Parallel programming for HPC: exam project
Exercise 3: MPI one-sided communication for the Jacobi method

Student: Isac Pasianotto

2024-06

0 Requirements

The task this assignment aims to solve is the parallelization of the Jacobi method using MPI one-sided communication. The starting point is the already parallelized version of the Jacobi method using MPI point-to-point communication, which is the solution of the previous exercise.

1 Implementation

Since the focus of this exercise is the communication between processes, all the section of the code responsible for the GPU computation has been removed, and the faster version of the output savin (MPI I/O) has been used.

The amount of data to be exchanged between processes is the same as in the previous exercise, but the communication is done using remote memory access (RMA) operations.

Since no specification was given about how precisely this communication should be performed, the following case has been implemented:

	Operation used to retrieve Data	
Number of windows	1 Window - PUT	1 Window - GET
	2 Windows - PUT	2 Windows - PUT

Table 1: Different implementations of RMA

With 1 window, I mean that the process will open a `MPI_Win` object exposing to other processes the whole matrix, while with 2 windows, the process will open two `MPI_Win` objects, one for the first row and one for the last row of the matrix. The notation PUT and GET should be self-explanatory: the first one is used to write data to the remote process with `MPI_Put`, while the second one is used to read data from the remote process with `MPI_Get`.

2 Results

To keep consistency with the previous exercises, the code has been run for two different matrix sizes and a fixed number of iterations:

- *Matrix size:* $1,200 \times 1,200$, *Iterations:* 10
- *Matrix size:* $12,000 \times 12,000$, *Iterations:* 10

The code was run on the Leonardo cluster on DCGP nodes, which are used for CPU computing, spawning one MPI process per node, and let OpenMP use all the available cores in the node.

The results are represented in the plots in the following pages, but can be summarized as follows:

- `MPI_Get` function seems to perform a little better than `MPI_Put`. However, this difference becames less evident scaling up the number of nodes and the matrix size
- The communication time of the one and two windows cases seems to be almost the same. The initialization time required to open the window is less in the one-window case
- The `RMA` approach seems to have a faster communication compared with `MPI_Send` + `MPI_Recv`. But the initialization time obliterates this advantages.
- Probably for a higher number of iteration RMA can be the best choice.

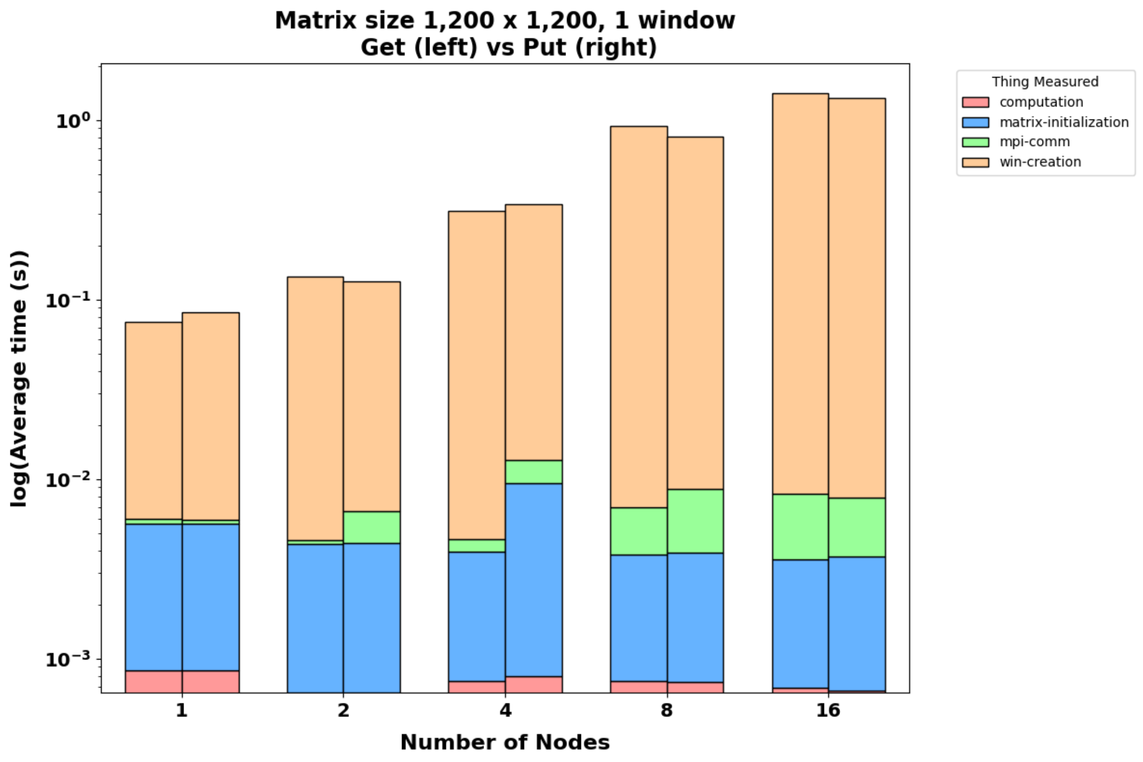

Figure 1: Matrix size: $1,200 \times 1,200$, 1 `MPI_Win`.

2.1 `MPI_Put` vs `MPI_Get`

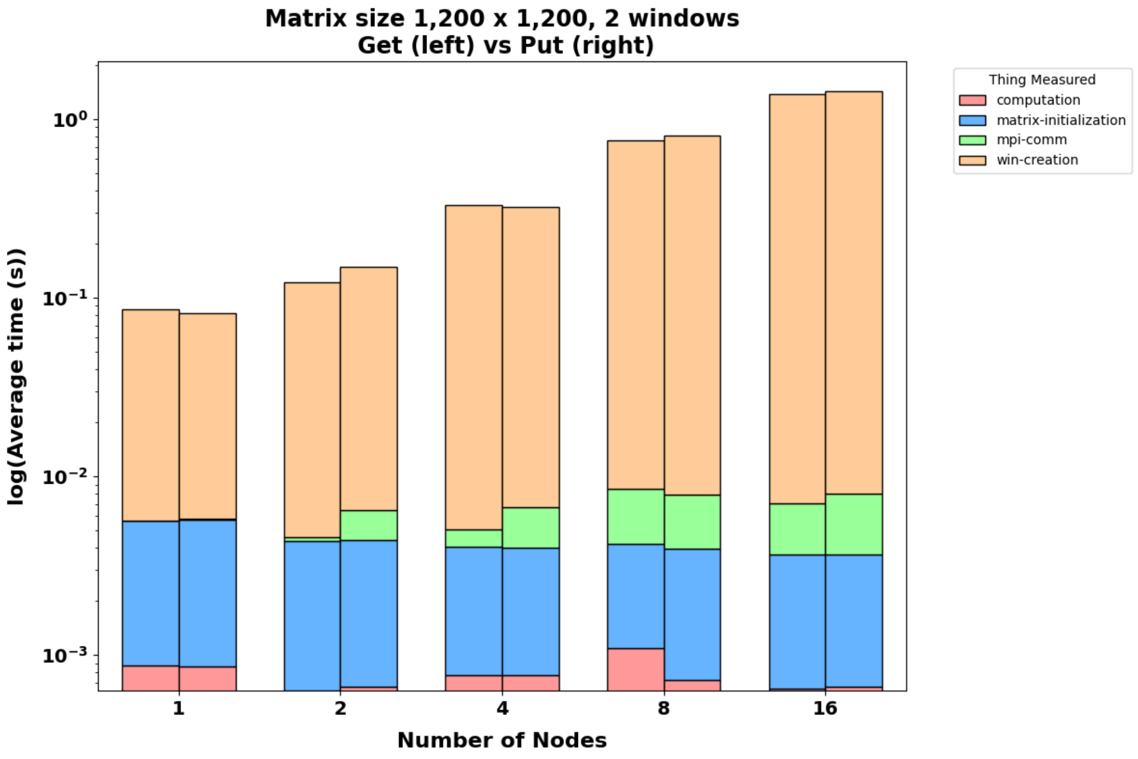

Figure 2: Matrix size: $1,200 \times 1,200$, 2 `MPI_Win`.

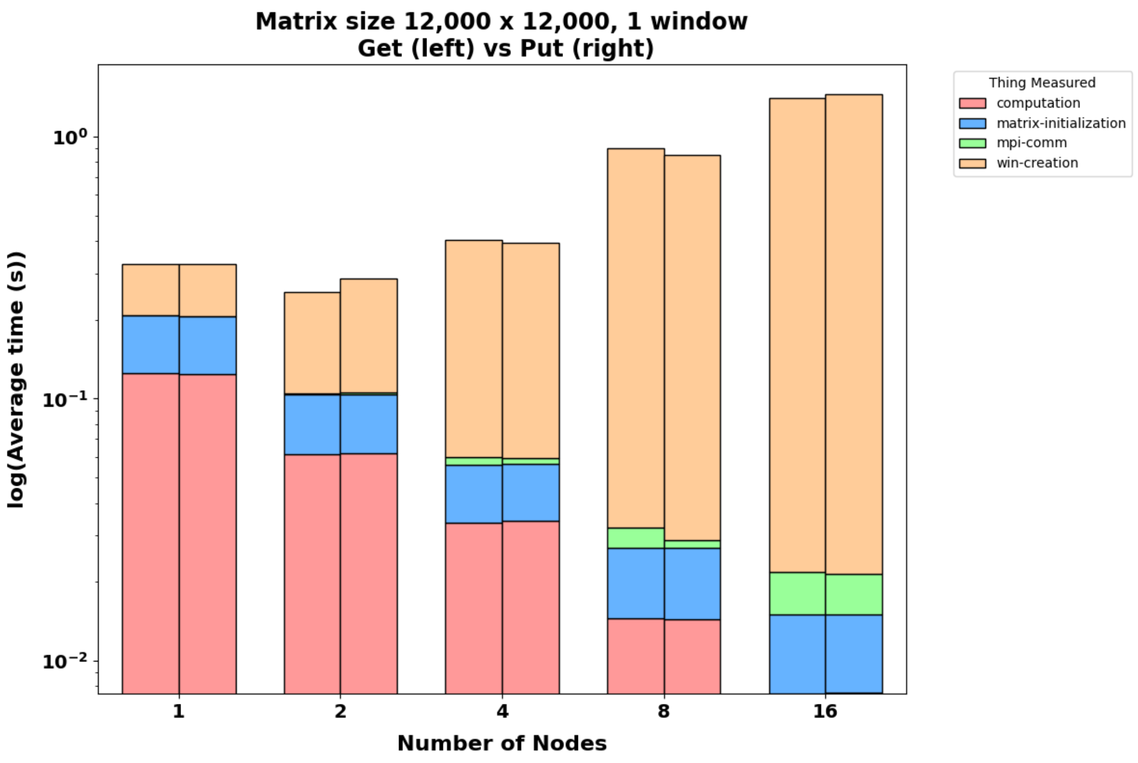

Figure 3: Matrix size: $12,000 \times 12,000$, 1 `MPI_Win`.

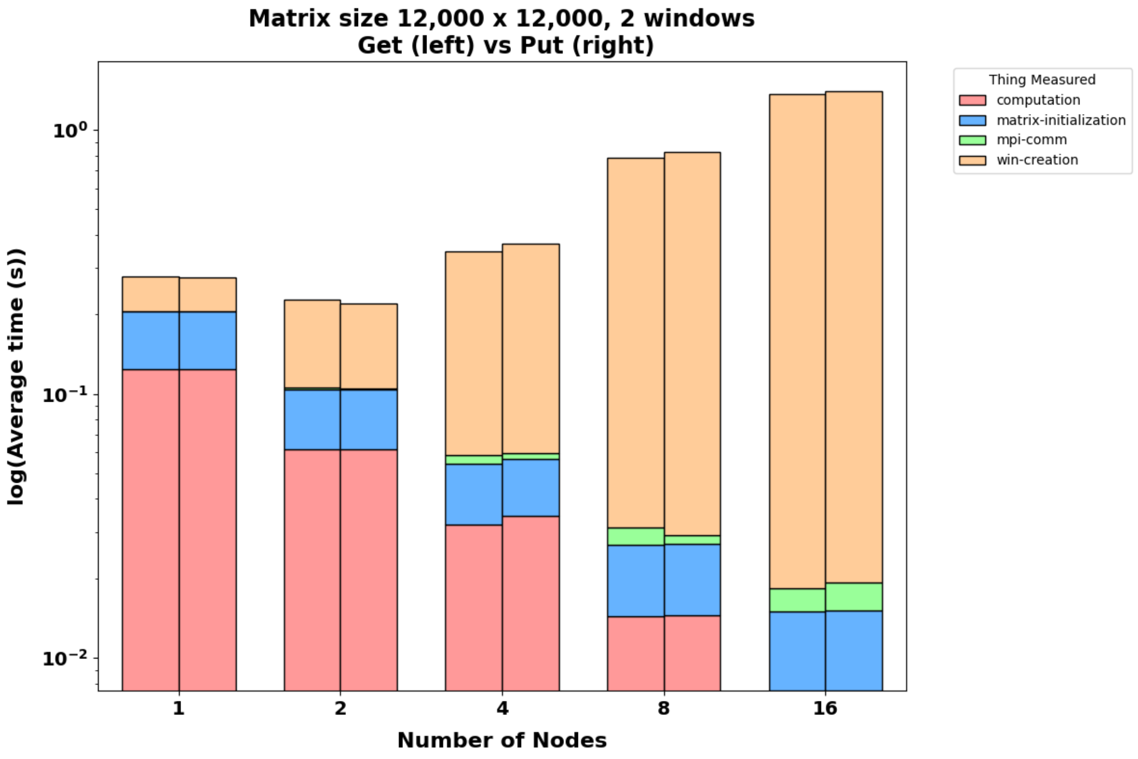

Figure 4: Matrix size: $12,000 \times 12,000$, 2 `MPI_Win`.

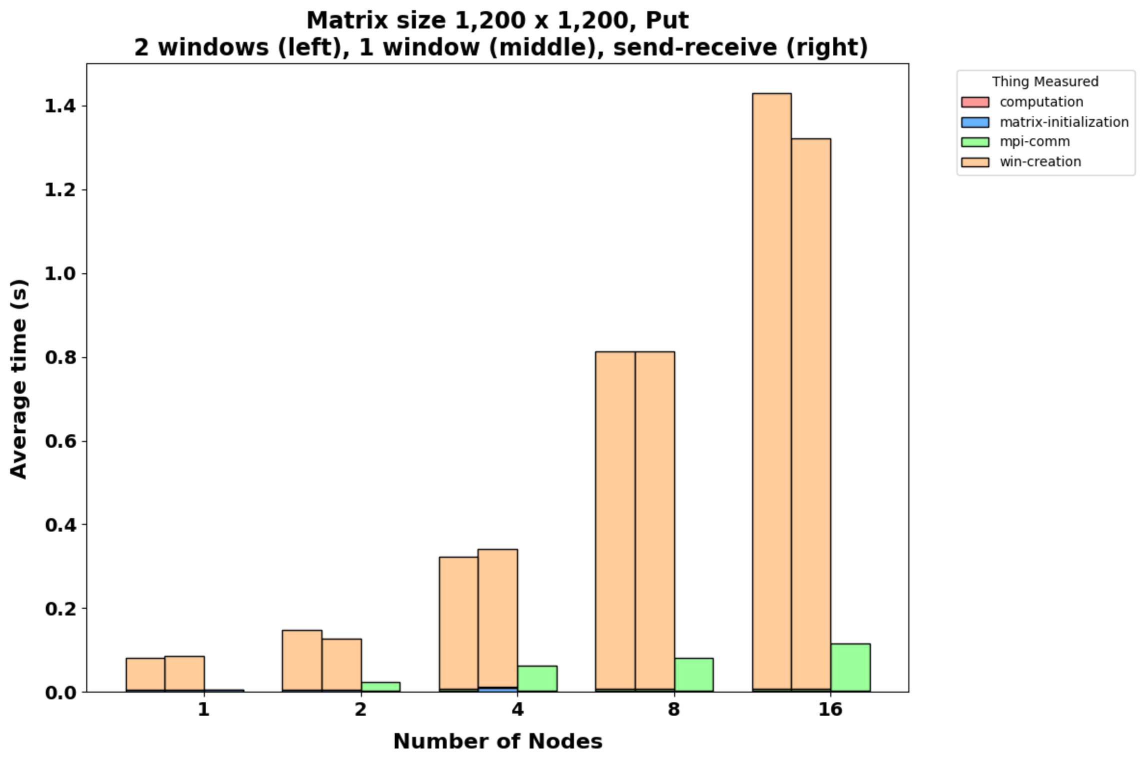

Figure 5: Matrix size: $1,200 \times 1,200$. The performance of the 1 window and 2 windows implementations is very similar.

2.2 1 window vs 2 windows

For the following plots, the `MPI_Put` function has been chosen as the communication method. Moreover, to have a comparison with the previous exercise, the results of the MPI point-to-point communication have been included, using `MPI_Isend` and `MPI_Irecv` functions.

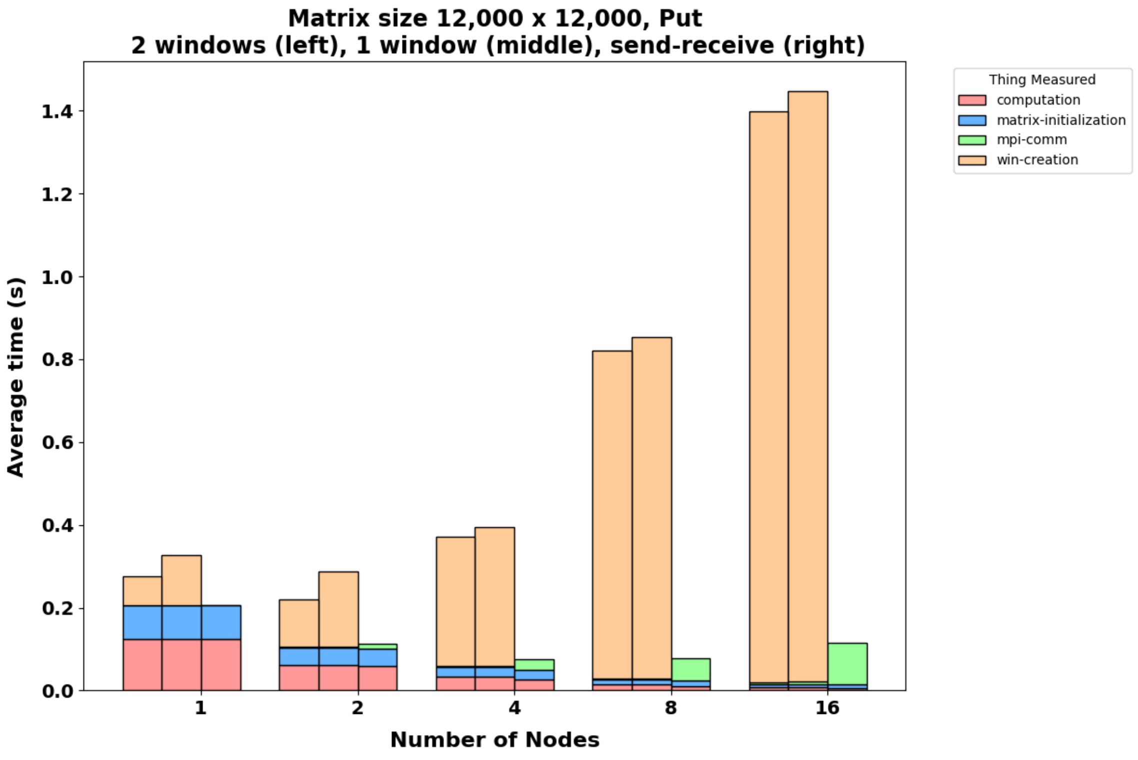

Figure 6: Matrix size: $12,000 \times 12,000$.