Riccardo Basso - Davide Ponzini

MPI Barrier

**Introduction**

For the MPI Barrier Lab, we always worked on ssh. Every test, modify to the cpp or creation of scripts, has been done on ssh and not in local.

The most difficult part was the exercise 6, especially because we had to implement our own barrier and we had to think about a good way to do it. We will provide also some graphic about it, so that we can compare ex5 and ex6.

**Ex1 - Ex2 - Ex3 - Ex4**

These exercises were quite easy for us and did not take a lot of time. We just had to compile and run or sometimes modify short lines of code.

But we’ve experimented how to work with MPI and how to use barriers.

All the files are inside the various exercise folders inside the archive.

Also the output.txt files are provided, so that you can compare the performance.

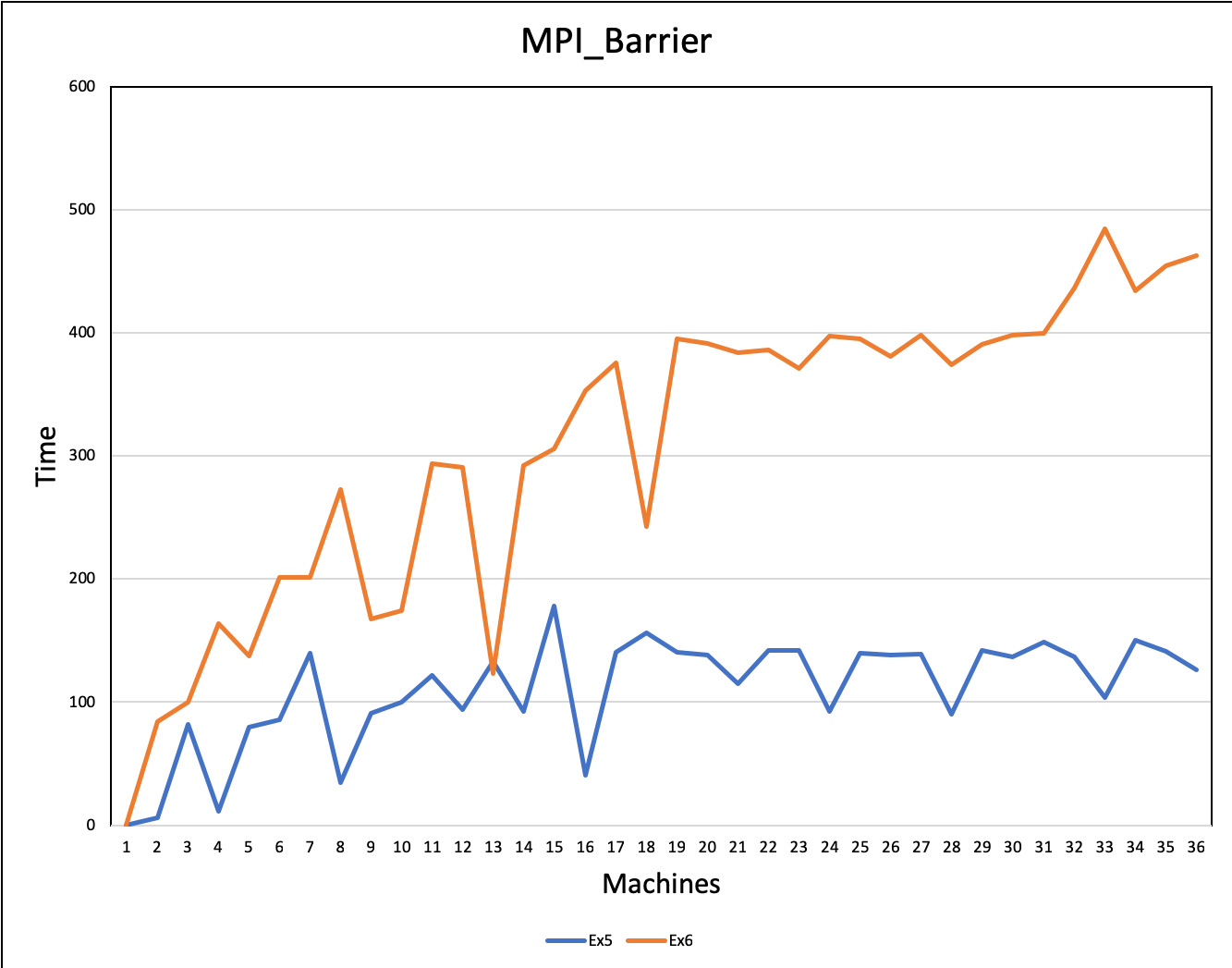
**Ex5 - Ex6**

These are the most difficult exercises because for the Ex6 we had to implement our own barrier, without using the pre-made function.

We have also evaluated performance as a function of the number of processes, and compared it against the performance of the native MPI\_Barrier() function used in the ex5.

If you want to run both Ex5 or Ex6, a script.sh is provided on both folders. You just have to write the command ./script.sh on terminal and there you go. Alternatively, for Ex6 you can run make n=5 and you compile and execute on 5 machines (5 is just an example, you can insert the number that you want until 36).

Here we provide our results, plotting the Time on the Vertical axes and the number of Machines on the horizontal one. The blue line indicates the performance of Ex5 pre-made Barrier function, the orange one instead is the performance of the Barrier implemented by us:



Every single data that we’ve plotted here is available inside the archive. The two output files are proved in folders Ex5 and Ex6.

As you can see, performance for the MPI\_Barrier() function in the exercise 5 are better than the one implemented by us. Also remember that the time is computed by multiplying \*1e6, so that we get microseconds.