University of Illinois at Chicago

Parallel Processing

ECE 566

A.Y. 2017 - 2018

Assignment 1

Cipoletta Antonio 655452970

Gualco Andrea 651262476

Contents

1 Introduction

The main goal of this assignment is to implement a parallel algorithm to sum an array of N integers over k processes. The k processes can be assigned 1:1 to physical processors or multiple processes share the same physical resources. The used cluster EXTREME allows to decide how to distribute the processes over the nodes, meaning for example that 16 processors can be allocated to 1 node of the cluster or divided between multiple nodes. For this reason multiple configurations are possible. Since the problem is very easy and not computationally expensive it has been tried to understand how the set-up of the cluster affects the performance, in particular the communication part.
The problem has been formulated using two logical topologies: a ring and an hypercube. Actually in this case being the task and data partition pretty straightforward it doesn't need the support of a logical architecture especially because a general message passing library `MPICH2` has been used.
Considering only the computational point of view the best way to partition the task and the data is to divide the total array in k part and assign it on each processor than recombine the partial sum in a way such that the resulting DFG is more balanced as possible. From the communication point of view is preferable to have an hypercube because both in fase of sending the data from the source node to all the others (Scatter operation) and in fase of reducing the partial sum to the final result (Reducing) the message exchanging is between nodes 1-hop away.
Two groups of experiments have been done:

1. Given a physical and logical parallel architecture change the dimension N of the array. The aim is to understand how the overhead both in terms of computation and in terms of comunication scale in a different way based on the approach used and the set-up of the cluster.

2. Given a value $N >> 1$ of the array and a logical architecture understand how scaling the available physical resources affects the overall performance.

2 Formulation

The sum of an array of length N has been implemented in the Extreme cluster using a hypercube topology.
First in each program the MPI library has been initialized using the function `MPI_Init`. After that, the topology has been declared using the function `MPI_Cart_create`.
Other functions used are:

- `MPI_Comm_size` in order to retrieve the number of processor used (k);
- `MPI_Comm_rank` in order to save the rank of each processor in the topology;
- `MPI_Cart_coords` in order to save the coordinates of each processor.

Every program, before the `return`, needs to call the function `MPI_Finalize` in order to close the MPI environment.

These functions are used in every approach tested. In particular the same task hsa been implemented usign 3 different approaches:

1. using scatter (one-to-all personalized broadcast) and all-to-one reduce;
2. using scatter (one-to-all personalized broadcast) and gather;
3. using scatter (one-to-all personalized broadcast) and all-to-one reduce, but implementing them using send and receive operations.

For this reason in approach 1 and 2 library functions has been used. In both the `MPI_Scatterv` has been used to implement the one-to-all personalized broadcast. In approach 1 has been used the `MPI_Reduce` to sum the data directly inside root node, while in 2 has been used `MPI_Gather` to collect the data in root node and them sum them.
The approach 3 is equivalent to approach 1 in term of functionality, but the scatter and reduce are implemented using the `MPI_Send` and `MPI_Recv` functions.

3 Parameter ranges

The input parameters of the programs are:

- n that is the length of the array to sum;
- k that is the number of physical processor used;
- p that is the number of virtual processor used.

First, different lengths n of the array has been used to test each number of physical and virtual processor used. In particular these values has been been used:

- $2^{10} = 1024$
- $2^{12} = 4096$
- $2^{14} = 16,384$
- $2^{16} = 65,536$
- $2^{18} = 262,144$
- $2^{20} = 1,048,576$
- $2^{22} = 4,194,304$

Then it has been decided the different number of physical and virtual processor used. In particular the choices are:

- nodes = 1 and processor_per_nodes = 1, so that $k = 1$ and $p = 16$
- nodes = 1 and processor_per_nodes = 16, so that $k = 16$ and $p = 16$

- nodes = 4 and processor_per_nodes = 4, so that $k = 16$ and $p = 16$

Than with a value of $n = 6,7108,864$ it has been modified the number of processors from 2 to $7 * 2^4$ in case of the ring in a geometrical way, in case of the hypercube using only power of 2.

4 Timing measurements

In order to have a quantitative measurement of the performance for each simulation a time measurement has been done. Since the cluster is a shared machine and due to the high sources of variability the following approch has been adopted.
In each source code the main part has been identified, i.e. the two sections of the logical scatter and logical reduce/gather. This part has been executed 1000 times with a time measurement for each iteration. At the end the average time and the deviation $\sqrt{\frac{\sum_{i=1}^{1000}(time[i]-average\_time)^2}{999}}$ has been computed and used to evaluate the simulation.

5 Results and analysis experiment 1

The first set of simulations has been done to test the different software approaches on 3 set-up of the cluster but with a total number of processes equal to 16:

- 4 nodes and 4 processors per node.
- 1 node and 16 processors per node.
- 1 node and 1 processor. The logical processors are all emulated on a single processor.

The dimension of the array N has been choosed in a range 2^{10} to 2^{24} with an exponential step of 2^2, so 7 simulations for each approach.

5.1 Ring

5.1.1 Observations

- For all the three approaches the timing performance grows linearly respect to N. Both the computation part grows with N (even if is negligible) and also the communication cost since the dimension of the the data that has to be exchanged during the scatter is linearly increasing.
- The two approaches using reduce and gather obtain, considering also the deviation, pratically equal performance on all three architectures. The reason for this is that the reduction operation is very simple just the sum of two numbers and the advantage compared to do all the work at the root node, where the complexity is O(k), is negligible. In terms of comunication the last part is also negligible because the amount of data exchanged is only one integer, i.e. 4 bytes.

- The performance in case of the implementation of the algorithm using send/rcv operations are always worse than using scatter/reduce. This is reasonable both because the collective primitives in MPI should be optimized for the physical topology of the cluster and also because in our implementation both the reduce and the scatter have been done using a linear communication scheme that for sure wastes the available interconnections on the physical cluster.
- The case with one processor and one node is obviously the slower one. Even if no data has to be exchanged through the interconnection network the work is pratically serialized plus the overhead due to the primitives call and the multiple context switches between the processes. The 4 node and 4 ppn architecture is slower than the 1 node with all the 16 processor. In particular the slope of the curve is different because there is an additional time spent for each byte sent to exit from one node.
- See figures 1, 2, 3 for the relative graphs

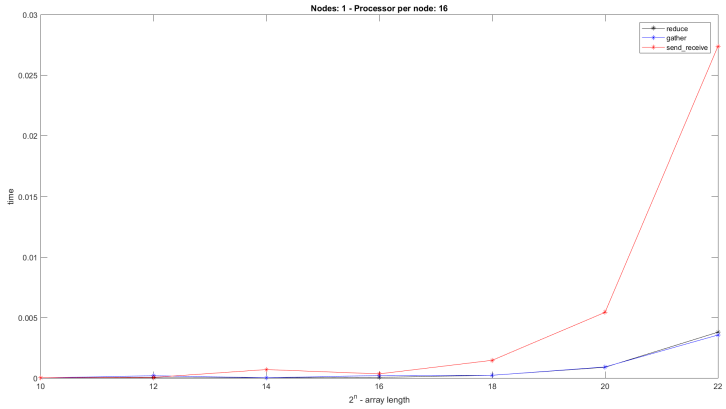

Figure 1: 1 node 16 ppn. Comparison between all the approaches in the ring

5.1.2 Timing results

In the tables 1 and 2 are presented the results and the deviations computed foreach architecture on different N. The number of repetitions of the code to take the average time are 1000

5.2 Hypercube

The same experiments have been done also on the hypercube with the same parameters as in the ring one.

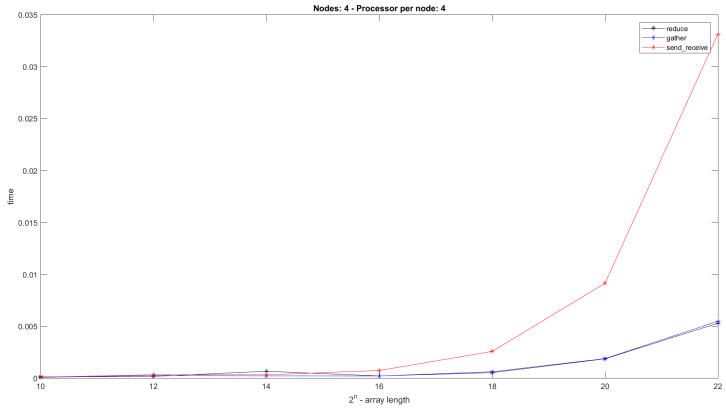

Figure 2: 4 node 4 ppn. Comparison between all the approaches in the ring

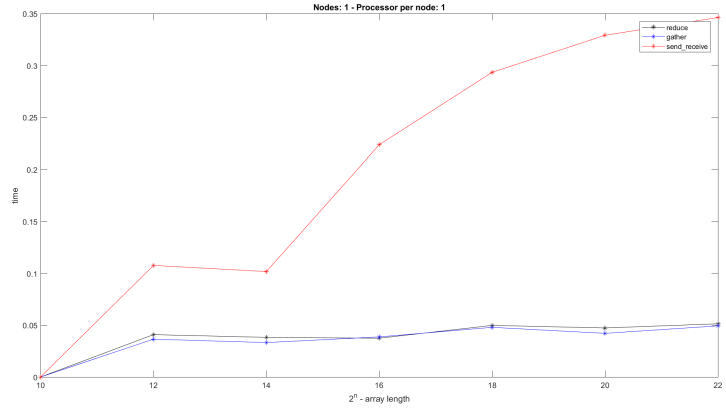

Figure 3: 1 node 1 processor. Comparison between all the approaches in the ring

5.2.1 Observations

Most of what it is reported before for the ring is also valid in case of the hypercube. One big difference is the difference between the time performance of the send/rcv implementation and the one using the collective primitive. The send/rcv implementation is still slower but not so limiting as in the case of the ring being the scatter/reduce algorithm implemented with the possibilities of having multiple communication operations concurrently. See figures 4, 5, 6 for the relative graphs

5.2.2 Timing results

In the tables 3 and 4 are presented the results and the deviations computed foreach architecture on different N. The number of repetitions of the code to take the average time are 1000.

Table 1: Scatter/reduce and Scatter/gather results ring

(a) Scatter reduce results ring

	n	avg time [s]	dev
n:4 - ppn:4	1024	0.000112	0.0001
	4096	0.000153	0.0002
	16384	0.000651	0.0091
	65536	0.000196	0.0001
	262144	0.000516	0.0001
	1048576	0.001850	0.0064
	4194304	0.005284	0.0064
n:‘1 - ppn:16	1024	0.000007	0.0000
	4096	0.000009	0.0000
	16384	0.000018	0.0000
	65536	0.000054	0.0000
	262144	0.000228	0.0633
	1048576	0.000888	0.0775
	4194304	0.003788	0.1733
n:‘1 - ppn:1	1024	0.000007	0.0000
	4096	0.040854	0.0084
	16384	0.038218	0.0100
	65536	0.037417	0.0093
	262144	0.049748	0.0161
	1048576	0.047284	0.0216
	4194304	0.051298	0.0975

(b) Scatter gather results ring

	n	avg time [s]	dev
n:4 - ppn:4	1024	0.000063	0.0000
	4096	0.000252	0.0007
	16384	0.000205	0.0003
	65536	0.000194	0.0001
	262144	0.000602	0.0003
	1048576	0.001871	0.0065
	4194304	0.005472	0.0090
n:‘1 - ppn:16	1024	0.000007	0.0000
	4096	0.000187	0.0019
	16384	0.000019	0.0000
	65536	0.000205	0.0020
	262144	0.000224	0.0000
	1048576	0.000915	0.0001
	4194304	0.003552	0.0002
n:‘1 - ppn:1	1024	0.000007	0.0000
	4096	0.036480	0.0106
	16384	0.033209	0.0111
	65536	0.038656	0.0085
	262144	0.047886	0.0177
	1048576	0.042068	0.0247
	4194304	0.049310	0.1017

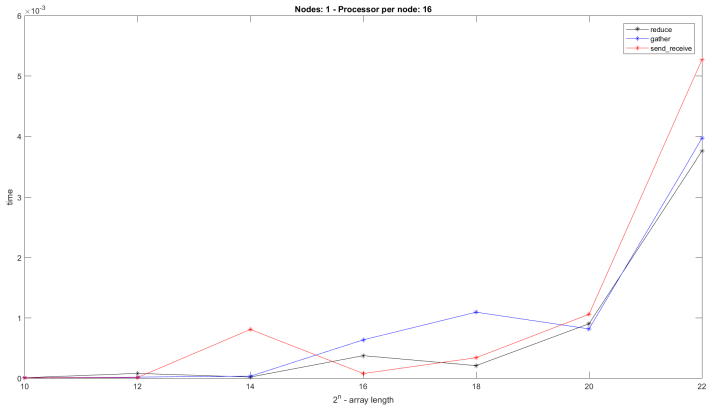

Figure 4: 1 node 16 ppn. Comparison between all the approaches in the hypercube

	n	avg time [s]	dev
n:4 - ppn:4	1024	0.000085	0.0000
	4096	0.000316	0.0006
	16384	0.000309	0.0006
	65536	0.000724	0.0001
	262144	0.002564	0.0006
	1048576	0.009137	0.0005
	4194304	0.033114	0.0014
n:‘1 - ppn:16	1024	0.000016	0.0000
	4096	0.000051	0.0003
	16384	0.000699	0.0047
	65536	0.000342	0.0001
	262144	0.001463	0.0022
	1048576	0.005427	0.0003
	4194304	0.027368	0.0008
n:‘1 - ppn:1	1024	0.000014	0.0000
	4096	0.107538	0.0244
	16384	0.101642	0.0209
	65536	0.223574	0.0528
	262144	0.293329	0.1719
	1048576	0.329045	0.4299
	4194304	0.346142	0.9668

Table 2: Scatter Reduce with MPI_Send & MPI_Recv results ring

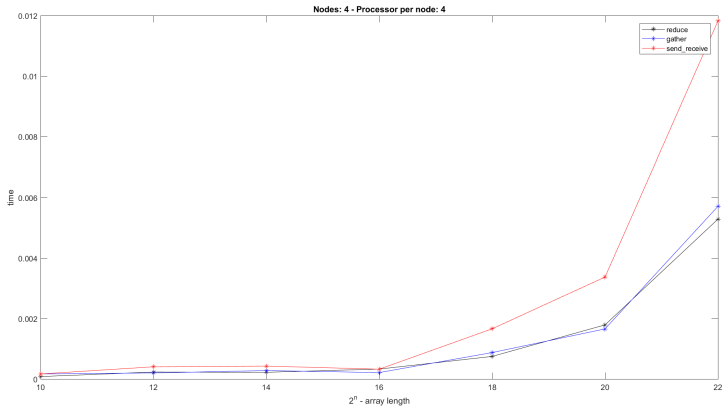

Figure 5: 4 node 4 ppn. Comparison between all the approaches in the hypercube

5.3 Ring vs Hypercube

5.3.1 Observations scatter/reduce

The performance in the three architecture are practically equal even if the two different topologies are created using `MPI_Cart_create` because the scatter and reduce function are

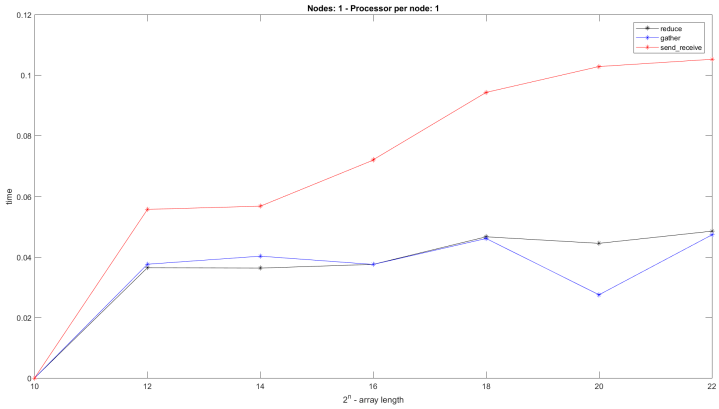

Figure 6: 1 node 1 processor. Comparison between all the approaches in the hypercube

implemented using collective primitives on the entire world of processors. For this reason the communication scheme on the topogies is the same leading to a similar behavior of the curves.
See figures 7, 8, 9 for the relative graphs

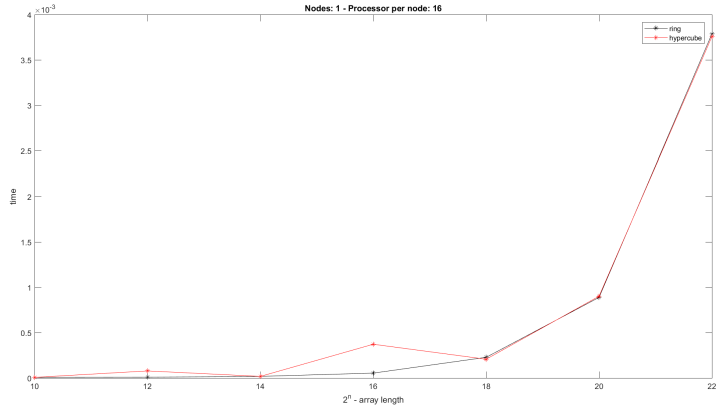

Figure 7: 1 node 16 ppn. Comparison scatter/reduce between ring and hypercube

5.3.2 Observations snd/rcv

In this case the the ring topology has worse performance than the hyperecube topology in all the three architectures examined. The overhead in the ring grows with an higher slope than the hypercube. Moreover the time in the ring is higher than the hypercube because the execution in the ring is almost serialized due to the topology, while in the hypercube there can be multiple communications at the same time.
See figures 10, 11, 12 for the relative graphs

Table 3: Scatter/reduce and Scatter/gather results hypercube

(a) Scatter reduce results hypercube

	n	avg time [s]	devi
n:4 - ppn:4	1024	0.000081	0.0000
	4096	0.000231	0.0002
	16384	0.000223	0.0002
	65536	0.000329	0.0002
	262144	0.000747	0.0064
	1048576	0.001789	0.0004
	4194304	0.005283	0.0003
n:1 - ppn:16	1024	0.000007	0.0000
	4096	0.000078	0.0008
	16384	0.000018	0.0000
	65536	0.000372	0.0025
	262144	0.000207	0.0000
	1048576	0.000901	0.0006
	4194304	0.003757	0.0003
n:1 - ppn:1	1024	0.000007	0.0000
	4096	0.036459	0.0088
	16384	0.036323	0.0097
	65536	0.037555	0.0086
	262144	0.046665	0.0171
	1048576	0.044504	0.0223
	4194304	0.048529	0.0983

(b) Scatter gather results hypercube

	n	avg time [s]	dev
n:4 - ppn:4	1024	0.000169	0.0002
	4096	0.000201	0.0001
	16384	0.000280	0.0002
	65536	0.000213	0.0001
	262144	0.000874	0.0064
	1048576	0.001656	0.0003
	4194304	0.005705	0.0110
n:‘1 - ppn:16	1024	0.000007	0.0000
	4096	0.000017	0.0002
	16384	0.000032	0.0004
	65536	0.000634	0.0031
	262144	0.001092	0.0032
	1048576	0.000815	0.0001
	4194304	0.003966	0.0002
n:‘1 - ppn:1	1024	0.000007	0.0000
	4096	0.037603	0.0082
	16384	0.040265	0.0122
	65536	0.037530	0.0093
	262144	0.046106	0.0188
	1048576	0.027499	0.0271
	4194304	0.047313	0.0915

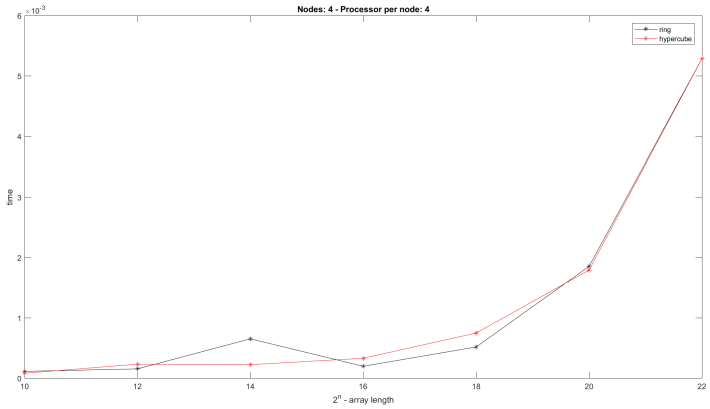

Figure 8: 4 node 4 ppn. Comparison scatter/reduce between ring and hypercube

	n	avg time [s]	dev
n:4 - ppn:4	1024	0.000167	0.0005
	4096	0.000404	0.0004
	16384	0.000427	0.0064
	65536	0.000325	0.0003
	262144	0.001660	0.0099
	1048576	0.003370	0.0090
	4194304	0.011817	0.0090
n:‘1 - ppn:16	1024	0.000005	0.0000
	4096	0.000010	0.0000
	16384	0.000806	0.0036
	65536	0.000076	0.0003
	262144	0.000338	0.0005
	1048576	0.001058	0.0001
	4194304	0.005267	0.0003
n:‘1 - ppn:1	1024	0.000007	0.0000
	4096	0.055728	0.0098
	16384	0.056783	0.0140
	65536	0.072009	0.0209
	262144	0.094263	0.0354
	1048576	0.102808	0.1087
	4194304	0.105205	0.2583

Table 4: Scatterreduce with with MPI_Send & MPI_Recv results hypercube

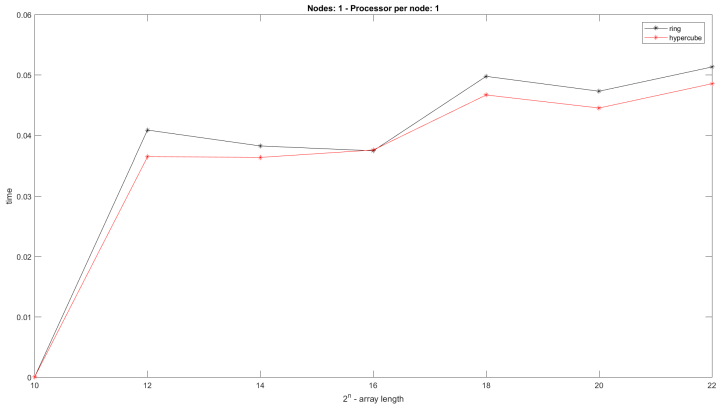

Figure 9: 1 node 1 processor. Comparison scatter/reduce between ring and hypercube

6 Results and analysis experiment 2

In this second experiment, the approach is the complementary of the first one: this time the dimension of the input array N is constant and equal to 6,7108,864. The number of available physical resources is the variable paramenter. The goal is to understand how the number

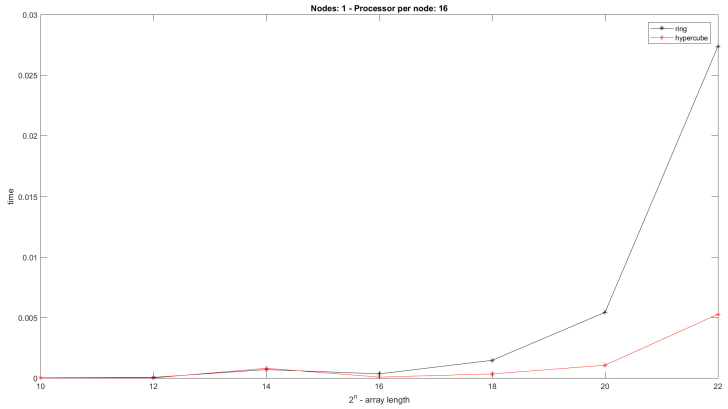

Figure 10: 1 node 16 ppn. Comparison snd/rcv between ring and hypercube

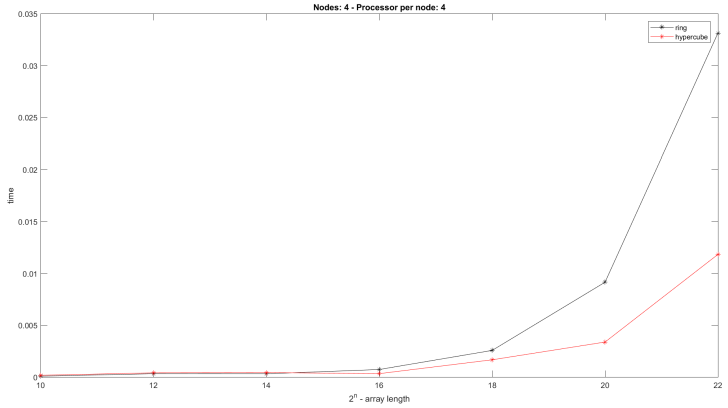

Figure 11: 4 node 4 ppn. Comparison snd/rcv between ring and hypercube

of physical resources available influence the time. In particular the trade-off between the speed-up in the computation due to the parallelization and the overhead introduced by the inter-processor communication.
It has been examined only the scatter/reduce and send/receive approaches, because they performe the same task in a different way.

6.1 Ring

6.1.1 Observations

In figure 13, it is shown the graph of the scatter/recduce on a ring using different number of processors.
It has to be noticed that the send/receive approach grows linearly with the number of processor, while the scatter/reduce approach is almost constant.
The send/receive case is divergent because the parallelization is not exploited well due to

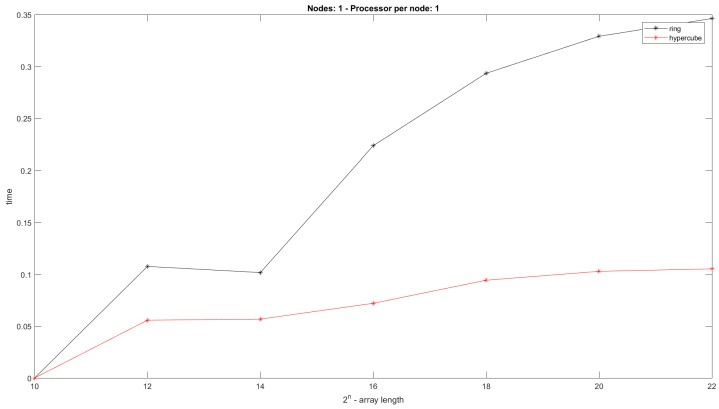

Figure 12: 1 node 1 processor. Comparison snd/rcv between ring and hypercube

topology configuration, so an increasing number of processors will increase the communication overhead

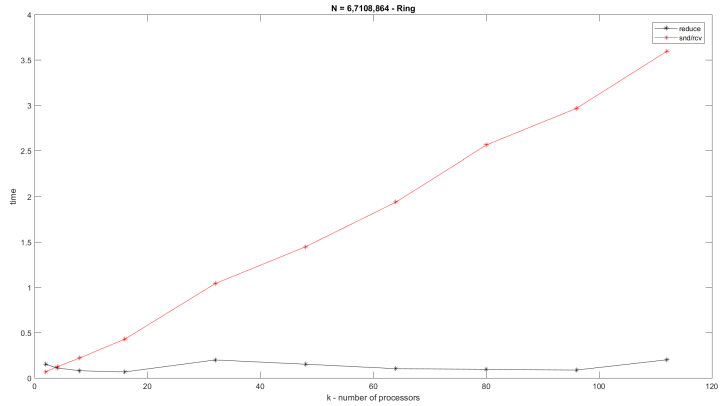

Figure 13: Comparison between scatter/reduce and snd/rcv ring

6.2 Hypercube

6.2.1 Observations

In figure 14, it is shown the graph of the scatter/recduce on a hypercube using different number of processors.
The behavior of the two curves is almost the same. The convexity of the curves is due to the fact that initially increase the number of processors decrease the computation time more than the comunication overhead. At some point, the gain in computation speed-up is overcome by the loss due to the comunication overhead.

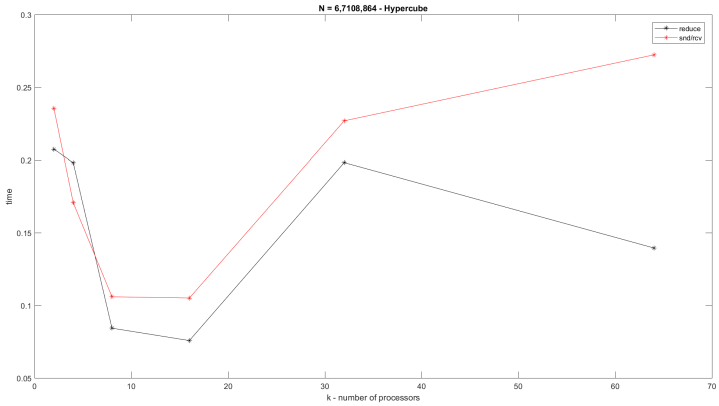

Figure 14: Comparison between scatter/reduce and snd/rcv hypercube

7 Lessons

From this assignment, we learnt how to use the `MPI` library. In fact in order to use the collective functions we needed to read the documentation deeply.
Moreover we also learnt how to use the EXTREME cluster of UIC.
Last but not least, we has the chance to implement the different algorithm seen during classes, understand deeper how they work and the issues related to each topology.