High Performance Computing final project
Exercise 1

Barrasso Marco

May 4, 2024

1 Introduction

The aim of this project was to asses the performance of different algorithms for two specific collective MPI operations. The evaluations were conducted on the ORFEO cluster where we utilized the OSU Micro-Benchmarks which is designed to provide a detailed performance analysis of various aspects of HPC networking, offering useful data on the efficiency and scalability of these operations.

The goal was to understand how different algorithms, provided by MPI, influence the latency of collective communications. Moreover, the performance of collective operations is not only determined by the choice of algorithm, also the number of processes, the manner in which they are mapped and the message size play a crucial role. Thus I tried to change the number of processes, the allocation and the message size to see how different mappings influence communication efficiency. All data were obtained utilizing 2 THIN node, setting 5000 warmup iterations and 15000 total iterations.

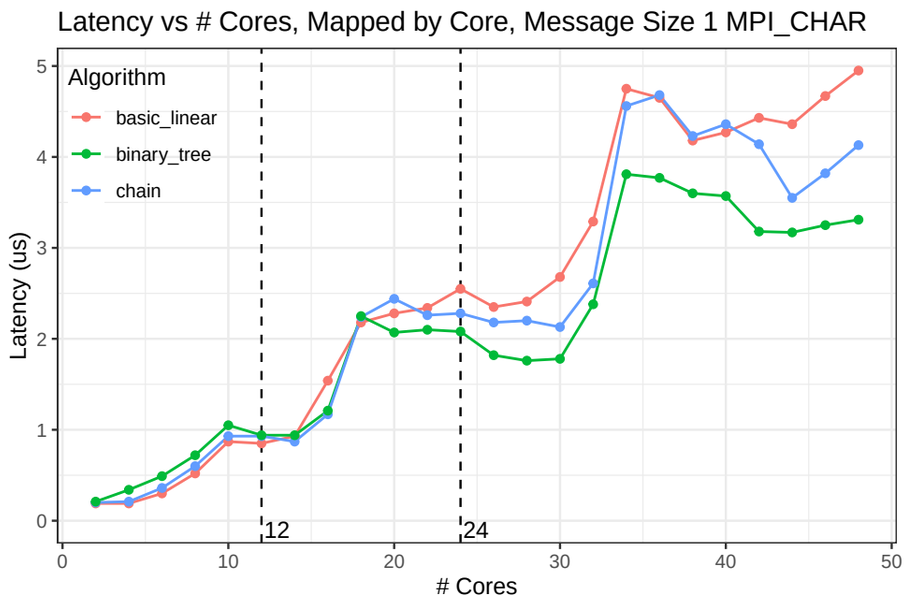
2 Broadcast Operation Analysis

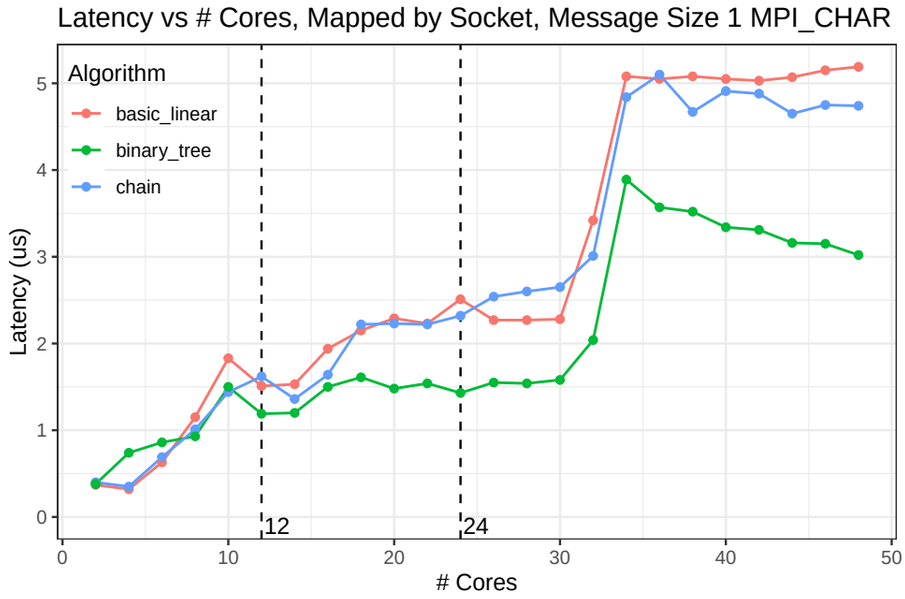
For the broadcast operation we selected three different algorithms: basic linear, chain tree and binary tree.

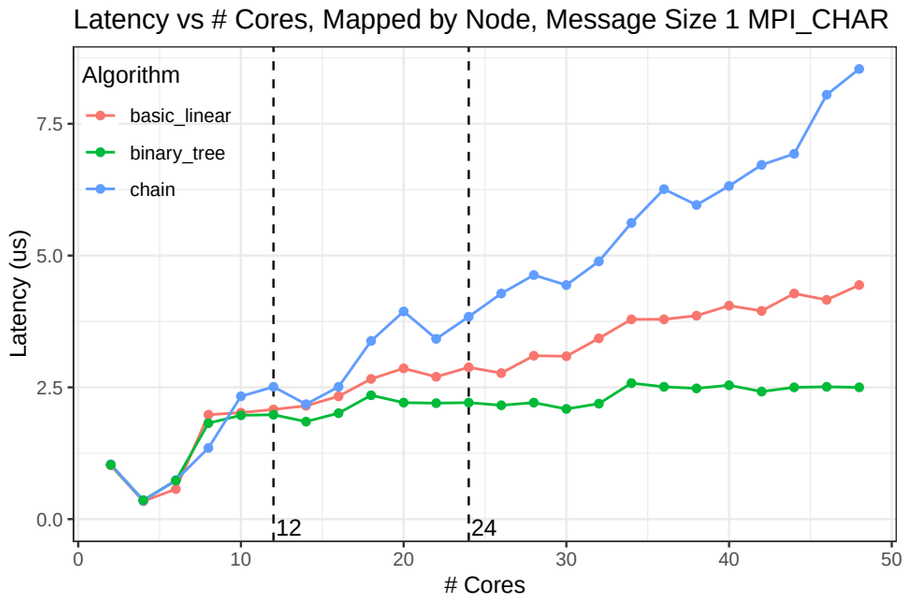
- **Basic linear**: is the simplest form of broadcasting. The root process sends the message directly to each other process in sequence, one after another.
- **Chain Tree**: Each internal node in the topology has one child. The message is split into segments and transmission of segments continues in a pipeline until the last node gets the broadcast message.
- **Binary Tree**: is an approach to distribute the message in a tree-like fashion. Starting from the root, the message is sent to two other processes, which then each become roots of their subtrees and forward the message to two more processes, and so on.

2.1 Different algorithms same allocation type

Below there are the plots reporting the latency(us) varying the number of processes with a fixed messsage size of 1 MPI_CHAR in order to understand the differences between algofithms.


Latency vs # Cores, Mapped by Core, Message Size 1 MPI_CHAR
Algorithm
basic_linear
binary_tree
chain
Latency (us)
Cores
12
24


Latency vs # Cores, Mapped by Socket, Message Size 1 MPI_CHAR
Algorithm
basic_linear
binary_tree
chain
Latency (us)
Cores
12
24


Latency vs # Cores, Mapped by Node, Message Size 1 MPI_CHAR
Algorithm
basic_linear
binary_tree
chain
Latency (us)
Cores
12
24

In our analysis of algorithmic performance across various hardware architectures, both core and socket allocations shows notable latency "jumps" when transitioning across sockets and nodes. Given a hardware configuration featuring sockets with 12 cores and nodes comprising 24 cores, one might expect these jumps to occur at the 12 and 24 process thresholds. Contrary to expectations, these latency increases manifest at 16 and 32 processes, suggesting additional factors beyond simple hardware topology influence communication latency.

From the initial plot, which focuses on core allocation, we observe minimal difference among the three algorithms when communication is confined to a single socket. The distinction becomes more pronounced as we extend communication across the entire node. In this scenario, the binary tree algorithm demonstrates superior performance compared to the linear and chain algorithms, which exhibit similar behaviors. The disparity is further accentuated in inter-node communication scenarios, where the binary tree algorithm distinctly outperforms the others.

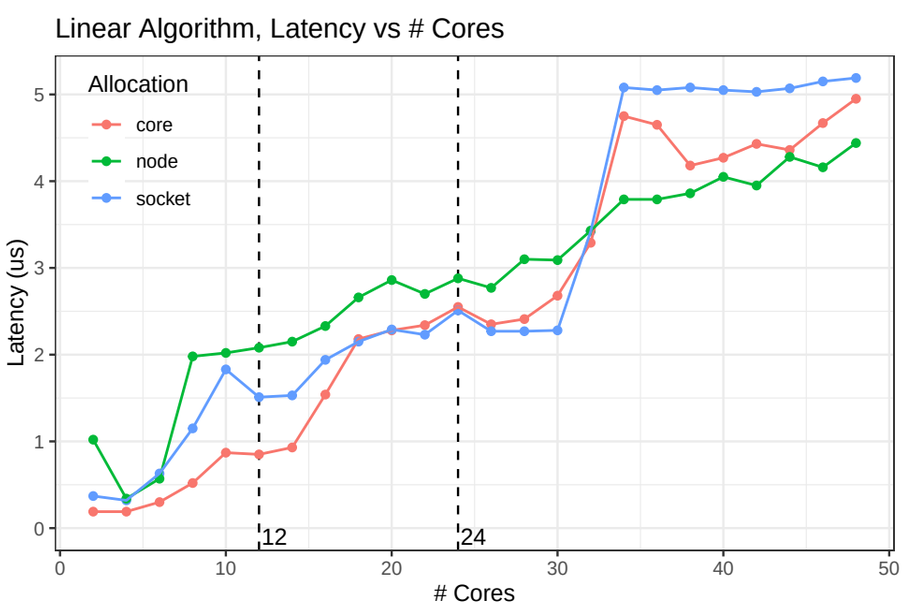
The second plot reinforces these observations, with algorithmic performance relationships remaining pretty similar as before within a single node. The binary tree algorithm slightly outperforms others, a trend that becomes significantly more noticeable in inter-node communication.

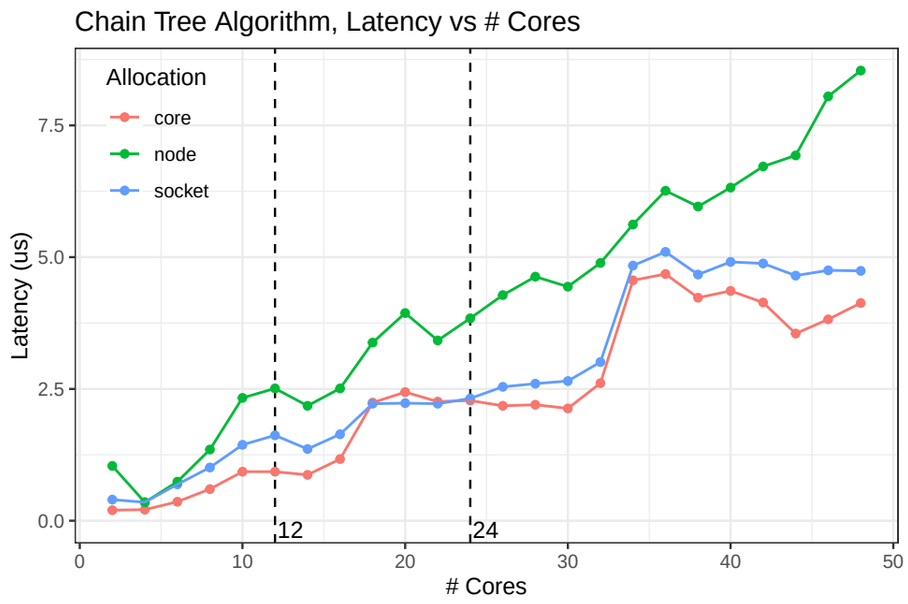
Interestingly, the final plot reveals a departure from previous patterns. Here, the chain algorithm differs markedly from the linear one, exhibiting the worst performance among the three. This outcome is logical when considering the mapping strategy; with node-based mapping, the chain algorithm incurs maximum message traversal distance, leading to elevated latency compared to the linear algorithm.

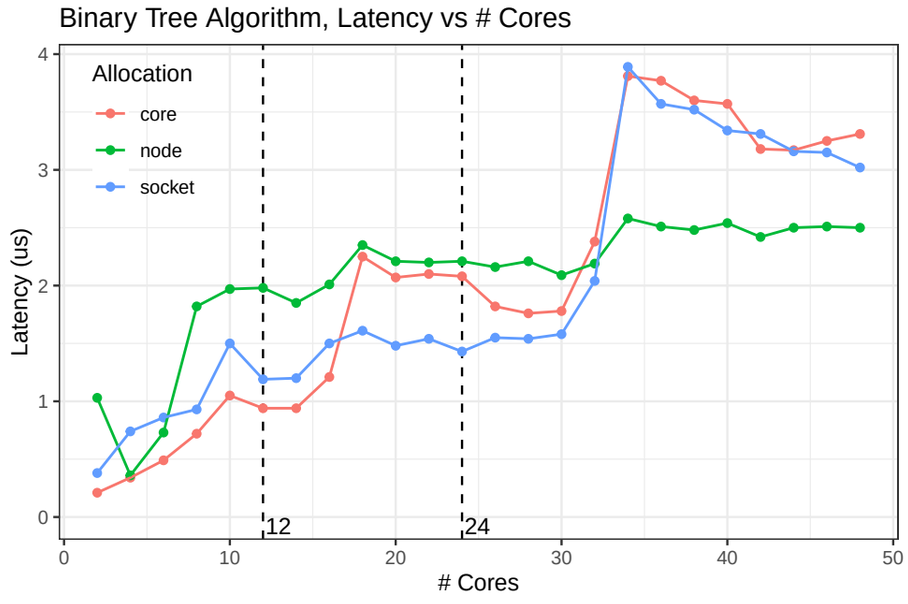
Lastly i wanted to point out that the latency when mapping by core and by socket is pretty similar instead when mapping by node is overall higher. This observation highlights the importance of mapping strategies on communication efficiency in distributed computing environments, emphasizing the need for algorithmic and architectural optimization to mitigate latency in large-scale parallel applications.

2.2 Same algorithm different allocation type

Below i reported plots representing the latency varying the number of processes with a fixed message size of 1 MPI_CHAR in order to understand the impact of the allocation on an algorithm.







From the linear algorithm plot there are similar performance with the different allocations until the communication changes node, after this there is a jump for the core and socket allocation instead for the node one the latency remains pretty stable and gives the best performance. For the chain algorithm instead the behaviour of the core and socket allocation is pretty similar instead we can see that from the beginning the worst performance is given by the node allocation for the same reason discussed in the previous chapter. Lastly the y axis of the third plot shows lower latency values associated with the binary tree algorithm, showcasing, as expected, its superior performance compared with the two algorithms analyzed above.

2.3 Modelling Broadcast Performance

After conducting an analysis with various numbers of cores, I modified the message size and developed a simple broadcast model for comparing three different algorithms. The analysis involved fitting a linear model, using message size and number of processes as predictors, with the allocation type fixed to cores. The transformations $\log_2(\text{Latency})$ and $\log_2(\text{Message Size})$ were applied to reduce data skewness. The initial model was defined as:

$$\log_2(\text{Latency}) = \beta_1 \cdot \text{Number of Processes} + \beta_2 \cdot \log_2(\text{Message Size})$$

Although both coefficients were statistically significant and the adjusted R-squared (R^2_{adj}) was relatively high, visual inspection of the residuals suggested a non-linear relationship between $\log_2(\text{Message Size})$ and $\log_2(\text{Latency})$. To address this, a quadratic term was introduced to the model, resulting in the following equation:

$$\log_2(\text{Latency}) = \beta_1 \cdot \text{Number of Processes} + \beta_2 \cdot \log_2(\text{Message Size}) + \beta_3 \cdot \log_2(\text{Message Size})^2$$

In the following section, I present a 3D plot illustrating the $\log_2(\text{Latency})$ with respect to the number of processes and $\log_2(\text{Message Size})$. The plot refers to the linear algorithm, the other are omitted as they showed similar patterns. This section also includes a detailed comparison of coefficient estimates and the adjusted R-squared values for each algorithm, highlighting how each model's complexity affects the explanatory power and predictive accuracy.

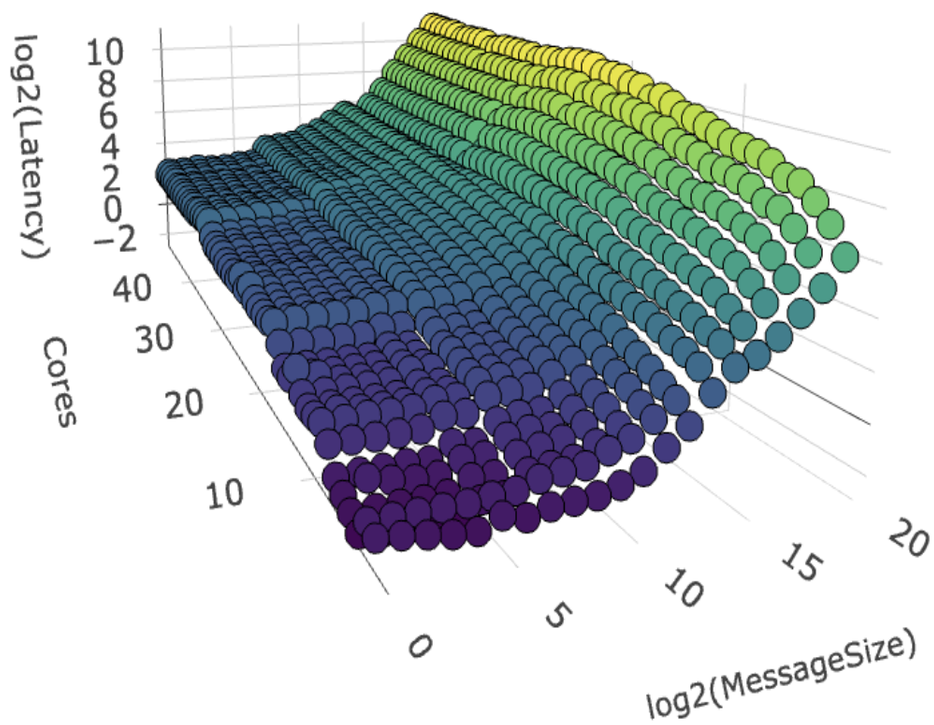


Figure 1: Linear Algorithm $\log_2(\text{Latency})$ w.r.t number of processes and $\log_2(\text{Message Size})$, –**map-by core**.

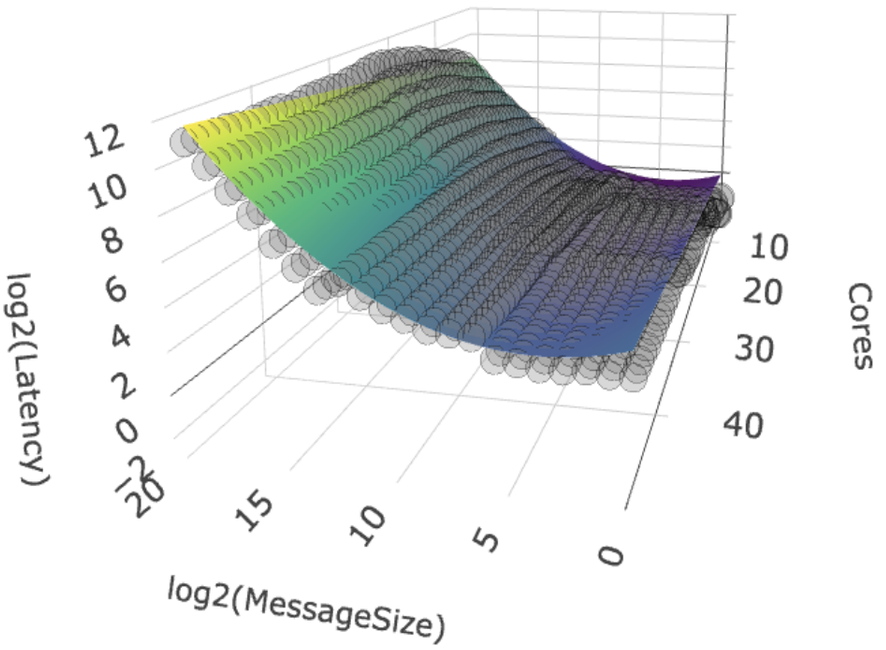


Figure 2: Estimated Regression Plane

Linear	**Estimate**	**Std. Error**	**P-value**	$\mathbf{R^2_{adj}}$
Processes	0.0703302	0.0013204	$<$ 2e-16	
$\log_2$(Message Size)	-0.2895294	0.0098068	$<$ 2e-16	
$\log_2(\text{Message Size})^2$	0.0350895	0.0005347	$<$ 2e-16	97,94 %

Table 1: Table presenting statistics of the model for the Linear algorithm

Chain	**Estimate**	**Std. Error**	**P-value**	$\mathbf{R^2_{adj}}$
Processes	0.0634918	0.0011072	$<$ 2e-16	
$\log_2$(Message Size)	-0.3145670	0.0082232	$<$ 2e-16	
$\log_2(\text{Message Size})^2$	0.0357085	0.0004484	$<$ 2e-16	98,32 %

Table 2: Table presenting statistics of the model for the Chain algorithm

Binary Tree	**Estimate**	**Std. Error**	**P-value**	$\mathbf{R^2_{adj}}$
Processes	0.0592922	0.0009693	$<$ 2e-16	
$\log_2$(Message Size)	-0.3221704	0.0071993	$<$ 2e-16	
$\log_2(\text{Message Size})^2$	0.0366723	0.0003925	$<$ 2e-16	98,71 %

Table 3: Table presenting statistics of the model for Binary Tree algorithm

The comprehensive analysis and subsequent modeling adjustments reveal significant insights into the performance dynamics of broadcast algorithms under varying conditions of message size and number of processes. The introduction of a quadratic term to the regression model was not merely a statistical adjustment but a necessary enhancement to capture the underlying patterns observed in the data.

From the 3D visualization of the $\log_2$(Latency), it is evident that the relationship between $\log_2$(Message Size) and latency is not linear but exhibits a curvature indicative of a quadratic influence. This visual observation aligns well with the statistical analysis, where the quadratic term associated with $\log_2(\text{Message Size})^2$ proved to be statistically significant.

In conclusion the model seems to describe quite well the relations between variables obtaining an R^2_{adj} really high for all of the three models. I wanted to say that the type of algorithm could have also been treated as a categorical variable but i didn't want to introduce more complexity to the model.

3 Gather Operation Analysis

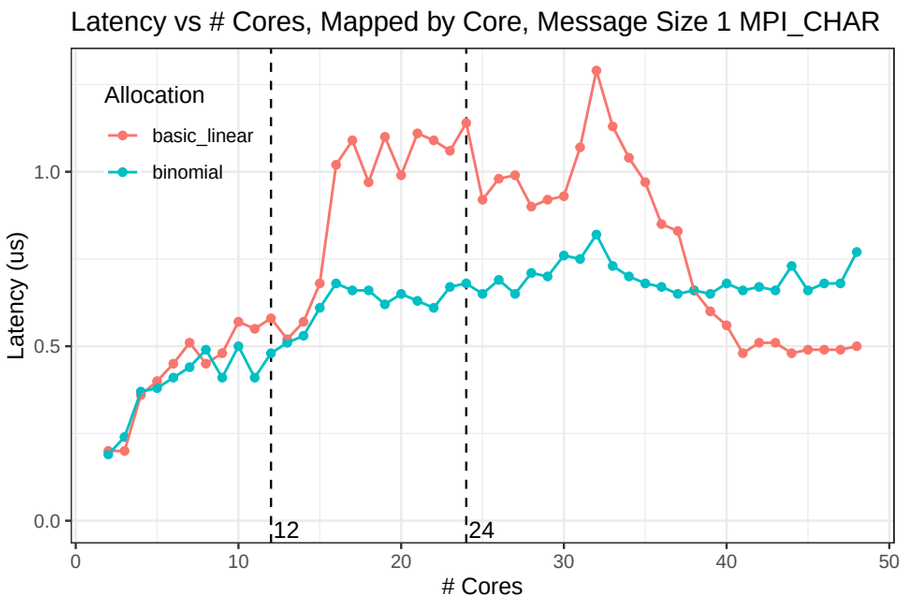
The analytical approach utilized for Gather closely follows the methodology applied in section 2 for Broadcast, ensuring consistency and allowing us to focus more directly on the results.
The Gather operation is one of the collective communication mechanism that involves collecting data from all processes in a group and assembling it in one single process, typically known as the root process. Each participating process (including the root) sends the contents of its send-buffer to the root process. The root process then collects these messages and organizes them into the order of their source ranks within its receive-buffer.
For the gather operation i selected two different algorithms: linear and binomial tree.

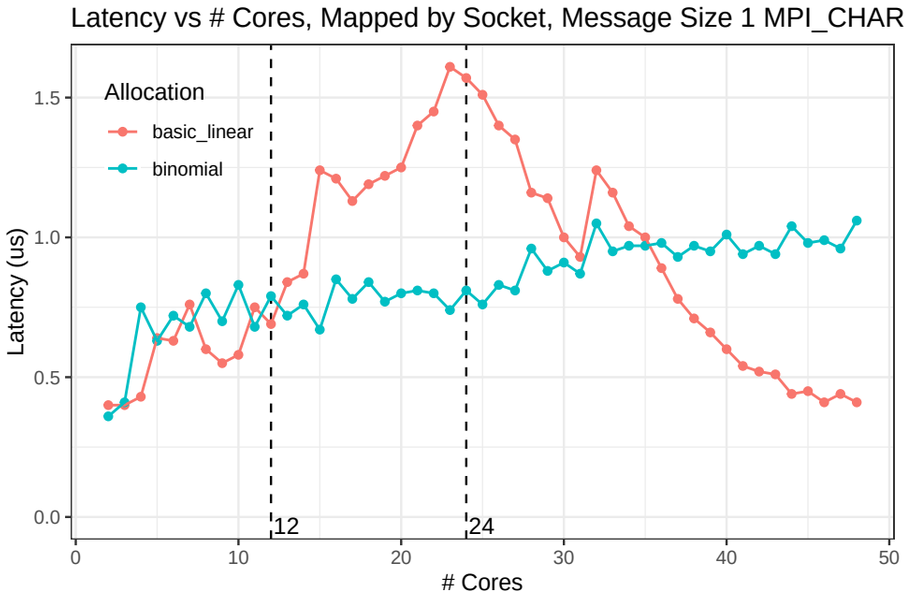
- **Basic linear**: the linear algorithm for the gather is straightforward, each process sends its data directly to the root process one by one.
- **Binomial Tree**: is more complex but generally more efficient in many scenarios, this algorithm uses a tree-based structure where communication is organized in a hierarchical manner.

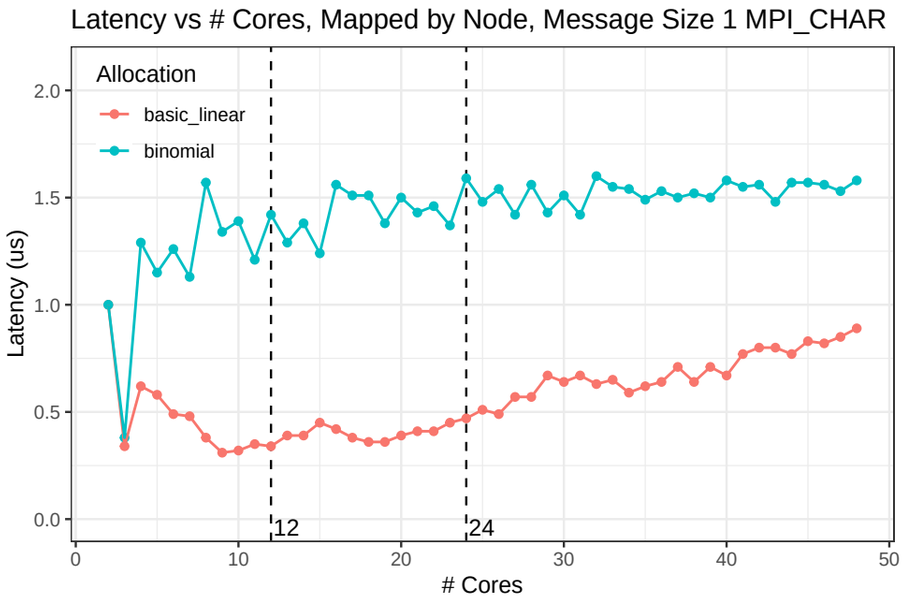
There was also the possibility to select a third algorithm, the linear with synchronization but after taking a look at the data its performance was way different from the other two so i decided to focus my attention on the comparison of the linear and binomial.
So as done in the section before the analysis is going to compare the performance of the two different algorithm fixing the allocation type.

3.1 Different algorithms same allocation type

Below there are the plots reporting the latency(us) varying the number of processes with a fixed messsage size of 1 MPI_CHAR in order to understand the differences between algorithms.


Latency vs # Cores, Mapped by Core, Message Size 1 MPI_CHAR
Allocation
basic_linear
binomial
Latency (us)
0.0
0.5
1.0
12
24
0
10
20
30
40
50
Cores


Latency vs # Cores, Mapped by Socket, Message Size 1 MPI_CHAR
Allocation
basic_linear
binomial
Latency (us)
0.0
0.5
1.0
1.5
12
24
0
10
20
30
40
50
Cores

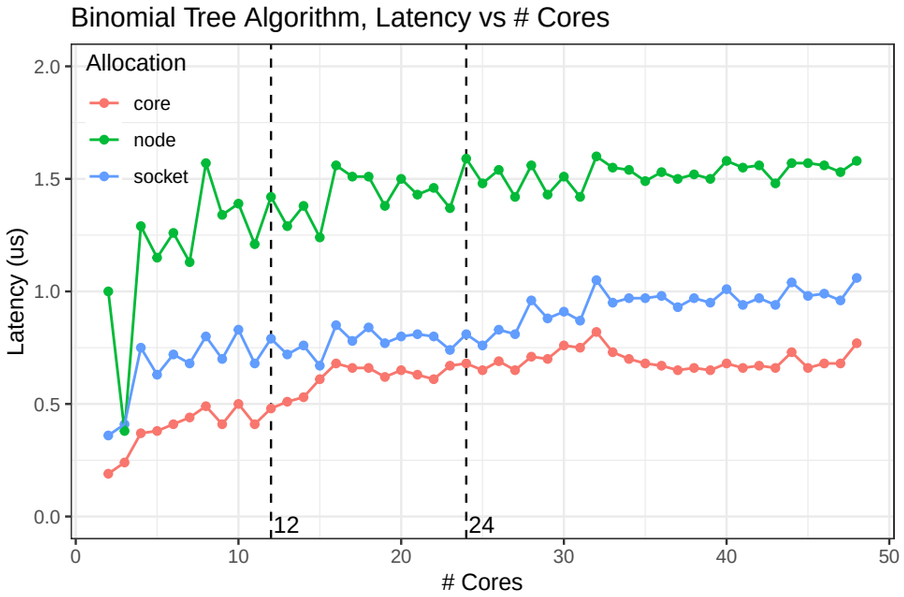

Latency vs # Cores, Mapped by Node, Message Size 1 MPI_CHAR
Allocation
basic_linear
binomial
Latency (us)
0.0
0.5
1.0
1.5
2.0
12
24
0
10
20
30
40
50
Cores

From the previous plots we can see that the linear algorithm in the case of core and socket mapping present a strange behaviour, the latency increase before 24 cores, as expected, but start to decrease when changing node, it suggests the existence of additional factor influencing the communication. The binomial algorithm instead seems to perform reasonably according to our expectation, with a logarithmic behaviour.

For the node mapping instead we can see that the linear model outperforms the binomial one, this could be due to the fact that the binomial algorithm utilizes a more complex communication pattern that involves setting up a tree structure. While this is usually efficient for large messages or a large number of processes due to reduced step complexity, the overhead of setting up and managing this tree can be disproportionately high for very small messages.This analysis shows that the optimal choice of algorithm can vary based on the specifics of the situation.

3.2 Same algorithm different allocation type

Since the linear algorithm showed an unexpected behaviour which i couldn't actually explain from now on we are just going to focus on the binomial algorithm, and in this section we are gonna compare it with different type of allocation.



The binary tree algorithm behave as expected with respect to different allocation type, the best performance is achieved by the core mapping, followed by the socket one. The node mapping force the communication to change node continuously leading to the worst performance.

3.3 Modelling Gather Performance

As before i collected the data changing also the message size in order to better describe the performance of the gather operation. The proposed model is the same as the one in the previous section:

$$\log_2(\text{Latency}) = \beta_1 \cdot \text{Number of Processes} + \beta_2 \cdot \log_2(\text{Message Size}) + \beta_3 \cdot \log_2(\text{Message Size})^2$$

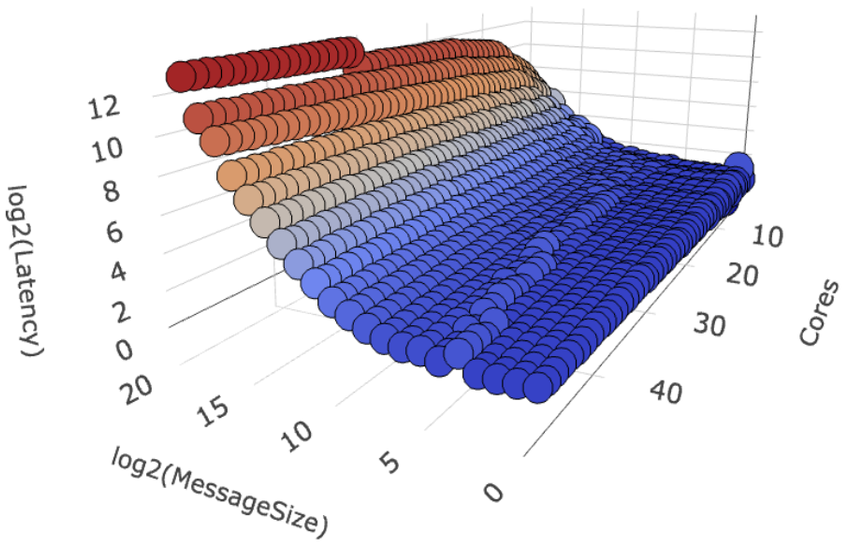


Figure 3: Binomial algorithm plot, –**map-by node**

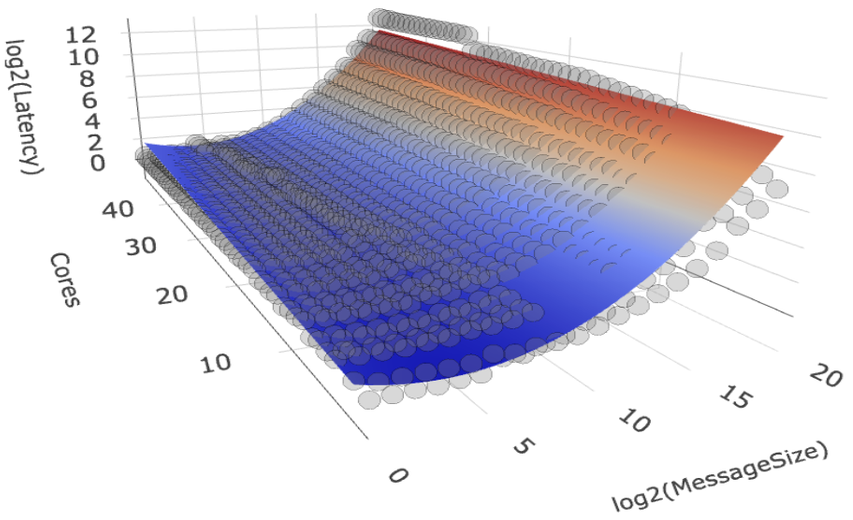


Figure 4: Estimated Regression Plane

Binomial Tree	**Estimate**	**Std. Error**	**P-value**	$\mathbf{R^2_{adj}}$
Processes	0.0394438	0.0011324	$<$ 2e-16	
$\log_2$(Message Size)	-0.3175077	0.0084104	$<$ 2e-16	
$\log_2$(Message Size)^2	0.0402835	0.0004586	$<$ 2e-16	98,47 %

The plots for the gather operation show notable similarities to those of the broadcast operation but with key differences. Firstly, the y-axis of the gather plots indicates higher latency in the gather operation compared to the broadcast. Upon examination of the data presented in previous tables, we can notice that the contribution of individual processes to the overall latency is less pronounced in the gather operation. This suggests a different dynamic in how processes affect the overall performance in gather scenarios.

The coefficients associated with $\log_2$(Message Size) are nearly identical between the two operations, indicating a consistent impact of message size on latency across both types of operations. However, the influence of $\log_2$(Message Size)^2 is greater in the gather operation, highlighting a more pronounced non-linear relationship between message size and latency as the operation complexity increases.

Overall, the model for the gather operation performs pretty well. All coefficients are statistically significant, which suggests that the model is robust and the predictors are well chosen. The high adjusted R-squared (R^2_{adj}) value confirms that a substantial proportion of the variance in latency is explained by this model.

References

[1] Model-based selection of optimal MPI broadcast algorithms for multi-core clusters URL https://www.sciencedirect.com/science/article/pii/S0743731522000697?via%3Dihub

[2] MVAPICH: MPI over InfiniBand, Omni-Path, Ethernet/iWARP, RoCE, and Slingshot https://mvapich.cse.ohio-state.edu/benchmarks/

[3] AreasSciencePark . Orfeo-doc 2023. URL https://orfeo-doc.areasciencepark.it

[4] Open MPI Documentation https://www.open-mpi.org/doc/

[5] Marco Barrasso, HPC Repository https://github.com/marcobarrasso1/HPC_Project