Parallel programming for HPC: exam project
Exercise 1: Matrix-matrix multiplication

Student: Isac Pasianotto

2024-05

0 Requirements

The goal of this exercise is to implement the matrix-matrix multiplication algorithm in a distributed setting using MPI. The program should be able to solve the problem $C = AB$ with $A,\ B,\ C \in \mathbb{R}^{N\times N}$.
Three different implementations are required:

- **Naive implementation:** The multiplication is done by a 3-nested loop.
- **Blas implementation:** The multiplication is done using the `dgemm` function from the BLAS (Basic Linear Algebra Subprograms) library.
- **Cuda implementation:** The multiplication is done using the CUDA version of the BLAS library.

Scaling test are required, in order to highlight the performance of the different implementations and to spot the bottlenecks.

1 Implementation

All the code are available on the repository of the project. The provided `Makefile` can be used to compile the code as follows:

- `make` will produce the executables which implements the naive implementation
- `make blas` will produce the executables which implements the BLAS implementation
- `make cuda` will produce the executables which implements the CUDA implementation

And the executables can be run on the Leonardo cluster. To completely exploit the potential of the code is recommended to run:

- On DCGP nodes, which are used for CPU computing, spawning one MPI process per node, and let OpenMP to use all the available cores in the node.
- On Booster nodes, which are used for GPU computing, spawning four MPI processes per node, in this way the program will use all the available GPUs in the node (every task will be pinned to a different GPU).

2 Results

The scaling tests leads to the following considerations:

- All the implementations are able to scale well with the number of nodes.
- As expected, the naive implementation is the slowest one, the BLAS implementation outperforms the naive one by two orders of magnitude, and the CUDA implementation outperforms the BLAS one by one order of magnitude.
- In the naive implementation, almost the totality of the time spend is in the computation of the C matrix. Matrix initialization and processes communication is completely negligible
- In the CUDA implementation, the major bottleneck, up to the tested 16 nodes, is the matrix initialization, but observing the trend of the stack plot is reasonable to think that increasing the number of nodes the bottleneck will become performing the MPI-AllgatherV operation, and moreover the time spent to compute the blocks of the result C will become more and more relevant since it seems to not scale well with the number of nodes.

In the following pages are shown some of the most relevant plots obtained during the scaling tests.

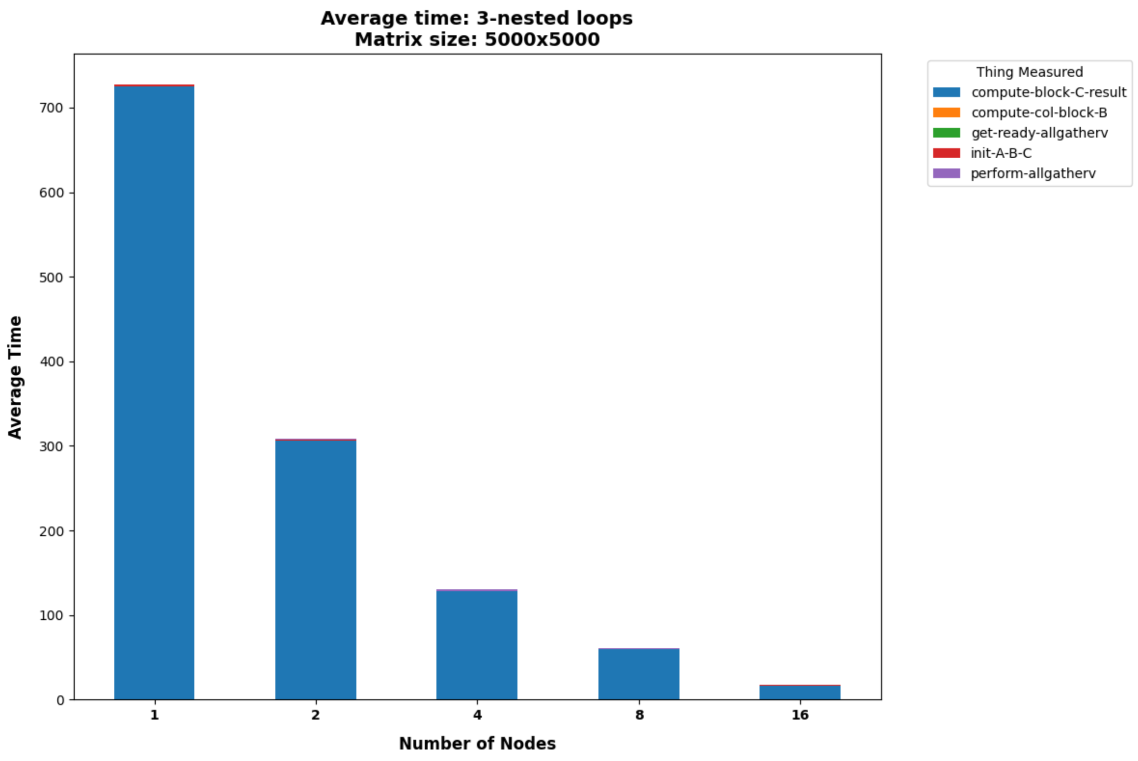

Figure 1: Scaling test of the naive implementation

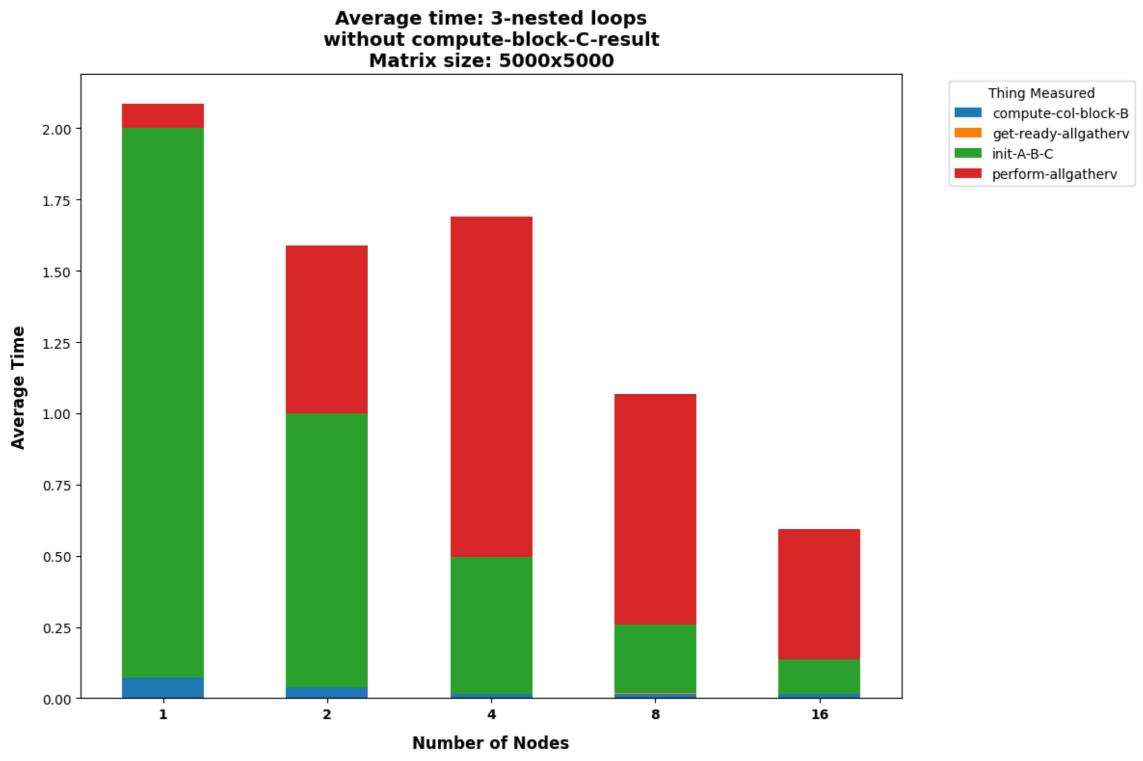

Figure 2: Scaling test of the naive implementation: zoom of the time scaling not considering the computation of C time

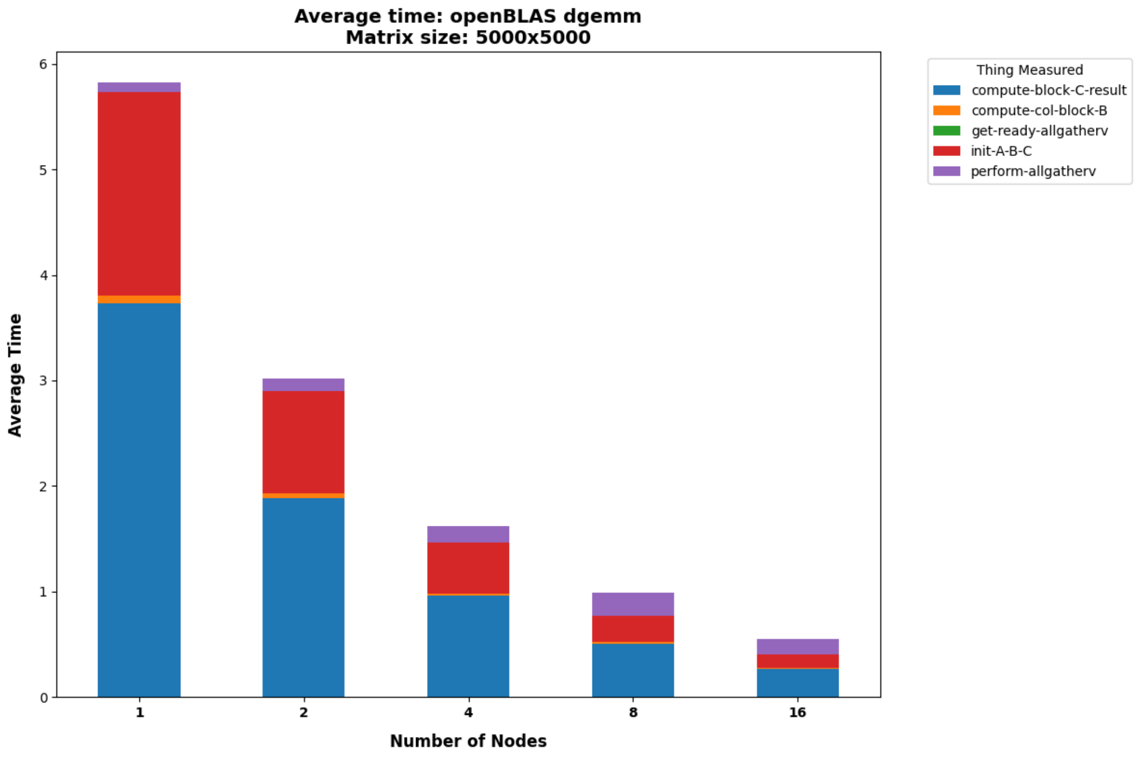

Figure 3: Scaling test of the BLAS implementation

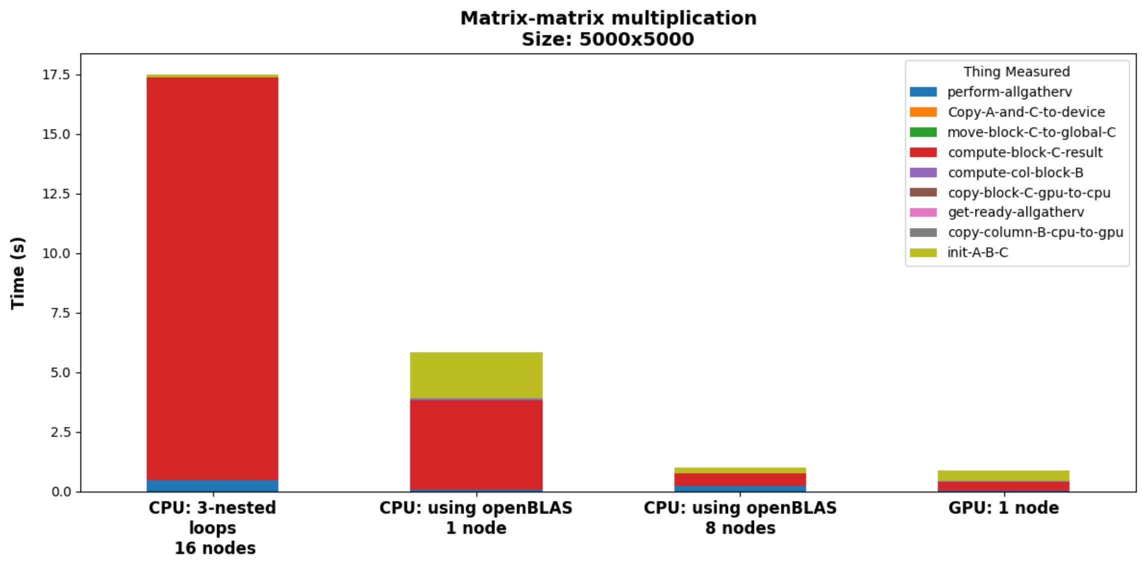

Figure 4: Comparison with the same size of matrix between the 3 implementations. This is why the scaling test of the CUDA implementation is done with a bigger matrix

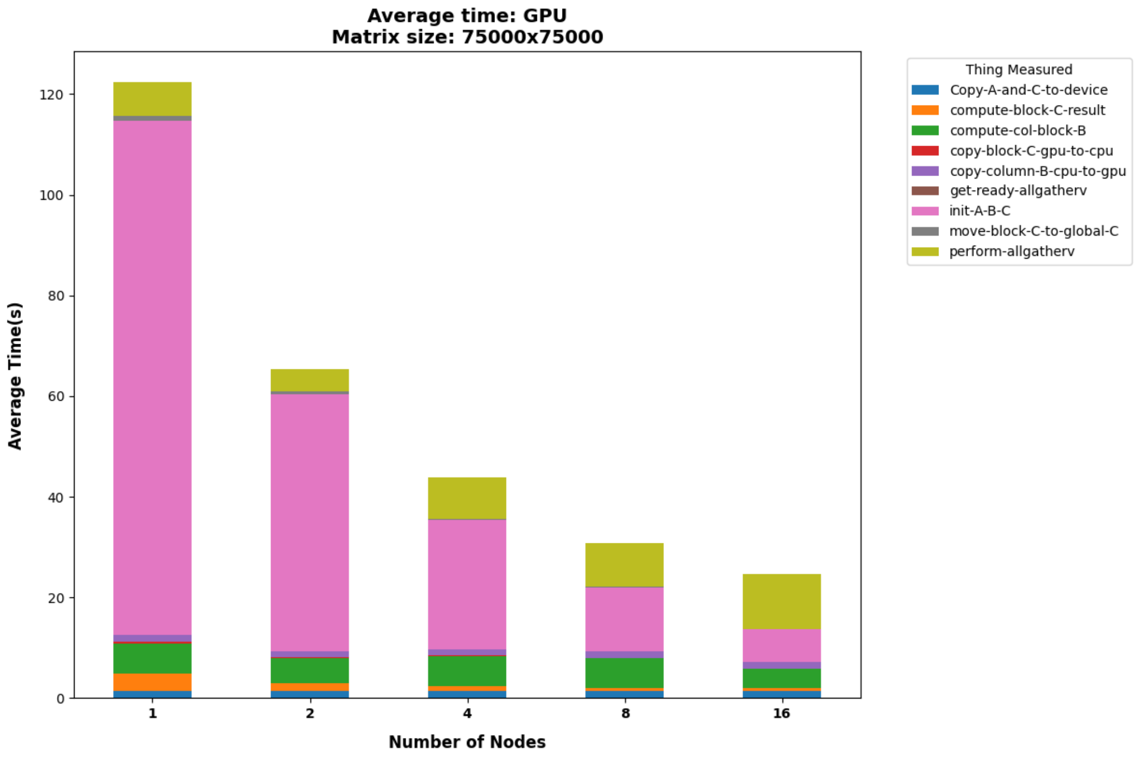

Figure 5: Scaling test of the CUDA implementation