurations, and algorithmic selections within OpenMPI. We have discerned the significance of contemplating elements such as cluster topology, binding strategies, variations in point-to-point communication latency, and scaling impacts to formulate precise performance models.