

HPCA3

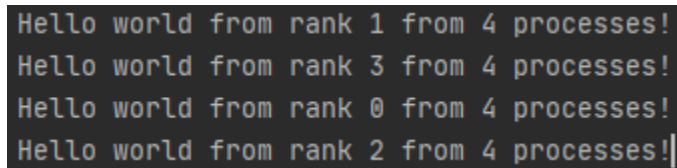
Arthur Picault & Brieuc Horard

10th May 2020

GitHub: <https://github.com/AeroPyk/HPCA3>

1 Exercise 1

1.1 Output



```
Hello world from rank 1 from 4 processes!  
Hello world from rank 3 from 4 processes!  
Hello world from rank 0 from 4 processes!  
Hello world from rank 2 from 4 processes!
```

Figure 1: Hello World!

1.2 Compiling

To compile the program on Beskow, since cray-mpich is loaded by default, we use:

```
cc my_program.c -o my_program
```

On my computer, with Windows OS and CLion IDE I installed MS-MPI and added to the CMake file the lines:

```
find_package(MPI REQUIRED)  
target_link_libraries(Ex1 ${MPI_LIBRARIES})
```

1.3 Running

On beskow, we run it as we used to.

```
salloc --nodes=1 -t 00:05:00 -A edu20.DD2356 -C Haswell  
srun -n 4 ./my_program
```

To change the number of processes we just change the -n option parameter to the number of processes we want.

During the lab, we realized it was really hard to get 128 nodes, and by seeing a comment in the discussion session we realized it was possible to run more than one task on a node. To do so we use the srun command as follow

```
srun --nodes=1 --ntasks-per-node=4 ./my_program
```

This result gives the same result as the previous command line but with only one node. With this we can easily go up to 128 tasks with limited nodes since a node can accept up to 64 tasks.

1.4 About MPI

Rank (rank) and total number of processes (size):

```
MPI_Comm_size(MPI_COMM_WORLD, &size);  
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

MPI is a standard (it's a book) therefore, several implementation exists. Most used MPI implementations are:

- OpenMPI
- MPICH

2 Exercice 2

2.1 MPI Blocking Communication & Linear Reduction Algorithm

2.1.1 Results

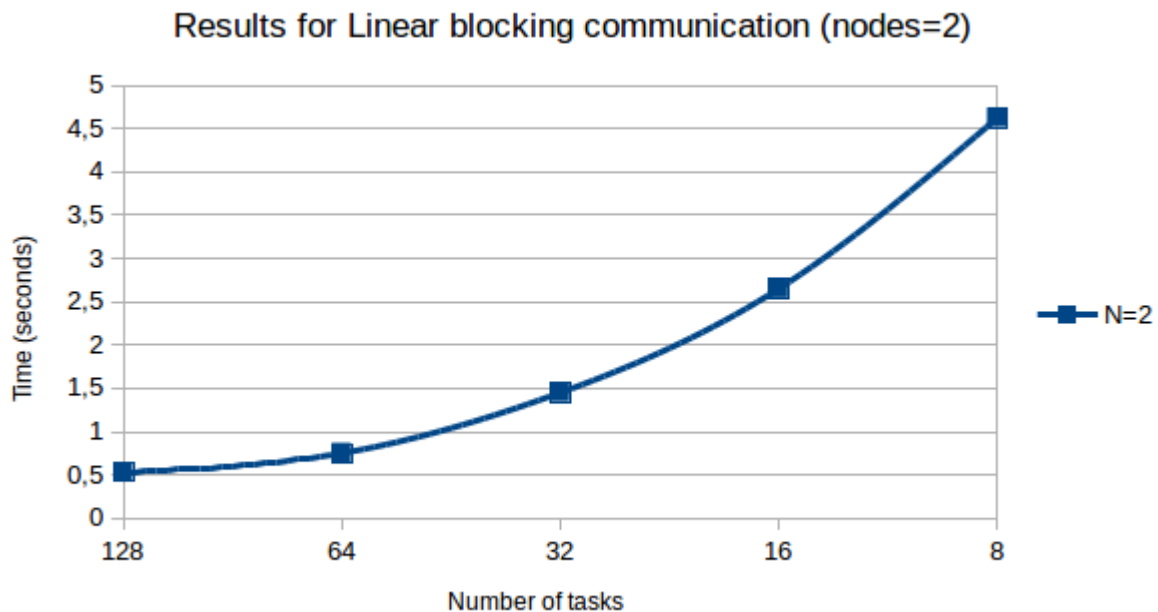


Figure 2: Linear blocking communications

Here, like in every graph we will see next, that using 128 tasks is much more faster than 8 tasks. However, the value of 128 tasks is 0.53 seconds and the value of 64 tasks is 0.75

seconds which is surprising because we would expected the time to double since we are using twice less of tasks. This behaviour is seen across all measurement, but we will talk about it at the end of the exercise.

2.1.2 Why MPI_Send and MPI_Recv are called “blocking” communication?

MPI_Send and MPI_Recv are called blocking function because the program cannot continue normal execution until the function has not complete and the data transaction (sent or received) is not completed. Therefore, there is a need that the other node is listening/sending the data at the same time. This kind of function is useful to avoid synchronization between nodes.

2.1.3 What is the MPI function for timing?

The function for timing is *MPI_Wtime()*. At first we tried to use the bash command *time* but we realized that this function is counting the time for the super-computer to initialize the whole program. So the execution time is drowned in other system time. That’s why it is crucial to use *MPI_Wtime()* to have the real execution time.

2.2 MPI Blocking Communication & Binary Tree Reduction Communication Algorithm

2.2.1 Results

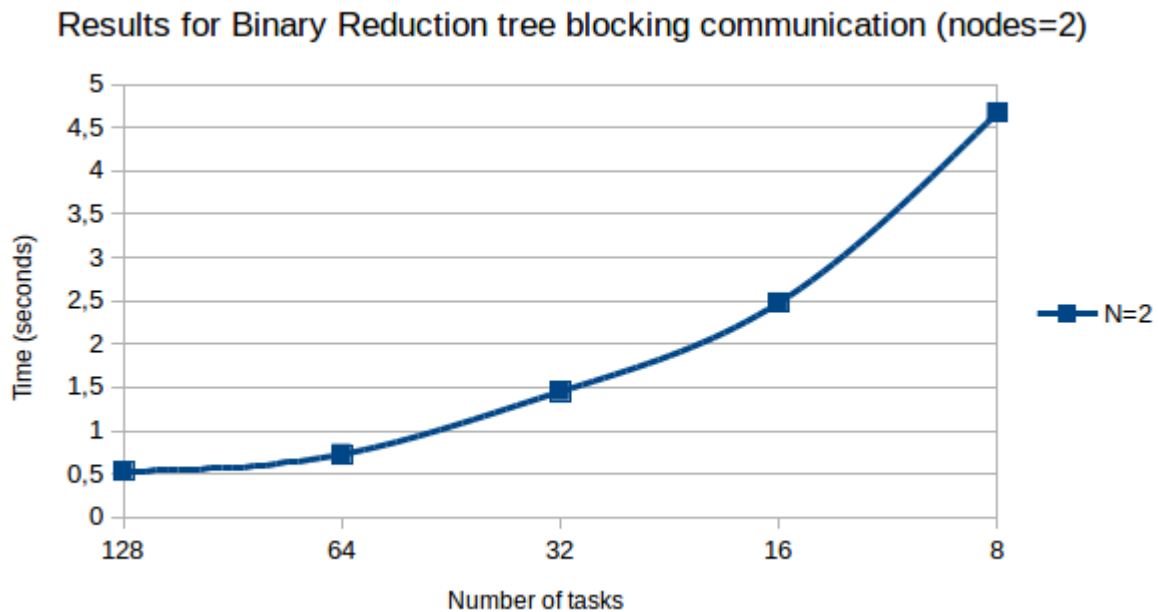


Figure 3: Binary tree reduction communication

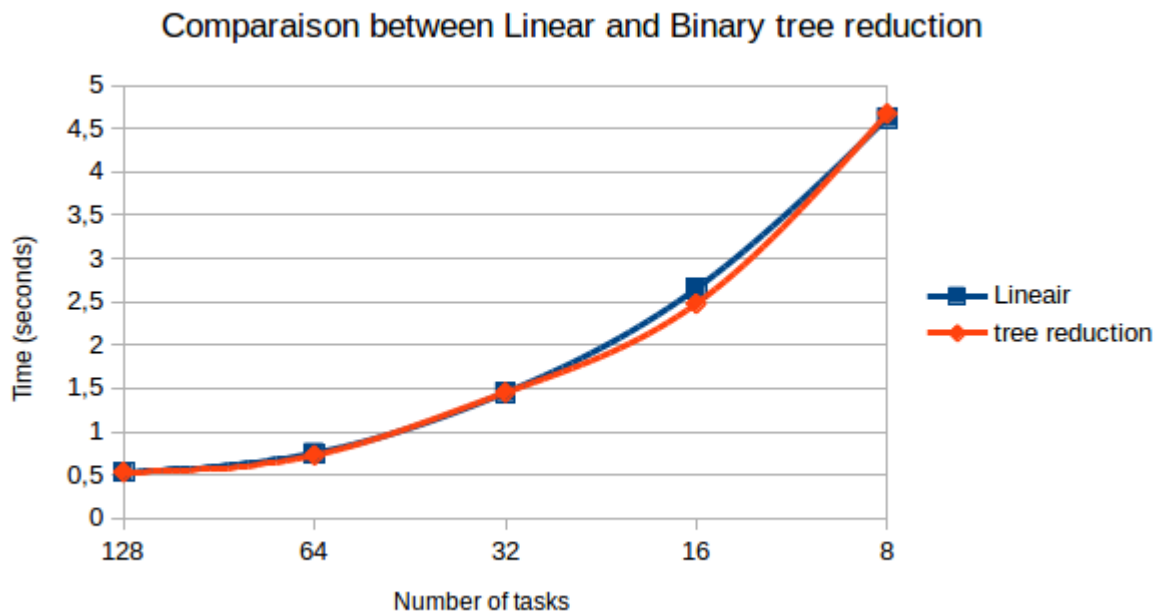


Figure 4: Comparison between binary and linear reduction

2.2.2 How does the performance of binary tree reduction compare to the performance of linear reduction?

Here we can see a slightly difference between the tree reduction and the linear reduction. However the difference is much more visible with more nodes. In this graph we used 2 nodes and the performance of the supercomputer to have 64 tasks on each node.

2.2.3 Increasing the number of processes, which approach (linear/tree) is going to perform better? Why? Think about the number of messages and their costs.

We can assume that there is a fixed time cost for sending the message and a variable time cost depending of the kind of the data. In our case, since we only send *double* type, we can simplify. Since the function is blocking, one node can only listen/send at one node at the time. In the linear algorithm, the time will be $N * time$ but the tree reduction is $\log_2(N) * time$. So when N , the number of node, increase, the reduction tree will be much more efficient than the linear.

2.3 MPI Non-Blocking Communication & Linear Reduction Algorithm

2.3.1 Results

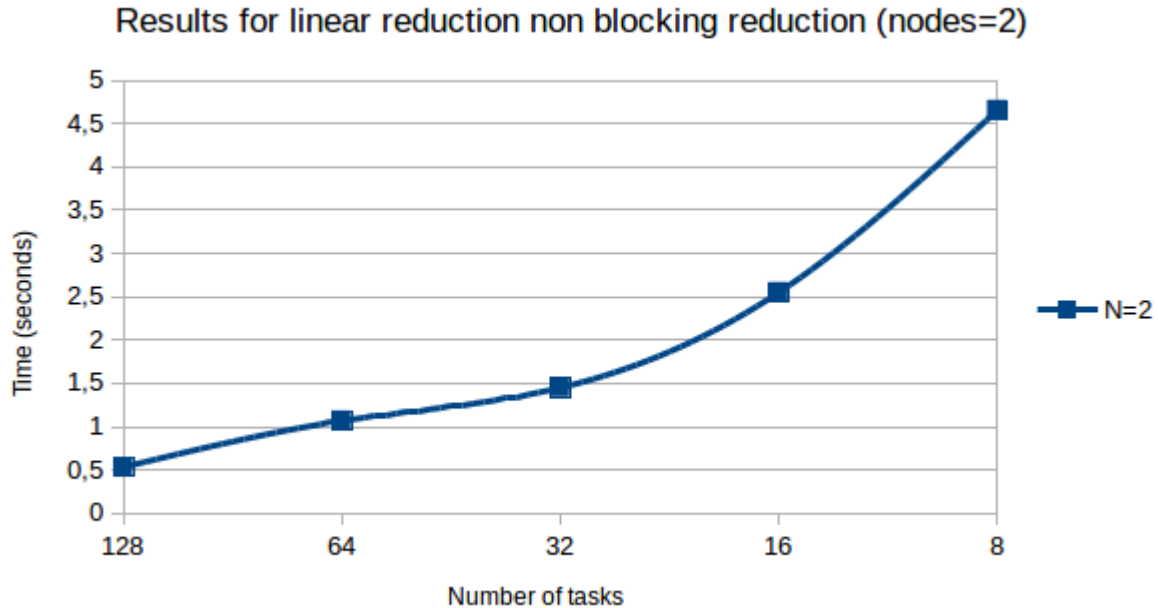


Figure 5: Linear non blocking reduction

2.3.2 What are the MPI functions for non-blocking communication? What does the ‘I’ in front of function name stand for in non-blocking communication?

The function for the non-blocking communication are *MPI_Irecv()* and *MPI_Isend()*. The ‘I’ stands for “Immediate return” which is another way of saying that they are non blocking. But with this process, the problem is the synchronization. In the case of a “sending node” which will send the data and will be over for the program, we want to be sure that the node sent its data before exiting the program. That is why we use *MPI_Wait()*.

2.3.3 How the performance of non-blocking communication compares to the performance of blocking communication. e.g. performance results in 2.1?

2.4 MPI Collective: MPI_Gather

2.4.1 Results

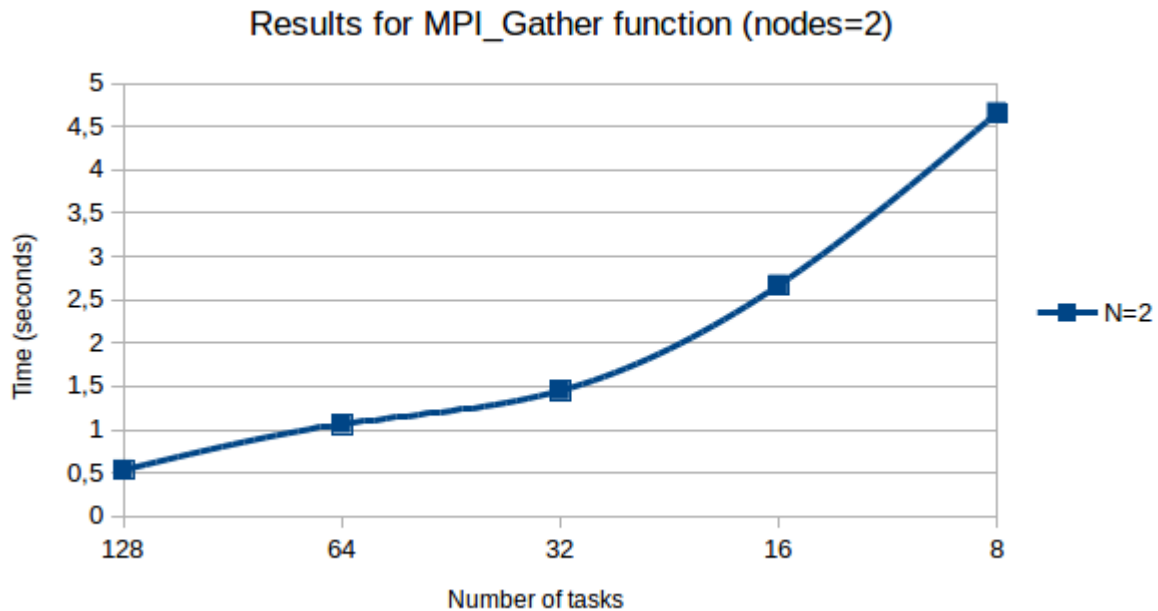


Figure 6: MPI_Gather

2.4.2 Which collective function we could use to send the final π to all the other processes?

Instead of *MPI_Gather()* we could use *MPI_Allgather()* which is a function All to All (All to one for Gather). In this function, all the nodes will have the result from every node but only the node 0 will print the result. That allows a much simpler program to read, but the performance might be influenced for heavy data (personalized structures or strings for example). Moreover, it use "a lot" of memory on every node.

2.5 MPI Collective: MPI_Reduce

2.5.1 Results

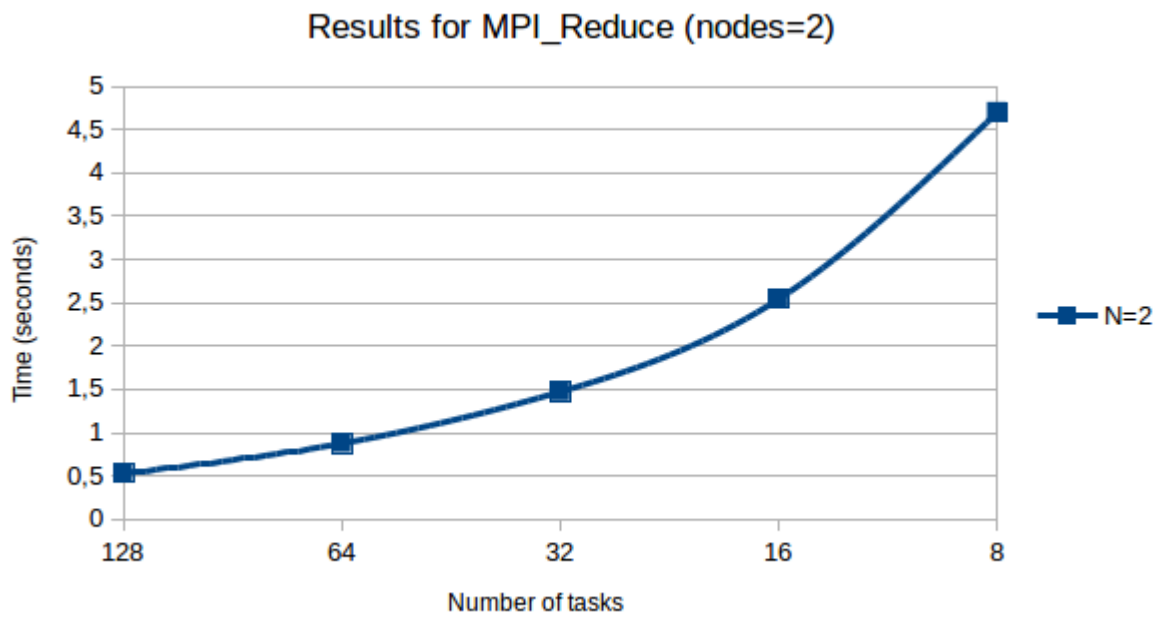


Figure 7: MPI_Reduce

2.6 MPI Windows and One-Sided Communication & Linear Reduction Algorithm

2.6.1 Results

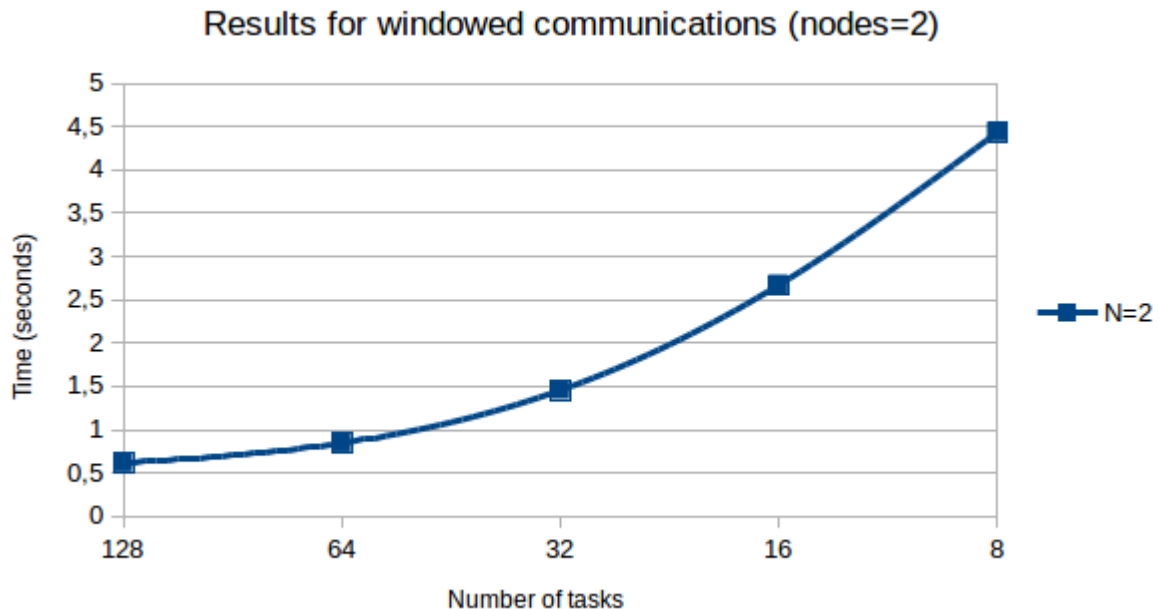


Figure 8: Windowed communications

2.6.2 Which approach gives the best performance among the 2.1 - 2.6 cases? What is the reason for that?

With our execution the best approach is the linear reduction. We would have expected an other result like the binary tree reduction. However it looks like the earned time in the reduction is consumed by the computation of source and destination. Moreover, all the computations that we made are linear so every task roughly at the same time, so they are almost already synchronized. With non linear computations, other techniques will be useful.

2.6.3 Which algorithm or algorithms do MPI implementations use for reduction operations? You can research this on the WEB focusing on one MPI implementation.

One algorithm used for the reduce function is a binary tree as we used¹. And now that we take a closer look, we have (almost) the same values between our results of binary tree and reduce's results, with a small advantage for reduce. It's maybe due to a better implementation of the binary tree by the MPI development team.

¹Rolf Rabenseifner, 2004 [PDF], acceded on the 19th may.
https://link.springer.com/content/pdf/10.1007/978-3-540-24685-5_1.pdf

2.7 Comments

For the purpose of this paper, all the previous graph are with 2 nodes. But we ran also with several nodes and see the results. First here the results.

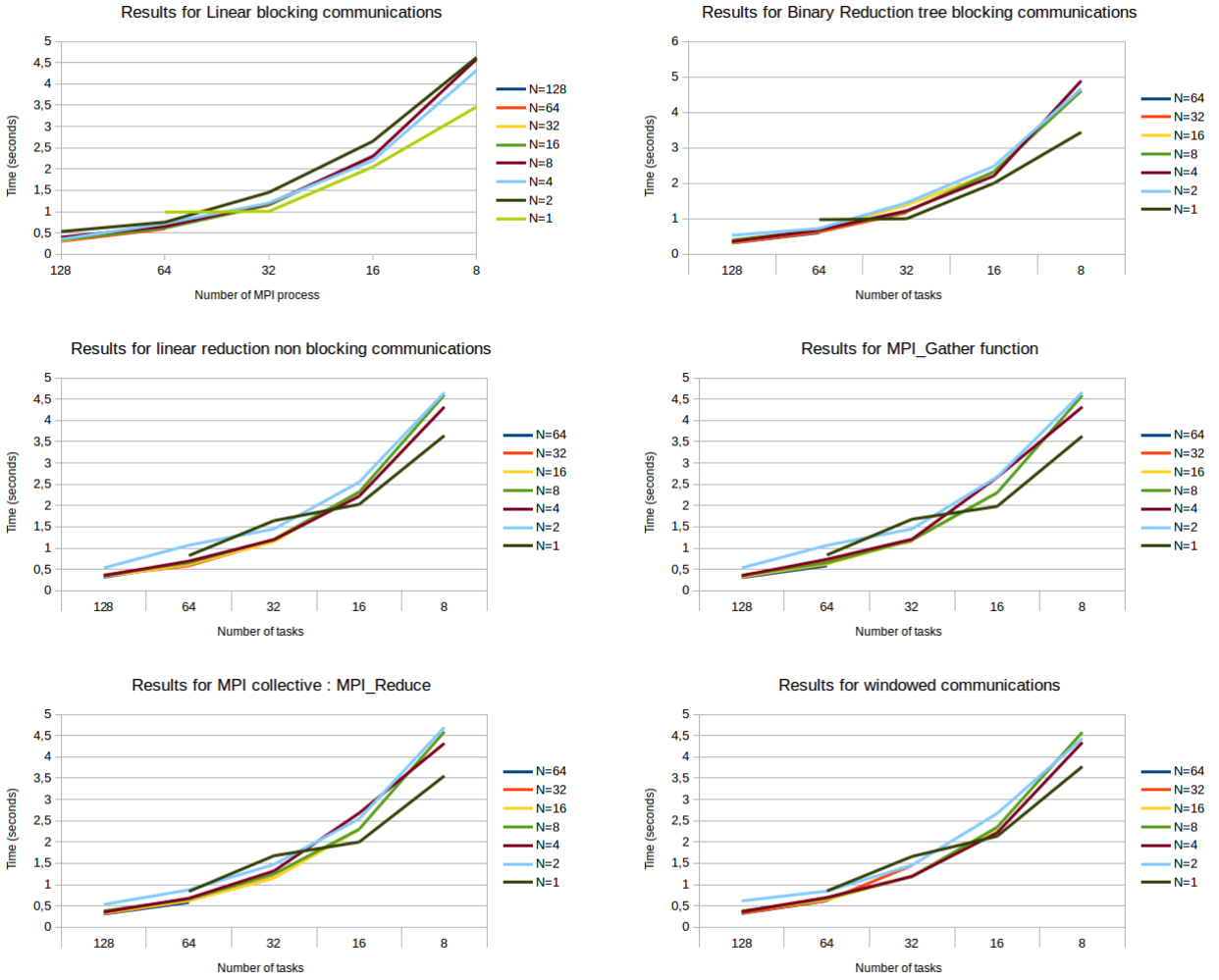


Figure 9: Global results

Here we can see that the curve with one node almost always finish above all other curves at 64 tasks (least efficient). Maybe there is a reduction of performances when a node is loaded with 64 tasks because all the task can't be run at same time.

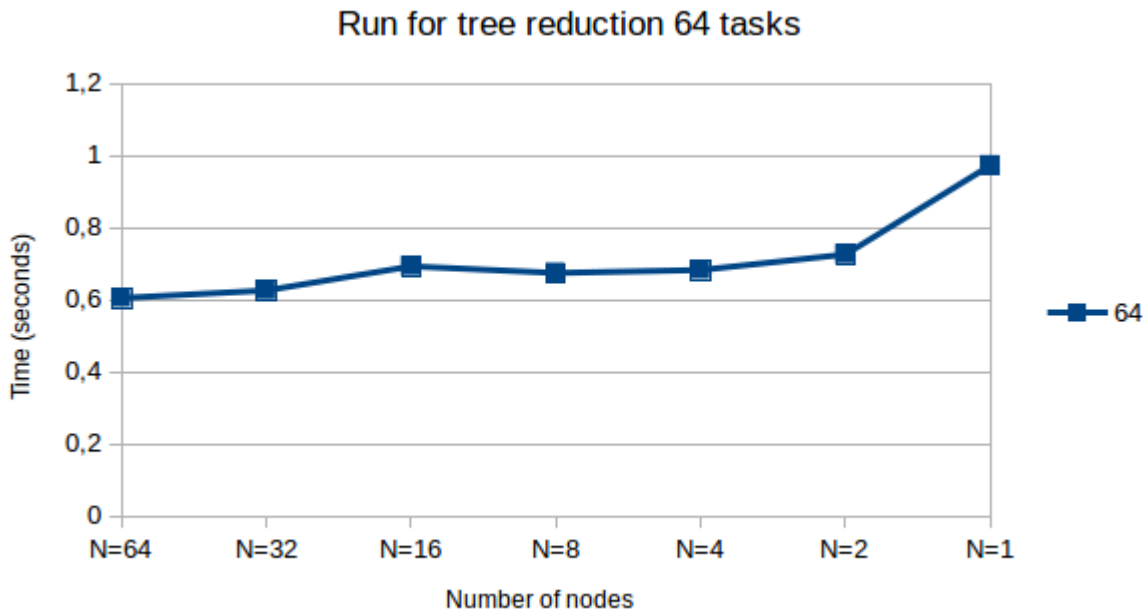


Figure 10: Run of binary tree reduction with 64 tasks given the number of nodes used

With this plot we can see a clear impact when a node is loaded with 64 tasks. We can see that the time of execution is genuinely the same for a number tasks lower than 32 per node, but there is a strong difference for the execution time with 64 tasks. A more precise study could show the impact and could measure it.

3 Exercice 3

3.1 Plots

After 5 runs of the ping-pong.c, it's unclear whether inter-node communication or intra-node communication is faster.

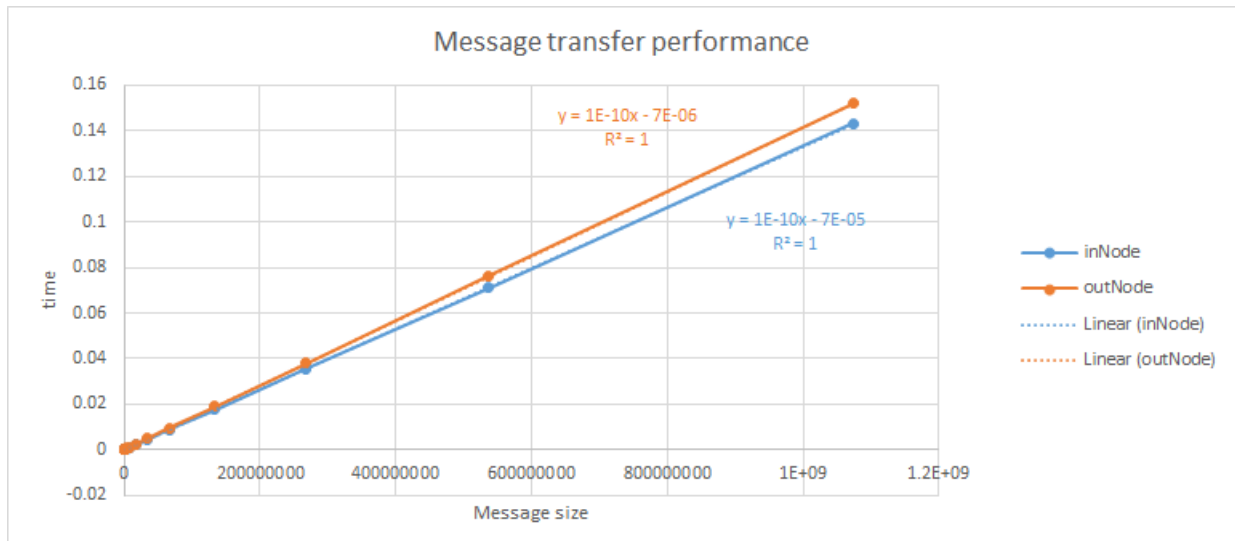


Figure 11: outNode faster

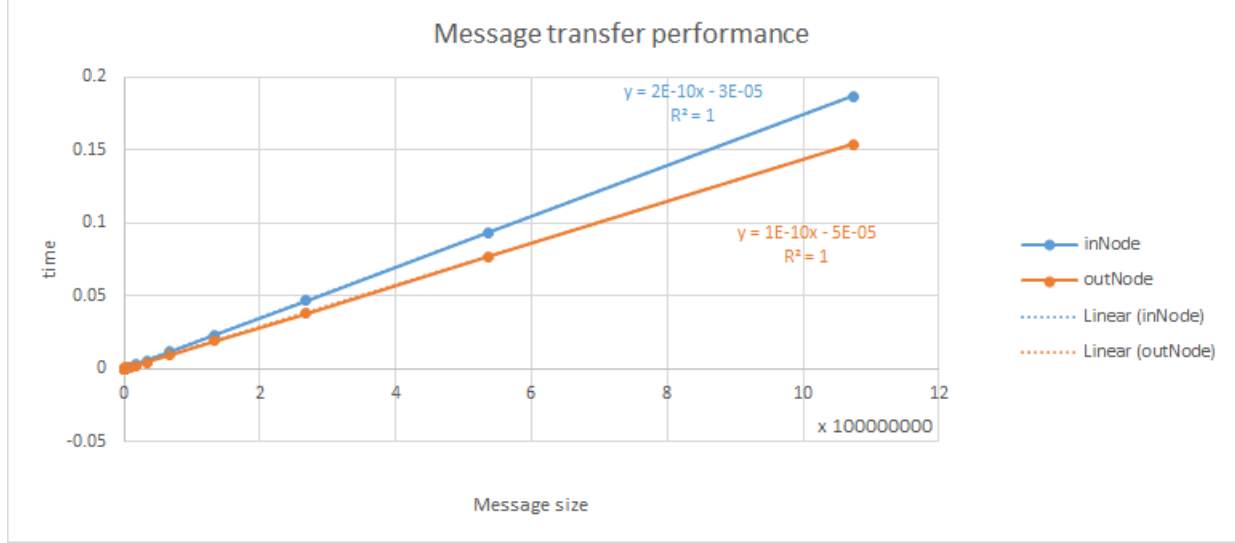


Figure 12: inNode faster

3.2 Bandwidth and latency

First figure:

	Bandwidth (Bytes)	Latency (s)
inNode	7.501750883496759e+09	-6.999058105083653e-05
outNode	7.061578299340122e+09	-6.968965292769078e-06

Second figure:

	Bandwidth (Bytes)	Latency (s)
inNode	5.744371877564083e+09	-3.382528598616390e-05
outNode	6.971617166135649e+09	-5.208039336964360e-05

As we can see the latency doesn't mean anything since it is negative. All of our experiment gave negative latency.

However we can see that the bandwidth between 2 node is roughly the same each time but the bandwidth inside a node varies.

3.3 Questions

Where should we get the nominal Beskow network bandwidth and latency values?

If we were to compare the values between intra-node and inter-node, I would say that for intra-node, switching from the first to the 2nd processus to answer the ping cost time while in inter-node configuration, the physical distance would slow down the rate.