High Performance Computing
Modeling the Latency of MPI Collective Communication Algorithms

Edoardo Insaghi

1 Introduction

The openMPI library is a widely used implementation of the Message Passing Interface, offering a number of features for parallel programming. Among its possibilities, openMPI provides multiple algorithms to perform collective operations, which play a fundamental role in parallel computing applications. Collective operations involve communication and coordination between processes, facilitating data exchange and synchronization through algorithms that rely upon point to point communications.

This report presents an evaluation of the performance of two selected openMPI collective algorithms for two fundamental collective operations: the broadcast operation, which is designed to distribute a piece of data from one process, often referred to as the root, to all other processes in a communicator; and the reduce operation, which is used to aggregate values from all processes in a communicator, often performing operations like summation, product, maximum, or minimum on the data held by each process.

The evaluation focuses on estimating the latency of three implementations of said operations: the linear algorithm, the pipeline algorithm, and the binary tree algorithm, considering variations in the number of processes, the size of the messages exchanged, and the allocation of the computational resources across the nodes. To conduct this evaluation, the OSU MPI benchmark has been employed, which is a well-established tool for assessing the performance of MPI implementations. The benchmark is executed on two whole EPYC nodes of the ORFEO cluster. Each of the two nodes is equipped with two AMD EPYC 7H12 CPUs, each possessing 64 CPU cores, for a total of 256 cores across the two nodes.

The goal is to compare the latency values obtained from the different openMPI implementations and to build a prediction model for the latency, taking into account the different variables at play mentioned earlier. In the following paragraphs the results of the analysis are presented. The first one focuses on how the latency depends on the number of MPI processes present in the communicator, while the second also includes the effect of the size of the data on the total total time needed to conclude the communication.

2 Latency and Number of Processes

This section focuses on assessing the impact of the number of MPI processes on communication latency in the two chosen operations, broadcast and reduce, through the use of a simple linear model in which the number of processes represent the covariate while the latency plays the role of the response variable.

As mentioned in the introduction, the data has been collected on two whole EPYC nodes. Both the number of warmup iterations and the total number of iterations have been increased drastically because default values showed somewhat inconsistent and noisy results during test runs. The size of the message has been set to one byte, in this way the total time needed to conclude the communication is given almost entirely by the latency, as the time needed for sending one byte is negligible. The following plot shows the results obtained from the benchmark of the broadcast collective communication.

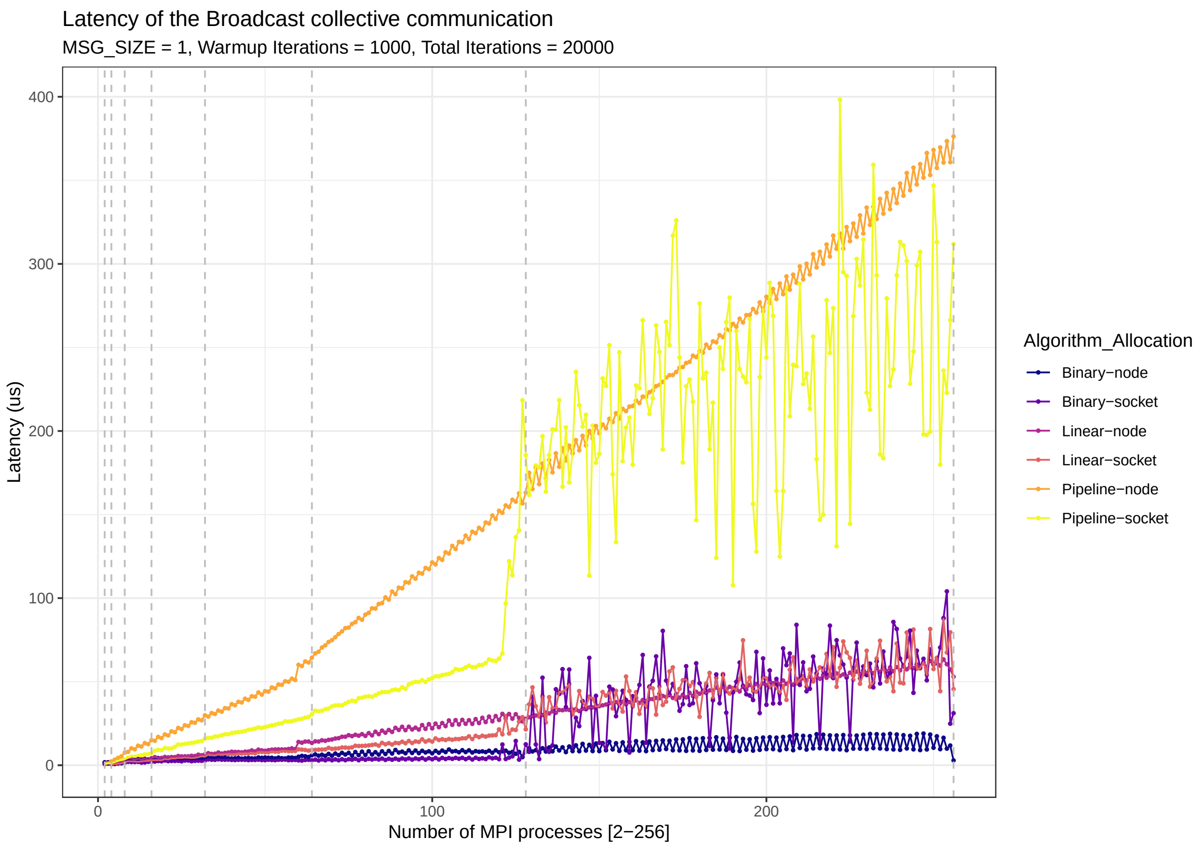


Figure 1: Comparison of different algorithms for the broadcast communication

The data shows that, as expected, the latency increases with the number of processes in the communicator, but huge differences emerge between the different algorithms and resource allocations. The pipeline algorithm seems to be performing the worst among the three, while the binary tree algorithm is the most robust with respect to the number of processes, which is to be expected since it is the only one which theoretically scales with the logarithm of computational units. The tests performed with allocation by socket also seem to show a recurring pattern, the latency has an evident discontinuity at around 128 cores, which is the size of one EPYC node. This phenomenon finds its explanation in the fact that when the computational units are allocated by socket, the two nodes start communicating only after one has exhausted all of its resources, and communication between nodes takes longer than communication within different processes inside the same node. This information is crucial and must be included in the analysis in order to obtain a better model.

The plot is quite messy, so it is worth zooming in on every algorithm to analyse each more closely and get better understand of the linear fit. The R^2 is not included in the report as it would be redundant since it is well beyond 0.9 for each model. The only consideration in this regard is that the R^2 tends to be lower for the mapping by socket because the variability of the data increases noticeably after 128 processes.

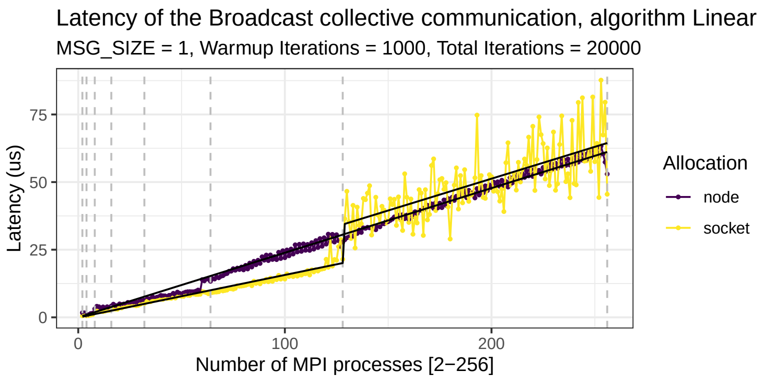


Figure 2: Broadcast, Linear Algorithm

(a) Allocation by Socket

	Estimate	Std. Error	t value	Pr(>|t|)
MPI_Processes	0.1572	0.0151	10.44	0.0000
MPI_Processes > 128	4.1839	2.9183	1.43	0.1529
MPI_Processes : MPI_Processes > 128	0.0781	0.0212	3.69	0.0003

(b) Allocation by Node

	Estimate	Std. Error	t value	Pr(>|t|)
MPI_Processes	0.2388	0.0007	360.94	0.0000

The results of the linear model tell us that the latency increases on average by 0.24 microseconds for every new processor in the communicator with the allocation by node, while it increases only by 0.16 microseconds until the 128 cores mark, after which a jump takes place and the regression lines have the same slope. The same analysis can be repeated for the other two algorithms, with somewhat similar results.

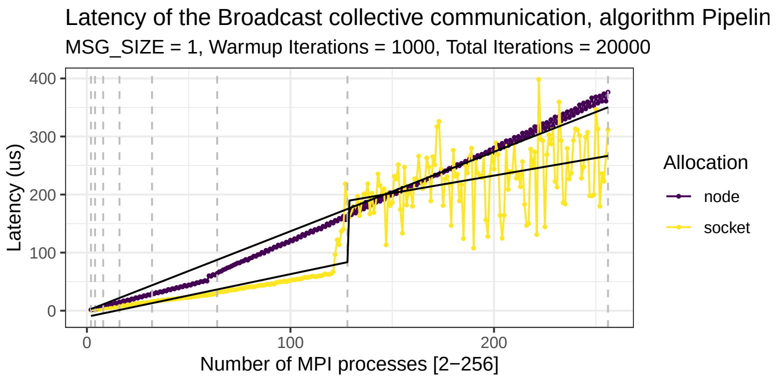


Figure 3: Broadcast, Pipeline Algorithm

(a) Allocation by Socket

	Estimate	Std. Error	t value	Pr(>|t|)
MPI_Processes	0.7352	0.0918	8.01	0.0000
MPI_Processes > 128	111.6247	17.7808	6.28	0.0000
MPI_Processes : MPI_Processes > 128	-0.1298	0.1290	-1.01	0.3153

(b) Allocation by Node

	Estimate	Std. Error	t value	Pr(>|t|)
MPI_Processes	1.3682	0.0060	229.15	0.0000

From the summary of the linear model we can see that the consideration on the results for the linear algorithm can be extended to the pipeline algorithm, with the only exception that the least squares line for the socket allocation maintains a smaller slope with respect to the allocation by node even after the communication between cores on different nodes takes place. Also, the number of cores has a much bigger influence on the latency, as the first plot showed.

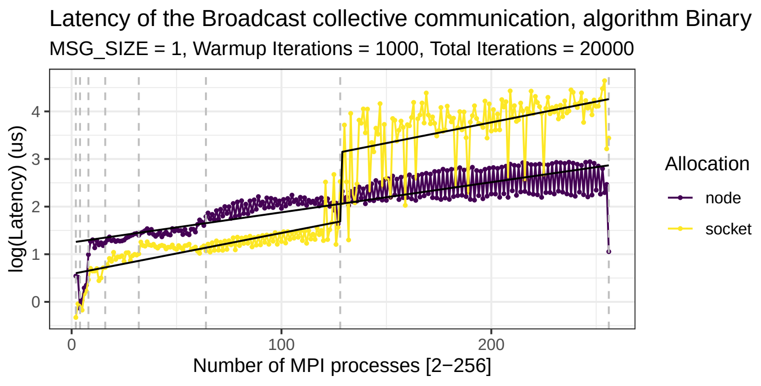


Figure 4: Broadcast, Binary Algorithm

(a) Allocation by Socket

	Estimate	Std. Error	t value	Pr(>|t|)
MPI_Processes	0.0086	0.0010	8.39	0.0000
MPI_Processes > 128	2.0269	0.1989	10.19	0.0000
MPI_Processes : MPI_Processes >	0.0001	0.0014	0.07	0.9459

(b) Allocation by Node

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	1.2472	0.0396	31.50	0.0000
MPI_Processes	0.0063	0.0003	23.74	0.0000

The results for the binary tree algorithm are absolutely comparable with the ones from the other two implementations. It is known from theory that with this algorithm the latency should scale logarithmically with the number of processes in the communicator, and therefore we expect the logarithm of the latency to be linear with respect to the number of cores. For this reason the logarithm of the latency has been taken, and the results of the fit are satisfactory after the transformation. In this last model an intercept has been added, this is necessary due to the logarithmic scaling, and adds noticeable explanation power at the cost of some interpretability.

The same approach can be repeated for the reduce collective communication, for which it is also possible to build a latency model using linear regression. This is the data collected on the two EPYC nodes, likewise with the broadcast communication.

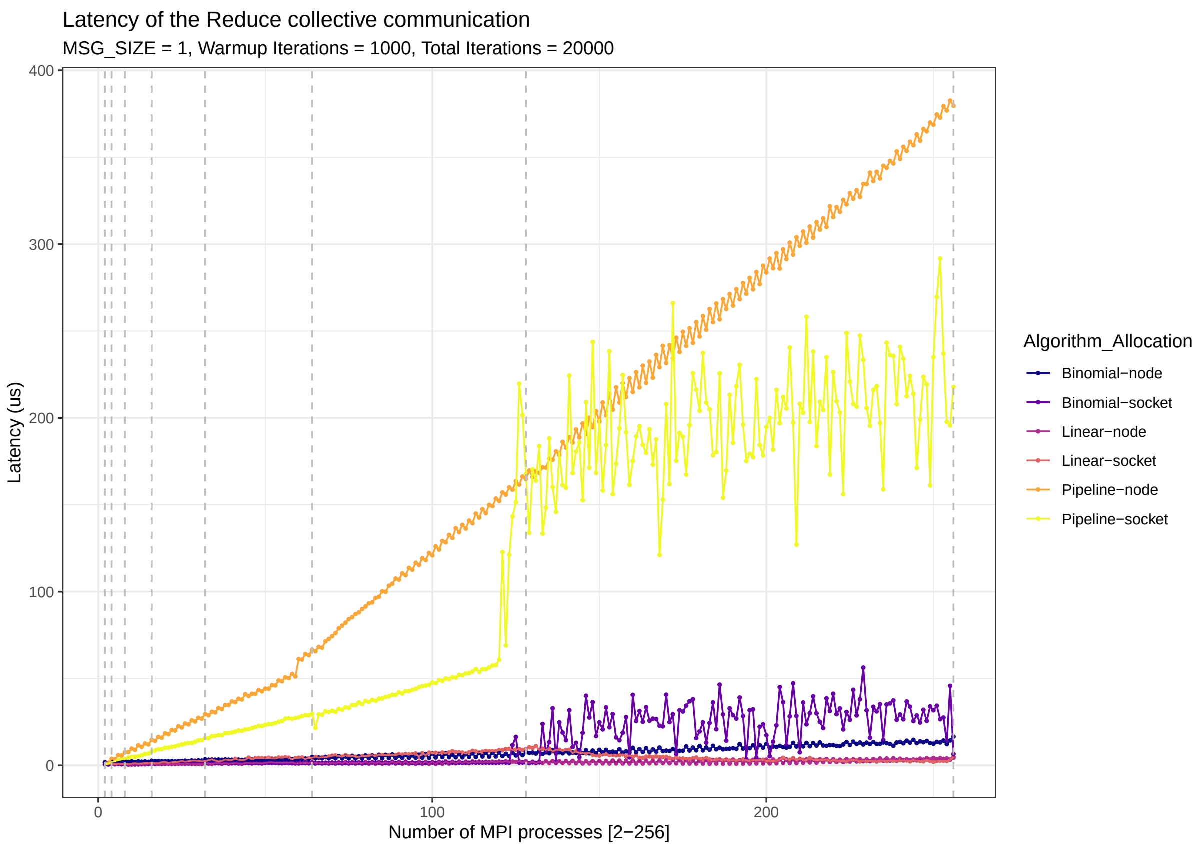


Figure 5: Comparison of different algorithms for the reduce communication

This plot shows some obvious similarities with previous one, the pipeline algorithm performs the worst amongst the three also in the reduce communication, and the data for the allocation by socket becomes more noisy after 128 cores, even if this effect seems to be less prominent than in the broadcast communication, especially for the linear algorithm. Actually something peculiar happens with the linear algorithm, as the following plot highlights.

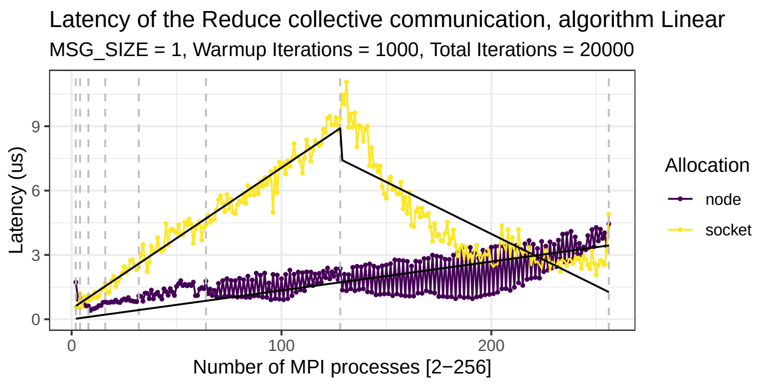


Figure 6: Reduce, Linear Algorithm

(a) Allocation by Socket

	Estimate	Std. Error	t value	Pr(>|t|)
MPI_Processes	0.0658	0.0021	30.66	0.0000
MPI_Processes > 128	13.6566	0.4157	32.85	0.0000
MPI_Processes : MPI_Processes > 128	-0.1142	0.0030	-37.85	0.0000

(b) Allocation by Node

	Estimate	Std. Error	t value	Pr(>|t|)
MPI_Processes	0.0134	0.0003	43.48	0.0000

When computational resources are allocated by socket, an unexpected trend emerges, the latency decreases with the number of processors after the 128 mark. This phenomenon defies every intuition about the topic, and while to this day I do not have a clear explanation for it, the reason is most likely to be looked for in the architecture of the nodes rather than the details of the algorithm. This results is also probably not due to an unfortunate run, because the same benchmark has be re run different times with unfortunately similar results.

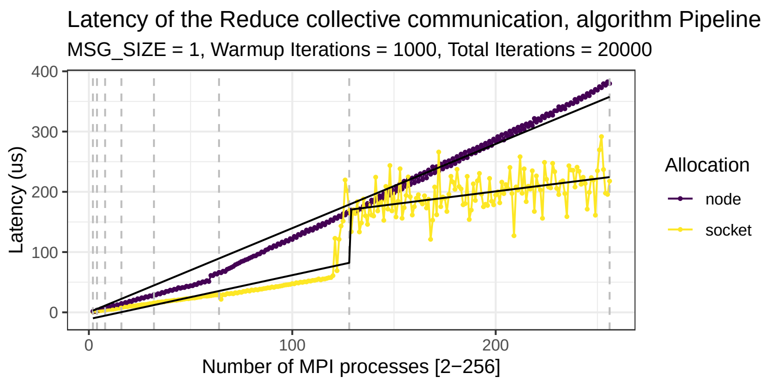


Figure 7: Reduce, Pipeline Algorithm

The results for the pipeline and binary tree algorithms on the contrary are much more easily

(a) Allocation by Socket

	Estimate	Std. Error	t value	Pr(>|t|)
MPI_Processes	0.7301	0.0606	12.05	0.0000
MPI_Processes > 128	116.0453	11.7366	9.89	0.0000
MPI_Processes : MPI_Processes > 128	-0.3068	0.0852	-3.60	0.0004

(b) Allocation by Node

	Estimate	Std. Error	t value	Pr(>|t|)
MPI_Processes	1.3969	0.0064	217.55	0.0000

comparable with their broadcast counterparts, with whom show important similarities. First of all, the pipeline algorithm is more influenced by the number of cores, as its slope is much steeper and the latency becomes more noisy and therefore less predictable after processes in different EPYC nodes start communicating when resources are allocated by socket. In the following plot, the last for this section, the y axis has been scaled with the natural logarithm as in the broadcast case, for essentially the same reasons.

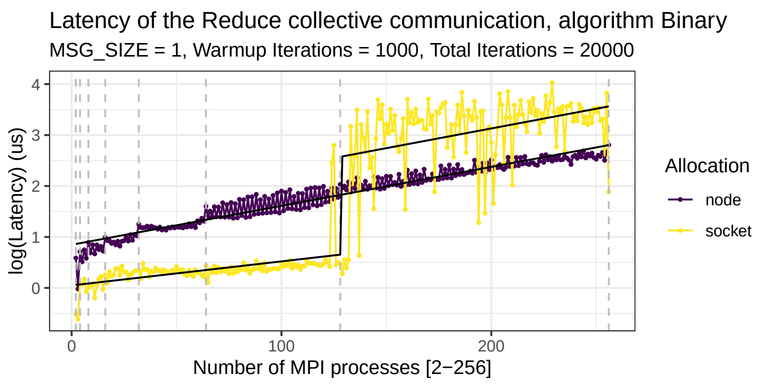


Figure 8: Reduce, Binary Algorithm

(a) Allocation by Socket

	Estimate	Std. Error	t value	Pr(>|t|)
MPI_Processes	0.0047	0.0013	3.70	0.0003
MPI_Processes > 128	1.5831	0.2469	6.41	0.0000
MPI_Processes : MPI_Processes > 128	0.0030	0.0018	1.68	0.0934

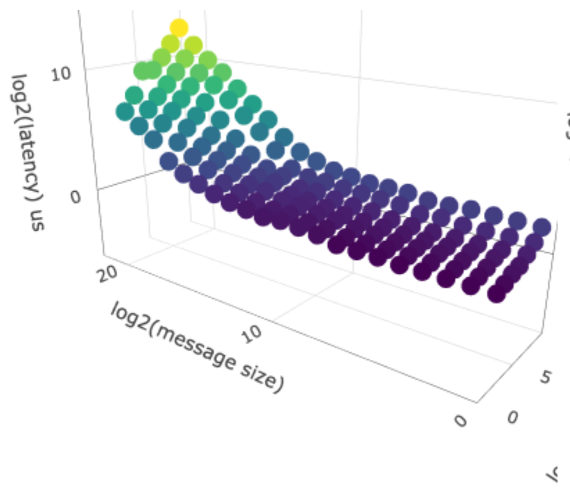
(b) Allocation by Node

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	0.8476	0.0184	46.01	0.0000
MPI_Processes	0.0076	0.0001	61.58	0.0000

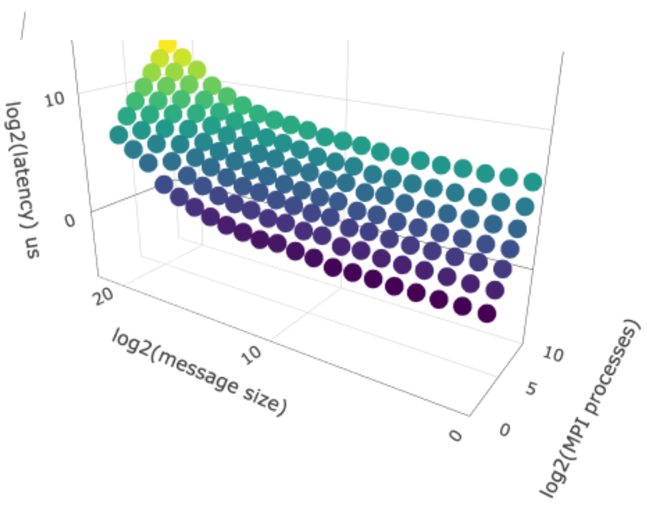
3 Latency and Message Size

In real world applications the size of the message is not just one byte, but it may vary substantially. In order to obtain a better performance model it is possible to include the impact that the size of the message have on the total time needed to conclude a communication. This allows to better understand the how the number of processes and the message size interact, and eventually leads to a more general model.

The data has been collected by varying both the size of the message and the number of cores with the powers of 2 to obtain a grid of values. The result is a surface of points that is somewhat smooth, with few exceptions that thought of as outliers. The following plots show the results for two algorithms used to implement the reduce communication, and serve mostly as an example as they do not add much insight. All three axes scaled logarithmically to make the plots more easily understandable, the ten remaining plots have been omitted from this paper but can be found in the */plots* folder of the Github repository.



(a) Binary Algorithm, allocation by Node



(b) Pipeline Algorithm, allocation by Node

These plots show that, of course, the size of the message plays a fundamental role in determining the total time of the communication. They also seem to exclude major forms of interactions between the number of processes and the size of the data.

Since the size of the message appears to explain the majority of the variance of the data, the linear model can be simplified. Also, given that the relationship between time and message size is faster than linear, in the linear model both the latency and the number of processes has been scaled logarithmically, which greatly increase the predictive power of the model. Hereafter are presented the results for the linear model of the broadcast collective communication.

Table 7: Summary of the Linear Model for the Broadcast Communication

	Estimate	Std. Error	t value	Pr(>|t|)
log2(MPI_Processes) : Binary-node	0.6440	0.0308	20.94	0.0000
log2(MPI_Processes) : Binary-socket	0.5938	0.0308	19.31	0.0000
log2(MPI_Processes) : Linear-node	0.8181	0.0308	26.60	0.0000
log2(MPI_Processes) : Linear-socket	0.8728	0.0308	28.38	0.0000
log2(MPI_Processes) : Pipeline-node	1.1559	0.0308	37.59	0.0000
log2(MPI_Processes) : Pipeline-socket	1.1131	0.0308	36.20	0.0000
Message_Size : Binary-node	8.131e-06	5.205e-07	15.62	0.0000
Message_Size : Binary-socket	9.099e-06	5.205e-07	17.48	0.0000
Message_Size : Linear-node	7.871e-06	5.205e-07	15.12	0.0000
Message_Size : Linear-socket	9.166e-06	5.205e-07	17.61	0.0000
Message_Size : Pipeline-node	5.947e-06	5.205e-07	11.43	0.0000
Message_Size : Pipeline-socket	8.130e-06	5.205e-07	15.62	0.0000

The summary of this last model confirms the results obtained from the data in the last paragraph, in the sense that the binary algorithm performs the best among the three and the pipeline shows the worst results with respect to the number of processes. In addition we obtain the estimates for the impact of the size of the message on the total communication time, which seem to be fairly similar among the three algorithms. the R^2 of the regression line is 0.96, which confirms that the model fits the data well as it is able to explain the majority of the variance of the data.

The same analysis can now be conducted on the reduce collective communication, with the following results.

Table 8: Summary of the Linear Model for the Reduce Communication

	Estimate	Std. Error	t value	Pr(>|t|)
log2(MPI_Processes) : Binary-node	0.6532	0.0342	19.09	0.0000
log2(MPI_Processes) : Binary-socket	0.4583	0.0342	13.39	0.0000
log2(MPI_Processes) : Linear-node	0.3954	0.0342	11.55	0.0000
log2(MPI_Processes) : Linear-socket	0.6488	0.0342	18.96	0.0000
log2(MPI_Processes) : Pipeline-node	1.1630	0.0342	33.98	0.0000
log2(MPI_Processes) : Pipeline-socket	1.1310	0.0342	33.05	0.0000
Message_Size : Binary-node	9.016e-06	5.793e-07	15.56	0.0000
Message_Size : Binary-socket	8.876e-06	5.793e-07	15.32	0.0000
Message_Size : Linear-node	1.077e-05	5.793e-07	18.59	0.0000
Message_Size : Linear-socket	1.001e-05	5.793e-07	17.29	0.0000
Message_Size : Pipeline-node	6.658e-06	5.793e-07	11.49	0.0000
Message_Size : Pipeline-socket	8.280e-06	5.793e-07	14.29	0.0000

The results of the model are consistent with the rest of the analysis, and while the R^2 is slightly lower that the broadcast model, at 0.89, it still represent a good adaptation to the collected data.