Comparison of openMPI Algorithms for Collective Operations

Gaia Alessio

17 May 2024

1 Introduction

In this report, I present the results of an investigation into the performance of different openMPI algorithms for collective operations, focusing on broadcast and scatter, in relation to message size and the number of processes involved. Optimizing collective operations is crucial because it impacts the overall efficiency of parallel applications.

Experiments were conducted using the OSU MPI benchmark on two EPYC nodes of the ORFEO cluster.

2 Broadcast

Broadcast algorithms enable the distribution of a message from a source process to all other processes within the communicator. These algorithms vary in communication topology and data transmission approach. Four distinct algorithms were chosen: basic linear, chain, binary tree, and knomial tree.

Data for the broadcast algorithm were collected using:

```
mpirun --map-by core -np $processes --mca coll_tuned_use_dynamic_rules true --mca
coll_tuned_bcast_algorithm 1 osu_bcast -m $size -x $repetitions -i $repetitions
```

2.1 Basic Linear

The Basic Linear algorithm takes a direct approach, in which the source process distributes the message to all other processes involved in the communication. This algorithm serves as a basic benchmark for comparing the latency of alternative approaches. It is characterized by simplicity and low implementation complexity. However, it may have scalability limitations when applied to large-scale implementations. Analysis suggests that latency tends to increase with both the number of processors and message size. An increase in the number of processors may lead to greater competition for network resources, while a larger message size may impose a heavier workload on each processor, factors that contribute to the increase in overall latency.

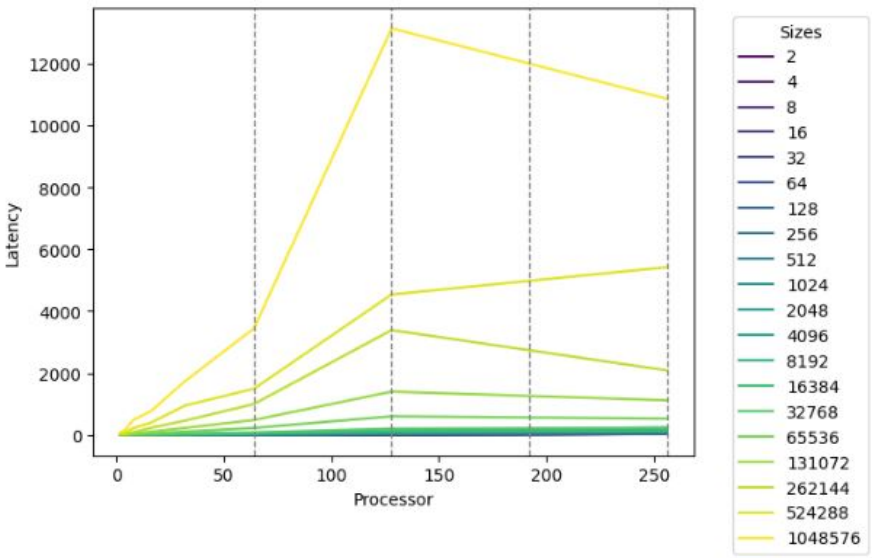

Figure 1: Latency vs. Processors for Different Sizes

Observations indicate that latency increases with the number of processors until it reaches 128 processors, corresponding to an entire EPYC node, after which the growth rate decreases or stabilizes.

Examining the data collected with a fixed message size of 4 bytes, a variation in latency is observed for 64, 128, 192, and 256 processors. This phenomenon may be attributed to the architecture on which the algorithms are tested; thus, these numbers correspond to saturation of one of the sockets within the two nodes, or it may result from algorithmic optimizations tailored to these specific values.

2.1.1 Model

To study the behavior of the data, we attempt to estimate a model that approximates them.

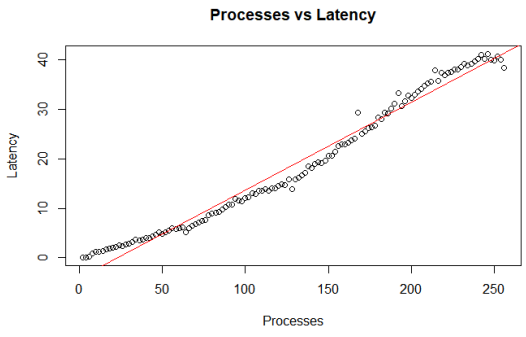

Figure 2: Linear model

The linear model can be expressed as $y = -4.055 + 0.177 \times \text{processes}$. The model has an R-squared value of about 0.9778, indicating that approximately 97.78% of the variation in latency can be explained by the number of processors in the model. This suggests that the model fits the data very well and that the number of processors is a strong predictor of latency.

2.2 Chain

The Chain algorithm utilizes a linear topology for message transmission, where each process forwards the message to the next process in the communicator. It offers improved scalability by distributing the workload evenly among processes. This is achieved by reducing the load on the source process, which only needs to transmit the message to its immediate neighbor, while subsequent processes forward the message along the chain. As a result, Chain is well-suited for medium to large-scale deployments where scalability is a concern.

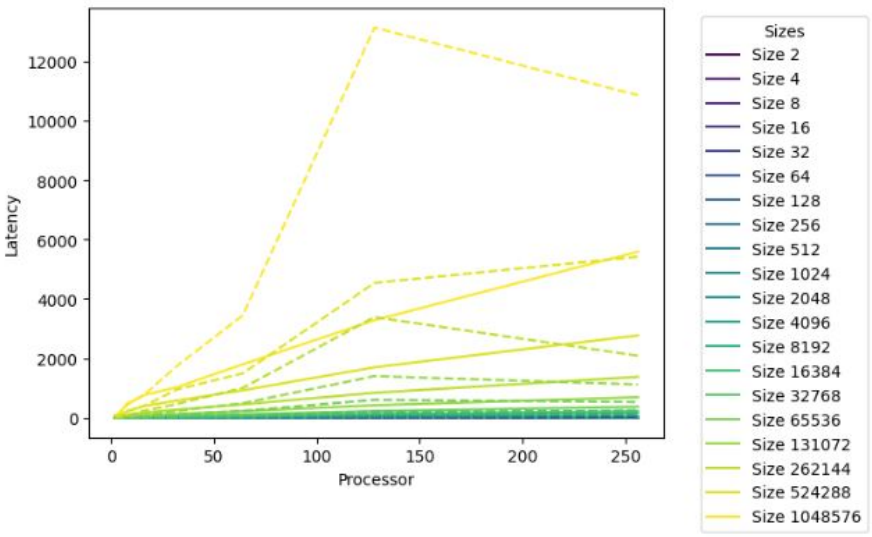

Figure 3: Comparison of Chain and Linear Algorithm for Each Size

Figure 3 shows the latency of the Chain algorithm (solid line) compared with the Linear algorithm (dashed line) to make the comparison.

It is evident from the graph that, in general, the Chain algorithm generally has lower latency than the Linear algorithm, with messages of various sizes and numbers of processes.

For small sizes (e.g., 2, 4, 8, 16), both show relatively low latencies. For larger sizes, the Chain algorithm shows a more consistent latency reduction trend as the number of processes and message sizes increase. In contrast, the Linear algorithm shows higher latency, especially when the workload intensifies.

2.2.1 Model

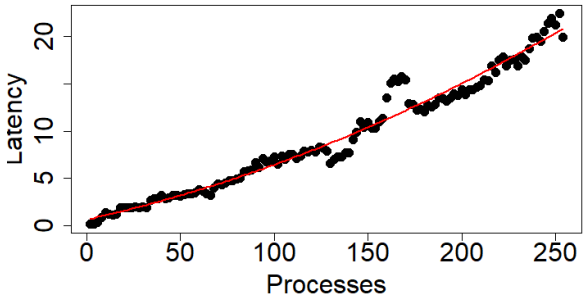

Figure 4: Second-degree polynomial model

In this case, the best-fitting model turns out to be $y = 6.443e^{-1} + 4.416e^{-2}\text{processes} + 1.392e^{-4}\text{processes}^2$. The model has an R-squared value of about 0.9694 so it explains about 96.94% of the variation in latency.

2.3 Binary Tree

The Binary Tree algorithm organizes processes in a hierarchical tree structure, with the source process at the root and leaf processes as the receivers of the broadcasted message. The message is transmitted in a binary way, with each internal node forwarding the message to its two child nodes. This approach enables efficient utilization of network bandwidth and improved scalability compared to linear topologies. Binary Tree is particularly advantageous for large-scale deployments as it ensures balanced workload distribution among processes.

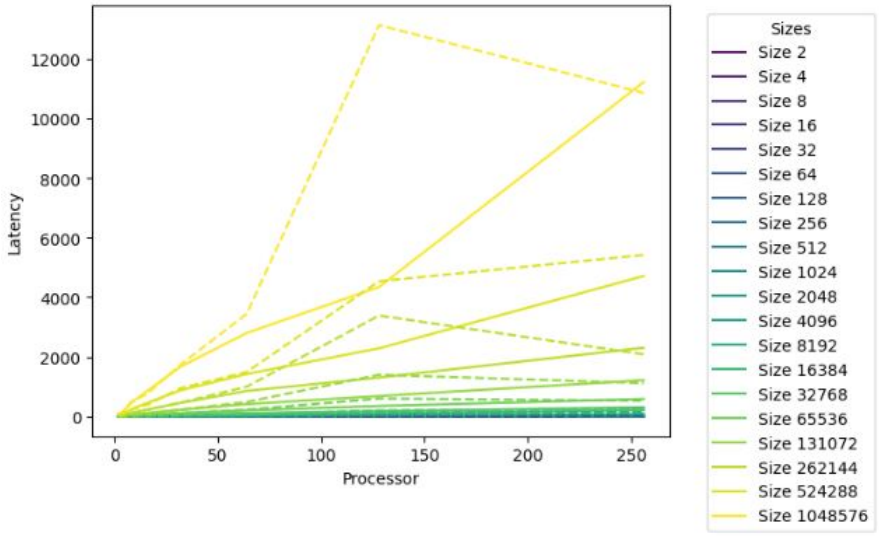

Figure 5: Comparison of Binary Tree and Linear Algorithm for Each Size

Both algorithms show relatively low latency for small message sizes, with small variations between different numbers of processes. But the Binary Tree algorithm consistently shows slightly lower latencies than the Linear algorithm for small message sizes. This suggests that

the hierarchical structure of the Binary Tree promotes more efficient message dissemination. With larger message sizes, latency increases significantly for both algorithms; however, the Binary Tree algorithm retains its advantage over the Linear algorithm, indicating its scalability and efficiency even with a larger number of processes.

Thus, while both algorithms have similar results for small message sizes, the Binary Tree algorithm demonstrates superior performance for larger message sizes. The hierarchical structure of the Binary Tree facilitates efficient message routing and dissemination, resulting in lower latencies compared to the Linear algorithm on different message sizes and numbers of processes.

2.3.1 Model

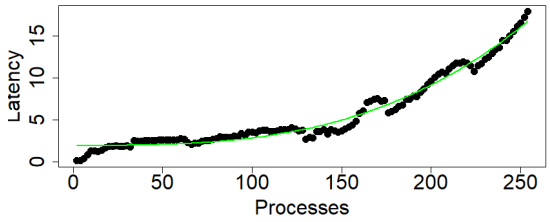

Figure 6: Third-degree polynomial model

In this case, the best-fitting model turns out to be $y = 1,957 + 8,99e^{-2}\text{processes}^3$. The model has an R-squared of about 0.9724, indicating that about 97.24% of the variation in latency is explained by the predictors in the model.

2.4 Binomial Tree

With a binomial tree structure, the message is further forwarded by each node after receiving it, leading to exponential growth of the tree structure resulting from the doubling of participating nodes at each propagation step.

The latency of the binomial tree algorithm tends to increase with data size as more communication steps are required, which contributes to the latency.

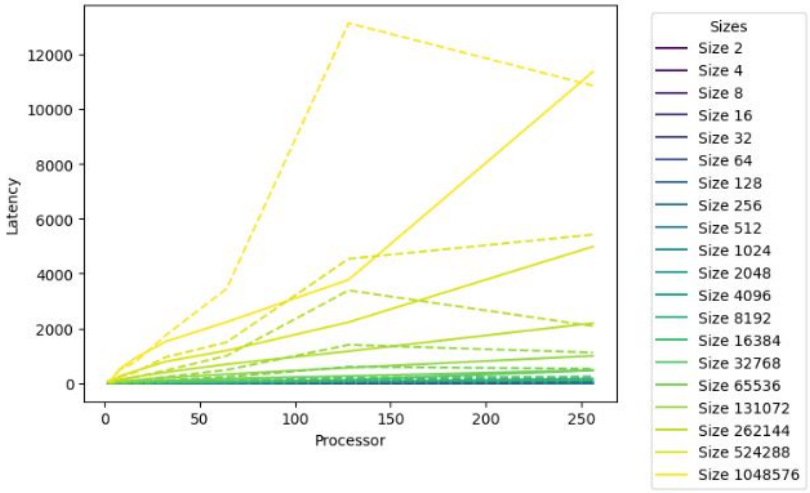

Figure 7: Comparison of Binomial Tree and Linear Algorithm for Each Size

It can be seen from Figure 7 that the Linear algorithm seems to show lower latency for smaller sizes (up to 2024), while the Binomial Tree algorithm tends to have lower latency for larger sizes.

However, as the data size and number of processes increase, the latency is comparable between the two algorithms although the Binomial Tree algorithm shows better scalability with increasing process size, as indicated by the less significant increase in latency with increasing process size compared to the Linear algorithm.

2.4.1 Model

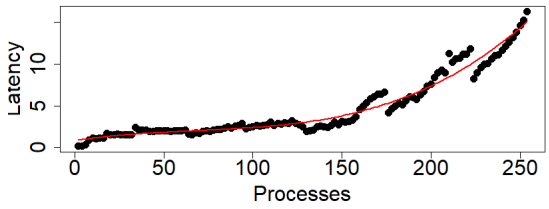

Figure 8: Third-degree polynomial model

In this case, the best-fitting model turns out to be $y = 8.297e^{-1} + 3.261e^{-2}\text{processes} - 3.403e^{-4}\text{processes}^2 + 1.704e^{-6}\text{processes}^3$. The model has an R-squared of about 0.9568, indicating that about 95.68% of the variation in latency is explained by the predictors in the model.

2.5 Comparison

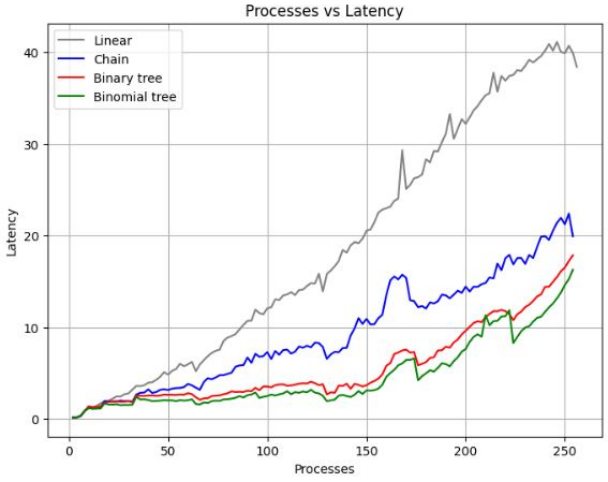

Figure 9: Comparison of Different Algorithms at Fixed Size 4

In Figure 9, we can compare all the previously mentioned broadcast algorithms where the latency of the four algorithms for a message size of 4 bytes has been reported.

3 Scatter

Scatter is an operation involving the distribution of data from a source process to all other participating processes. This process divides a message into parts and sends it to different processes, each of which receives a specific portion of the message, facilitating the parallelization of complex tasks and improving overall performance. Four different algorithms were used: ignore, basic linear, binomial, and non-blocking linear.

Data for the scatter algorithm were collected using:

```
mpirun --map-by core -np $processes --mca coll_tuned_use_dynamic_rules true --mca
coll_tuned_scatter_algorithm 0 osu_scatter -m $size -x $repetitions -i $repetitions
```

3.1 Ignore

The Ignore algorithm implements an approach in which the algorithm used is not specified by the user but is chosen each time based on message size and the number of processes.

This algorithm turns out to be slower than the others because a different algorithm is chosen each time to achieve better performance based on the different size and number of processes.

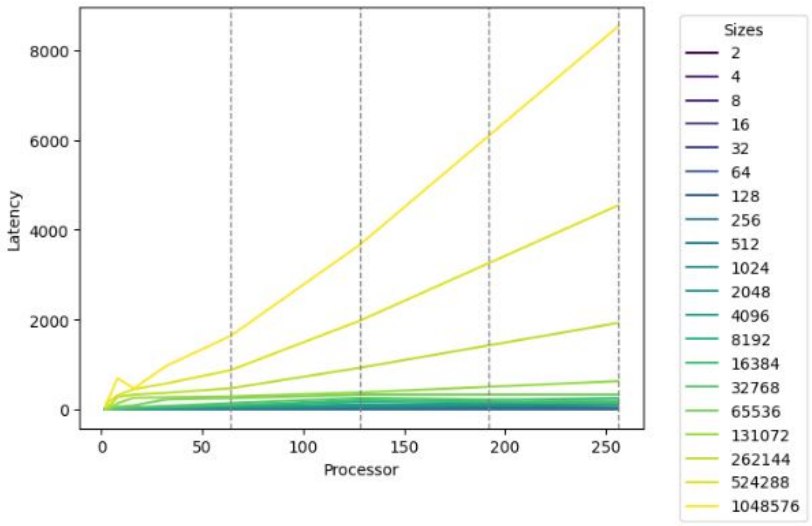

Figure 10: Latency vs. Processors for Different Sizes

3.1.1 Model

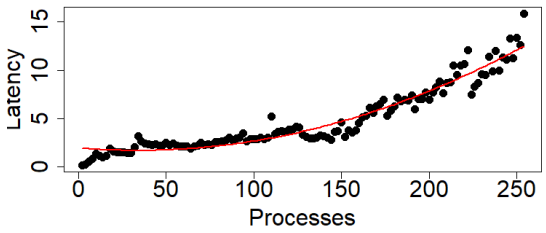

Figure 11: Third-degree polynomial model

In this case, the best-fitting model turns out to be $y = 1.937 - 1.493e^{-2}\text{processes} + 2.223\text{processes}^2 + 4.640 \times \text{processes}^3$. The model has an R-squared of about 0.9312, indicating that about 93.12% of the variation in latency is explained by the predictors in the model.

3.2 Basic Linear

In the Basic Linear algorithm, the source process sends the message to all other processes in sequence, one after the other.

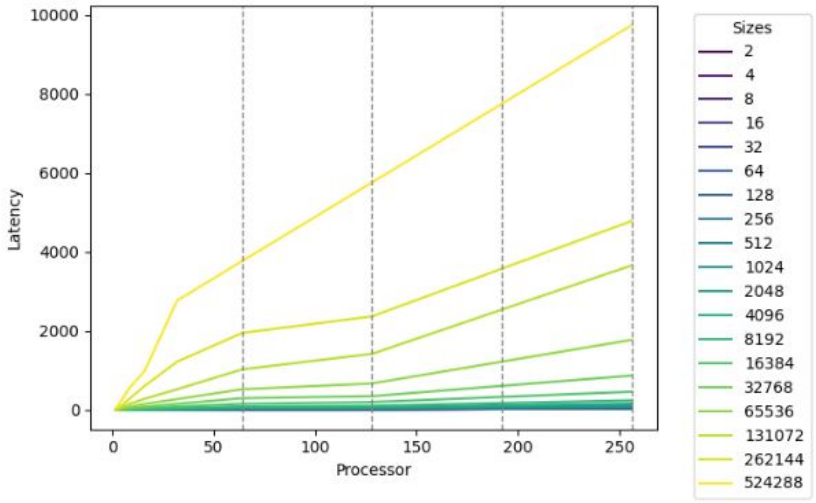

Figure 12: Latency vs. Processors for Different Sizes

The data show that, as expected, as the number of processes increases, latency increases. This is because parallelization, although it speeds up work, increases communication overhead. In addition, increasing data size increases data processing time and thus latency.

3.2.1 Model

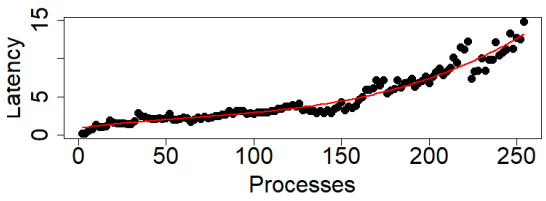

Figure 13: Fourth-degree polynomial model

In this case, the best-fitting model turns out to be $y = 8.708e^{-1} + 3.008e^{-2}\text{processes} + 1.009e^{-6}\text{processes}^3$. The model has an R-squared of about 0.9441, indicating that about 94.41% of the variation in latency is explained by the predictors in the model.

3.3 Binomial

The Binomial algorithm uses a binary tree topology in which the message is sent along the shortest possible path.

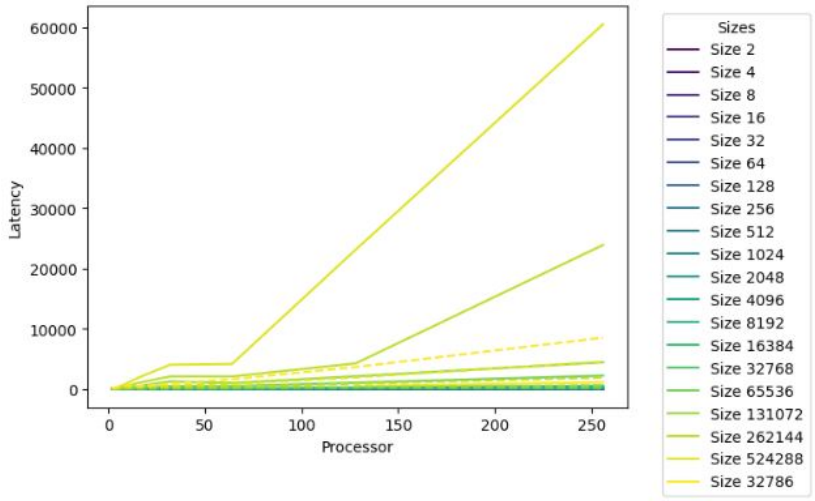

Figure 14: Comparison of Binomial Tree and Linear Algorithm for Each Size

From Figure 14, it can be noted that for the small dimensions, the Linear algorithm shows a latency that grows gradually as the number of processes increases. The Binomial algorithm shows similar behavior with slightly better performance. However, the Linear algorithm works well for relatively small data and the additional parallelism does not offer significant advantages.

For a large number of large processes and messages, it is noted that the latencies reached by the Binomial algorithm are higher than that of the Linear implementation, as can be expected.

3.3.1 Model

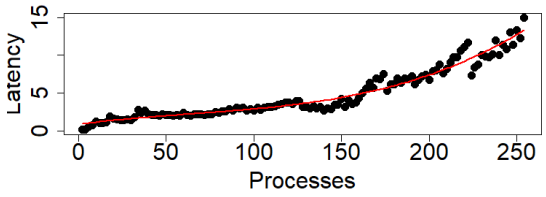

Figure 15: Fourth-degree polynomial model

In this case, the best-fitting model turns out to be $y = 8.705e^{-1} + 3.060e^{-2}\text{processes} + 1.135e^{-6}\text{processes}^3$. The model has an R-squared of about 0.9514, indicating that about 95.14% of the variation in latency is explained by the predictors in the model.

3.4 Non-blocking Linear

The Non-blocking Linear algorithm uses a non-blocking approach that allows processes to continue to function during message transmission. This implementation therefore does not wait for each scatter operation to be completed before starting the next. This method may require more implementation complexity, but offers greater scalability and better performance.

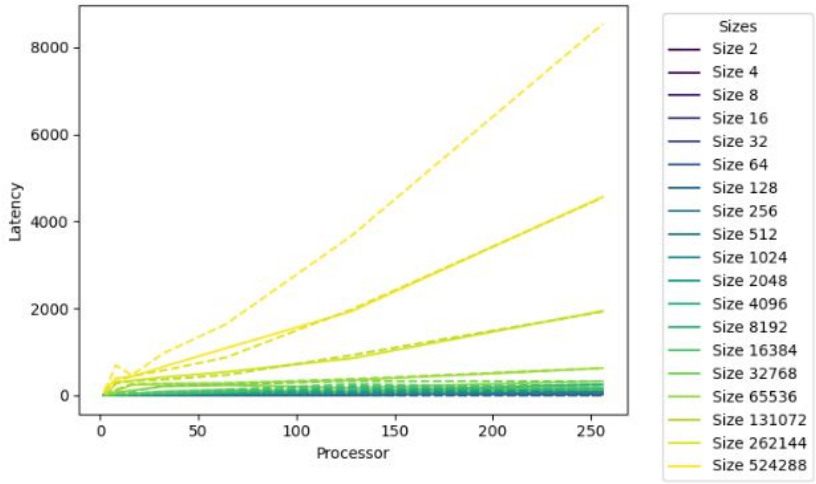

Figure 16: Comparison of Non-blocking Linear and Linear Algorithm for Each Size

For small data, the Linear algorithm can generally have a lower latency than the Non-blocking Linear algorithm. This is because the Linear algorithm is simpler and requires less synchronization between processes, thus reducing overall overload and, consequently, latency.

With larger data sizes, it becomes critical to optimize system resource utilization and reduce overall execution time, and the Non-blocking Linear algorithm becomes preferable in this case because it allows for greater overlap between communication and computation, allowing processes to continue with computation as data is transferred. This can result in a reduction in the overall time needed to complete the task.

3.4.1 Model

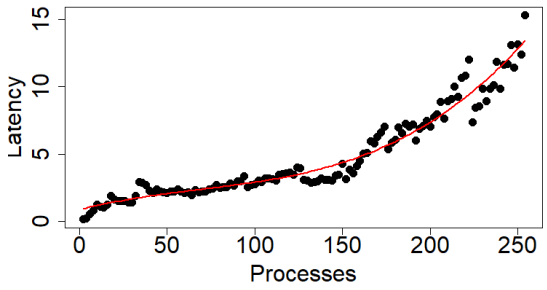

Figure 17: Fourth-degree polynomial model

In this case, the best-fitting model turns out to be $y = 8.753e^{-1} + 3.324e^{-2}\text{processes} + 1.247e^{-6}\text{processes}^3$ The model has an R-squared of about 0.9517, indicating that about 95.17% of the variation in latency is explained by the predictors in the model.

3.5 Comparison

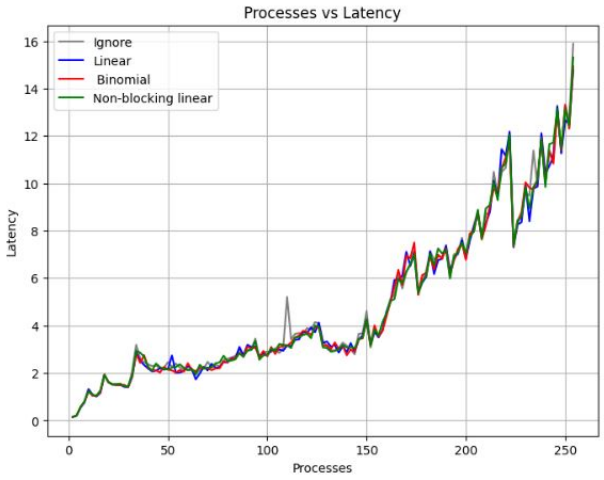

Figure 18: Comparison of Different Algorithms at Fixed Size 4

In Figure 18, we can compare all the previously mentioned scatter algorithms where the latency of the four algorithms for a message size of 4 bytes has been reported.

It can be seen that algorithms behave very similarly.