# HPC - Exercise 1

Sara Carpenè

February 2024

## 1 Introduction

This exercise aims to estimate the latency of different openMPI implementation available in the OSU Benchmark varying the number of processes and the size of the messages. The chosen algorithm are broadcast and scatter. Broadcast has been run on two EPYC nodes on ORFEO HPC cluster, every EPYC node in ORFEO has 2x64 cores. The number of processors taken into account in collecting performance numbers vary from 2 to 256. The message size varies from 2 byte to 1MB. Scatter has been run on two THIN nodes on the same cluster; every THIN node in ORFEO has 2x12 cores. In this case the number of processes taken into account varies from 2 to 48, and the size of the message from 2 byte to 1MB.

## 2 Description of the collecting strategy

The data of the broadcast algorithm were collected using:

```
mpirun --map-by core -np $processes --mca coll_tuned_use_dynamic_rules true --mca
coll_tuned_bcast_algorithm  n_alg osu_bcast -m $size -x $repetitions -i $repetitions
```

The data of the scatter were collected using:

```
mpirun --map-by core -np $processes --mca coll_tuned_use_dynamic_rules true --mca
coll_tuned_scatter_algorithm n_alg osu_scatter -m $size -x $repetitions -i $repetitions
```

It has been collected data considering the *–map-by-core* flag.

## 3 Broadcast

In the broadcast operation a process called root sends a message with the same data to all processes in the communicator. The algorithms that have been chosen are 1 for the baseline and 4,5,6 as study cases.

### 3.1 Basic linear algorithm (b1)

In this algorithm, for N processes, the root sends the message to the N-1 other processes.This algorithm is used as a baseline to compare the latency of other algorithms.

The data shows that as the size of the transmitted message increases, so does the latency. This is a common characteristic of broadcast algorithms: when the amount of data being sent and received increases, the time it takes to complete the operation increases.

For instance, when the message size increases from 2 to 4 bytes, the latency slightly increases from 0.13 to 0.14 microseconds. However, when the message size reaches 1048576 bytes (1 MB), the latency significantly jumps to 78.62 microseconds. This suggests that there may be a threshold beyond which the algorithm is no longer efficient or scalable.

The number of processes involved also affects latency. It seems that the latency increases with the number of processors until 128 processors (which corresponds to an entire EPYC node), and then the growth becomes less significant or stabilizes.

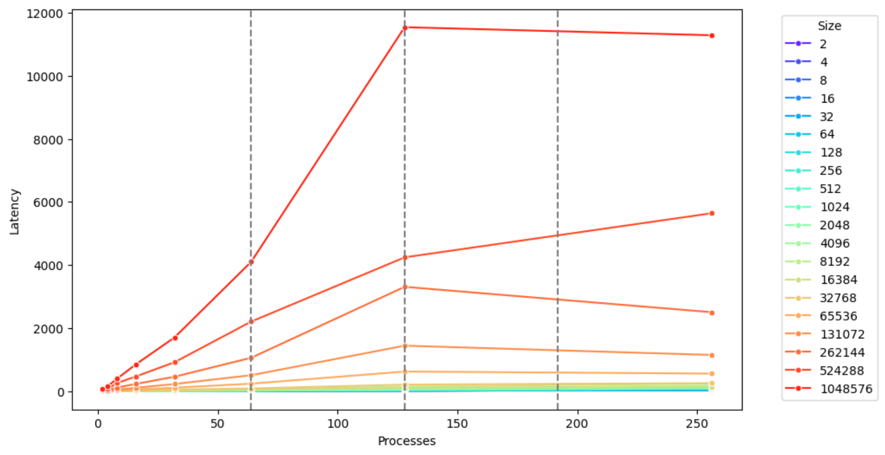


Figure 1: Basic linear broadcast (b1)

In figure 1 it has been plotted the latency versus the numbers of processes for different size of the message.

By watching the data collected with a fixed size of 2 byte it can be noted that there is a slight decrease of the latency for 64, 128, 192, 256 processors. This can be related to the architecture on which this algorithms are tested, thus the numbers correspond to the saturation of one of the socket inside the two nodes, or it can be due to an optimization of the algorithm for this specific value.

To build a model that aims to estimate the effective latency of the broadcast algorithm while excluding, as much as possible, the time to transmit the message, the size of the message has been fixed to 2.

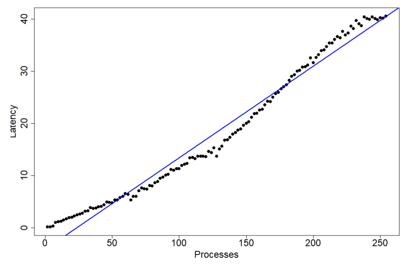


Figure 2: Basic linear model (b1)

The plotted model is a linear model of the form

$$\text{latency} = -4.16 + 0.18\text{processes}$$

with an adjusted R-squared of 0.9761. It strikes a good balance between being a well-fitting model for the data without overfitting, and maintaining interpretability. It provides insights to the expected latency given the number of processes.

*NOTE: For the following figures related to broadcast it has been plotted in light grey the linear algorithm for more immediate comparison.*

## 3.2 Binary tree algorithm (b5)

In this algorithm the root sends the message to two processes that are identified as children of the root, then each internal process sends the message to its two child processes, continuing this pattern until reaching the leaf node, which has no child processes to send the message to.

For this algorithm the latency grows faster when the size of the message increases, rather than when the number of processes increase.

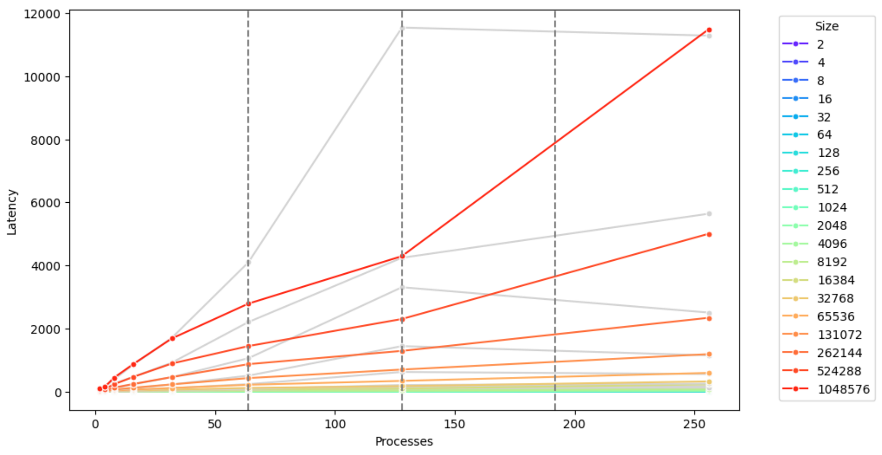


Figure 3: Binary tree broadcast (b5)

With respect to the number of processes the data suggests that the algorithm scales well with an increase of the number of processes, up to a certain threshold. Beyond this threshold, latency begins to rise significantly, suggesting a scalability limit for the algorithm. It's important to note that this steep increase in latency commences at 128 processes, which aligns with the saturation of the first EPYC node.

By comparing the results with the ones of the linear algorithm it can be seen that the last one tends to have lower latencies than the binary tree broadcast when dealing with larger message sizes and small number of processes involved and they are competitive for quite big message size and big number of processes.

For the other cases the binary tree broadcast performs better than the linear one in terms of latency, especially for small size of the messages. It has been fitted a model that depends on the cube of the number of processes. The resulting formula is:

$$latency = 1.997 + 9.034e^{-7}processes^3$$

the model has an R squared adjusted of 0.9695.

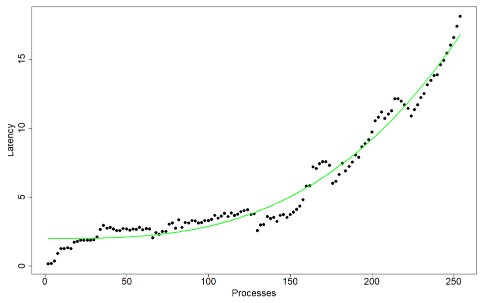


Figure 4: Binary tree broadcast (b5)

## 3.3 Split binary tree (b4)

This algorithm organizes the message transmission like a binary tree. The root sends the message to two processes, every internal process has two children. Before starting the communication the message is split into two half, the left part of the tree propagates the left part of the message, same for the right part. Then in the last step the right and left nodes exchange their half of the messages in order to complete the broadcast operation.

Also for this algorithm we can notice an improvement with respect to the linear algorithm especially for huge size of processes with small sized messages. For example as the number of processes increases from 2 to 32, the latency for a 2-byte message in the binary tree remains practically unchanged (from 0.20 to 0.27), while in the linear algorithm it increases significantly (from 0.13 to 2.99). This pattern is less pronounced but can still be observed for larger messages. More in general as the number of processes increases, the linear broadcast latency increases more steeply than the binary tree broadcast.

It can be noted that for big sized messages and small number of processors (lower than 16) the linear broadcast algorithm performs better than the split binary tree, maybe this can be due to a higher communication overhead, which can happen with tree-based algorithms.

The binary tree algorithm appears to be more efficient than the linear broadcast for larger data sizes and a higher number of processes. This is likely due to the tree's ability to distribute the workload more evenly across the processes, which can lead to better scalability; but the linear algorithm performs better in simpler situation with small size of processes and messages.

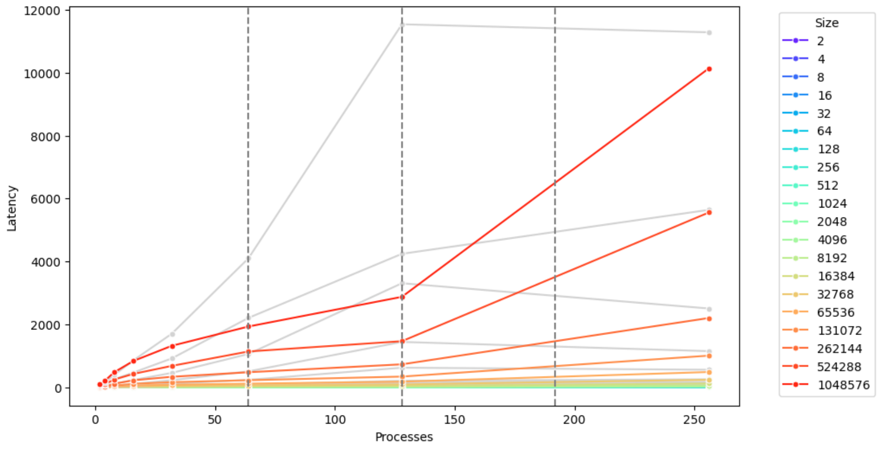


Figure 5: Broadcast 4

For the split binary tree we can notice that the latency seems to slowly increase till 128 processes, and the growth becomes faster, similarly with what we observed in the binary tree algorithm. By comparing the two graphics of split binary tree and binary tree it can be observed that the main improvement of the spitting approach is obtained for big size and small number of processes.

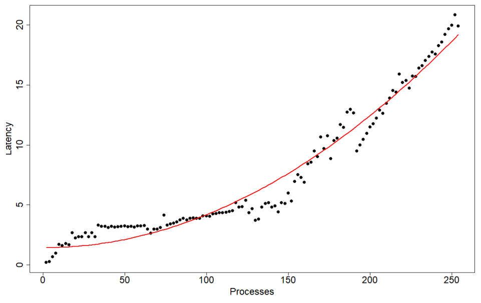


Figure 6: Spit binary tree broadcast (b4)

It has been fitted a model that depends on the square of the number of processes. The resulting formula is:

$$latency = 1.441 + 2.750e^{-4} processes^2$$

the model has an R squared adjusted of 0.9628.

## 3.4 Binomial tree algorithm (b6)

Unlike the binary tree the maximum nodal degree of the binomial tree decreases from the root down to the leafs. At each step the whole message is sent at once. Each node that has already received the message sends it further on. This allows the tree to grow exponentially as at each step the number of nodes that send the message is doubled. For this algorithm, latency tends

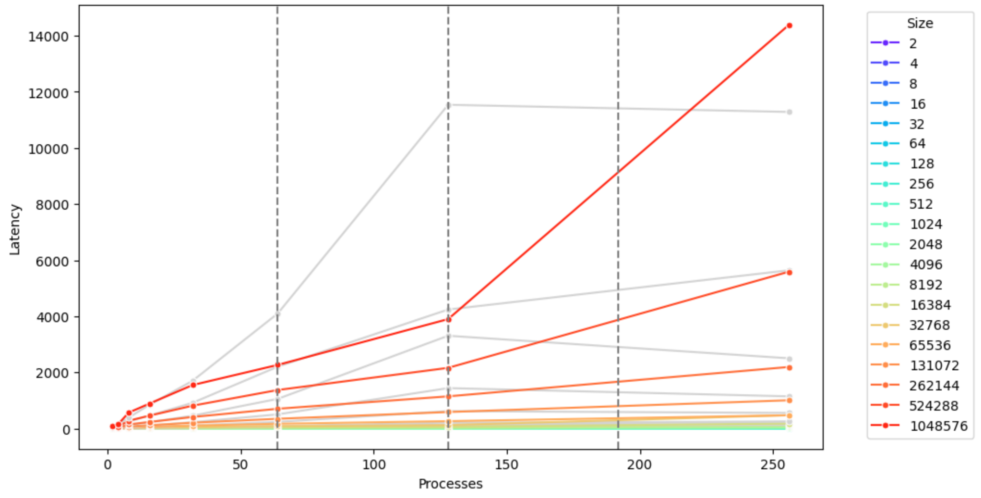


Figure 7: Broadcast 6

to increase with the size of the data.

This is an expected behaviour as the algorithm has to propagate the data through a tree structure, which requires more communication steps for larger data sizes. The binomial tree algorithm shows a more pronounced increase in latency as the processes count increases, which could be due to the overhead of managing the tree structure at scale.

By comparing binomial tree latency with the one of linear broadcast we can notice that the last one tends to be lower for larger data sizes and small processes count, and that they are competitive for big data size and processes count.

For smaller data sizes and process counts, the latency is relatively low and similar for both algorithms. The main advantage of using the binomial tree is reached for small data size (2- 32) and huge number of processors; for example for a two sized message when going from 2 to 256 processor the linear algorithm increase of (38.22-0.13=38.09) when the binomial tree algorithm increases of (4,57-0.15=4.43).

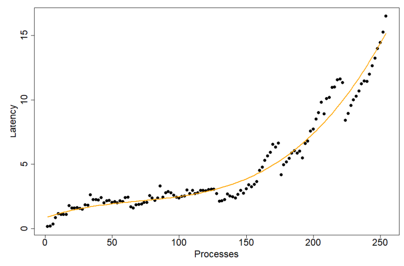


Figure 8: Binomial tree broadcast (b6)

It has been fitted a model that depends on power of number of processes up to degree. The resulting formula is:

$$latency = 8.254e^{-1} + 3.620e^{-2}proc$$

$$-3.745e^{4}proc^{2} + 1.789e^{-6}proc^{3}$$

the model has an R squared adjusted of 0.956.

## 3.5 Conclusion

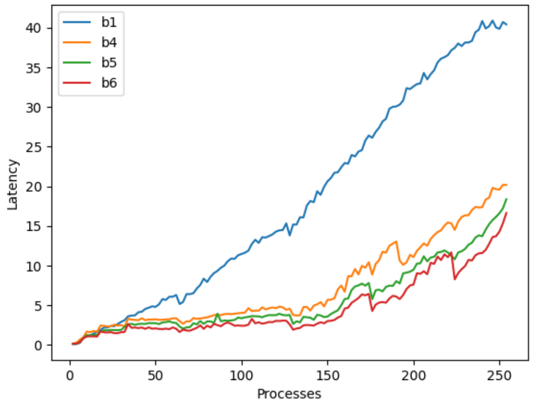


Figure 9: Comparison between the algorithms

We can compare all the previous broadcast algorithms in figure 9. It has been reported the latency of the four algorithms for a message size of 2 byte.
In table 10 it has been reported the difference in latency between the maximum and the minimum size of the non-fixed parameter for the different algorithms, in order to see which one scales better for the extreme cases of min/max size and number of processes.

|  | fixed size |  | fixed processes |  |
|---|---|---|---|---|
|  | 2byte | 1MB | 2 procs | 256 procs |
| b1 | 38.09 | 11204.30 | **78.49** | 11244.70 |
| b4 | 6.96 | **10046.22** | 90.88 | **10130.14** |
| b5 | 5.35 | 11400.35 | 88.58 | 11483.58 |
| b6 | **4.42** | 14300.55 | 86.76 | 14382.89 |

Figure 10: Comparison of the data

From table 10 it can be seen that the latency of the linear broadcast grows a lot, with respect to the other ones, for small sized messages when the number of processes increases. Same happens for binomial tree with big sized messages. Regarding the fixed processes data there is less variance in the results, but still can be observed some differences.

## 4 Scatter

Scatter algorithms aims to send data to all processes in a communicator starting by a designated root. It is similar to broadcast, but broadcast sends the same piece of data to all processes, while scatter sends chunks of the data to different processes.

### 4.1 Ignore (s0)

For this value of the *coll_tuned_scatter_algorithm* variable the user does not specify a particular scatter algorithm. The default algorithm is chosen based on the size of the message, the number of processes, and other internal factors of Open MPI.

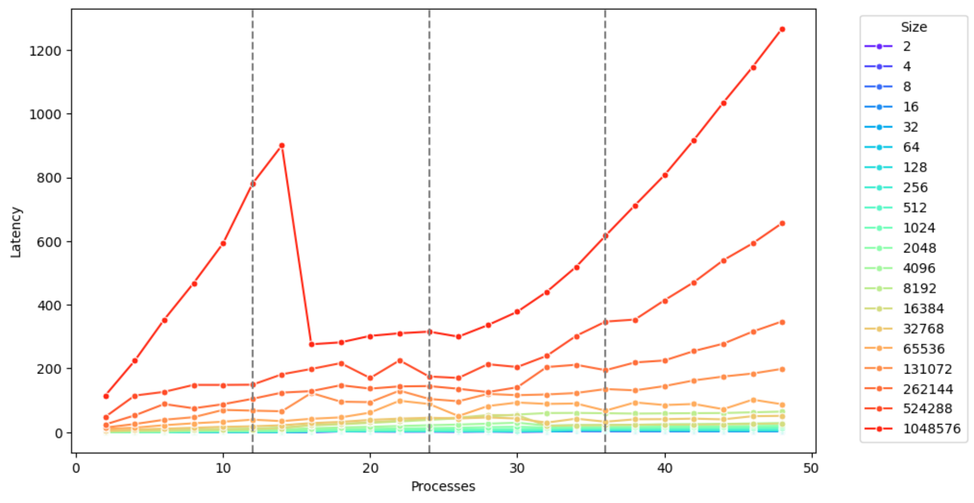


Figure 11: Scatter 0

Analyzing the data we can observe that the trend of latency for this algorithm is characterized by a slower rate of increase compared to other algorithms previously discussed in this report, this is due to the fact that a different algorithm is chosen at each step to better perform according to the given size and number of processes. This choice has a substantial impact on the algorithm's performance, indeed it results to have the lower latency between the analyzed scatter algorithms, except for a few cases in which the number of processes is small and the size of the message is big (but I suspect that this may be due to some noise in the collected data).

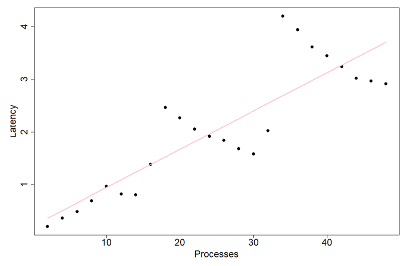


Figure 12: Ignore algoritm (s0)

It has been fitted a model that depends on the number of processes. The resulting formula is:

$$latency = 2.0338 + 4.9285 processes$$

the model has an R squared adjusted of 0.7379 .

## 4.2 Basic linear (s1)

In a basic linear scatter, each process receives a contiguous block of data from the root process. The root process sends data to each process in sequence.

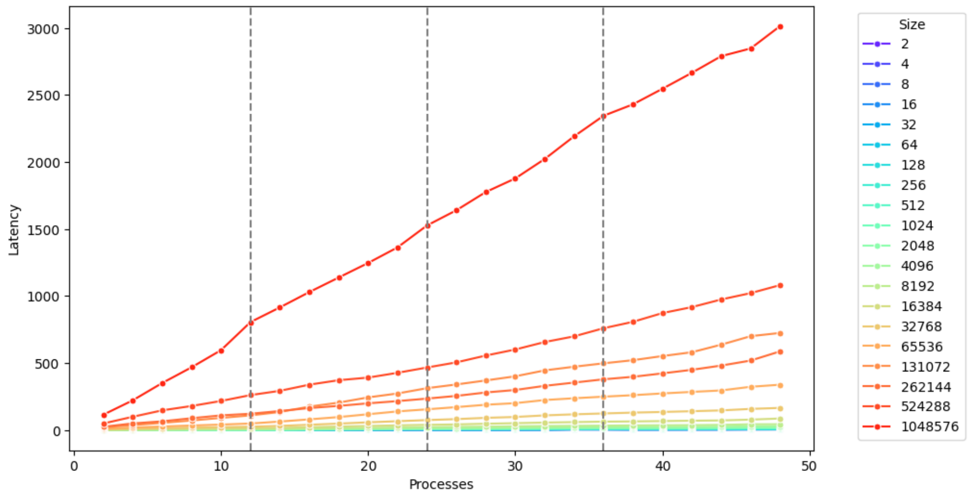


Figure 13: Scatter 1

The data shows that as the number of processes increases, the latency increases: this is consistent with the idea that while parallelization can speed up the computation, it also introduces communication overhead. The latency also increases as the data size increases, which is expected as more data requires more time to process.

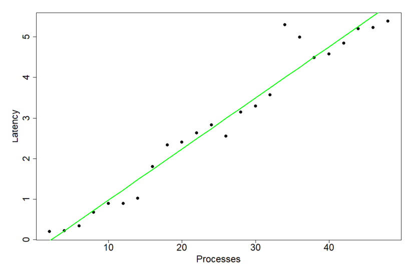


Figure 14: Basic linear scatter (s1)

It has been fitted a model that depends on the cube of the number of processes. The resulting formula is:

$$latency = 1.93 + 6.875e^{-7}processes$$

the model has an R squared adjusted of 0.9377 .

## 4.3 Binomial (s2)

The binomial scatter algorithm uses a binary tree structure to distribute data. Each process receives data from a process that is logarithmically distant from the root process. From figure 15

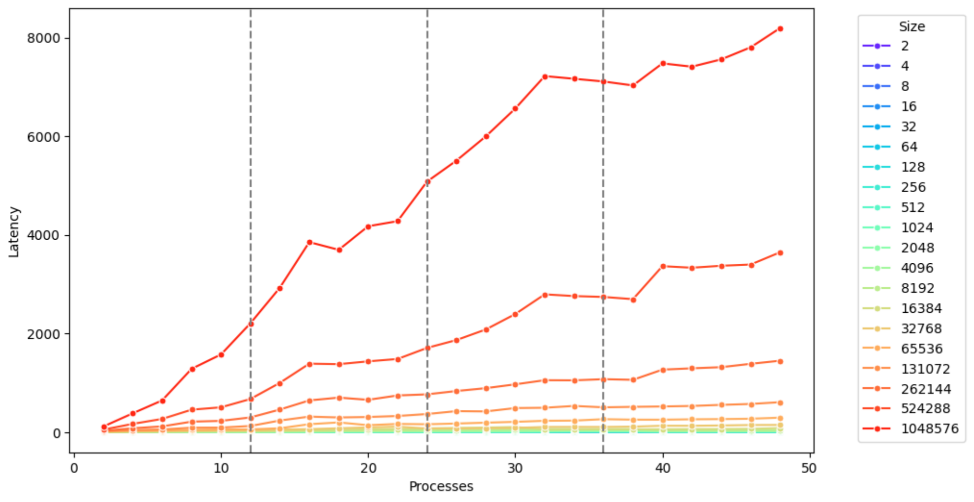
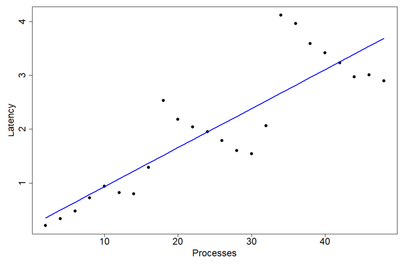


Figure 15: Scatter 2

it can be observed that the latencies reached by the binomial algorithm are higher than the one of the default implementation, as it may be expected. The main worsening is reached for high number of processes and big sized message, while for the other combinations the two algorithms perform quite similar.



I has been fitted a model that depends on the number of processes. The resulting formula is:

$$latency = 0.21 + 0.072processes$$

the model has an R squared adjusted of 0.7376 .

Figure 16: Binomial tree scatter (s2)

## 4.4 Linear_nb (s3)

In a non-blocking scatter, the scatter operations are performed asynchronously. This means that the MPI implementation does not wait for each scatter operation to complete before starting the next one. This can lead to performance improvements by allowing the system to overlap communication and computation, which is particularly convenient in systems with high network latency.

For the collected data it can be seen that as the data size increases the latency also increases; in particular the rate of growth is higher for big message size and big number of processes. The latencies of this algorithm are really close to the default ones, this may suggest that the non-blocking linear algorithm can be valuable.

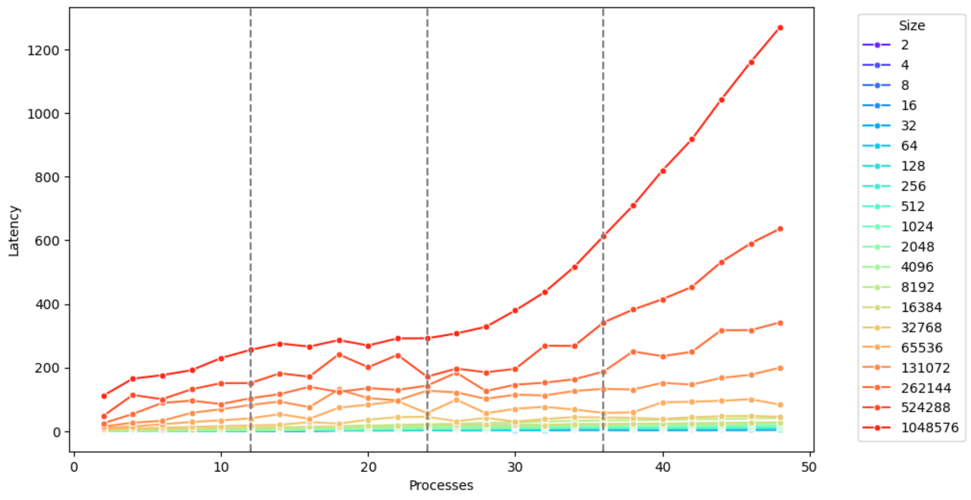


Figure 17: Scatter 3

With respect to the linear scatter algorithm it can be seen that the main improvement of using a non-blocking approach is reached for small sized messages and big numbers of processes.

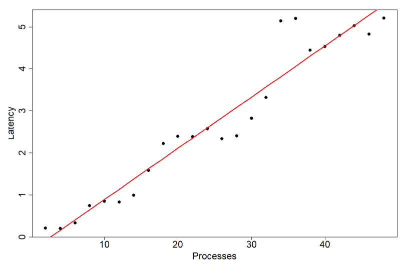


Figure 18: Non blocking scatter (s3)

I has been fitted a linear model model that depends on processes. The resulting formula is:

$$latency = -0.323080 + 0.121807 processes$$

the model has an R squared adjusted of 0.9289 .

## 4.5 Conclusions

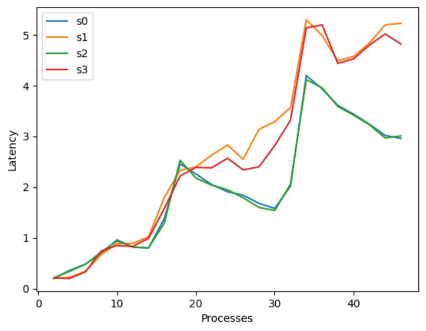


Figure 19: Comparing scatter

In figure 19 it has been compared the latency of the 4 algorithms for a fixed size of 2 bytes. It can be seen that the algorithms perform quite similarly (especially for small number of processes). As for the broadcast algorithms, it can be hypothesized that there is an optimization in transferring data when the size is a power of two, indeed, when the data size corresponds to these powers there is a slight decrease in latency.

In table 20 it has been reported the difference of the latency between the maximum and the minimum size of the non fixed parameter for the different algorithms, in order to see which one scales better for the extreme cases of min/max size and number of processes.

| | fixed size | | fixed processes | |
|---|---|---|---|---|
| | 1byte | 1MB | 2 procs | 256 procs |
| s0 | **2.64** | **1151.81** | 115.11 | **1264.28** |
| s1 | 5.25 | 2899.99 | 114.51 | 3009.25 |
| s2 | 2.57 | 8081.88 | 111.52 | 8190.83 |
| s3 | 4.93 | 1158.85 | **111.32** | 1265.24 |

Figure 20: Comparison of the scatter data

From table 20 it can be seen that the latency of the binomial scatter grows a lot, with respect to the other ones, for big sized messages when the number of processes increases. When the number of processes is fixed to 2 all the algorithms perform quite similarly, as it may be expected since the differences in design are not considerable for such a small number of processes.

## References:

1. Orfeo documentation: https://orfeo-doc.areasciencepark.it/

2. OSU Benchmark documentation: https://mvapich.cse.ohio-state.edu/benchmarks/

3. Emin Nuriyev, Juan-Antonio Rico-Gallego, and Alexey Lastovetsky. ”Model-based selection of optimal MPI broadcast algorithms for multi-core clusters.”. Journal of Parallel and Distributed Computing, vol. 165, 2022, pp. 1-16. ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2022.03.012.