# Advanced HPC - Final assignment

Tommaso Tarchi

October 3, 2024

University of Trieste

## Matrix-matrix multiplication

## General algorithm

1. $A$, $B$ and $C$ are allocated as blocks of $n_{rows}$ rows across processes (called $A_{loc}$, $B_{loc}$ and $C_{loc}$).
2. Each process initializes $A_{loc}$ and $B_{loc}$ randomly;
3. If using cuBLAS, $A_{loc}$ is moved to device.
4. Considering each $C_{loc}$ as divided into $n_{procs}$ adjacent blocks of $n_{rows} \times n_{cols}$ elements, the following points are iterated $n_{procs}$ times:
   - each process communicates to all other processes only $B_{loc}$'s data needed to compute the i-th block of $C_{loc}$;
   - each process gathers data coming from all other processes into a single matrix, called $B_{col}$;
   - if using cuBLAS, $B_{col}$ is moved to device;
   - each process computes the i-th block of $C_{loc}$.
5. If using cuBLAS, $C_{loc}$ computed on device is moved back to host.

## Implementation details

- Matrices distribution was handled by evenly distributing rows among processes. Remainder rows were assigned to first processes.
- For random initialization of $A$ and $B$, a unique seed was computed randomly by process 0 and broadcasted to all other processes, which then set different seeds for each one of their thread.
- OpenMP was used to speed up the initialization of $A$ and $B$.
- MPI communications were performed using `MPI_Allgatherv()`.
- As BLAS library we chose Intel MKL 2023.2.0.
- When using BLAS and cuBLAS, we accumulated the computed entries of $C_{loc}$ directly on the memory allocated for $C_{loc}$, by combining the pointer to matrix and stride arguments.
- When using cuBLAS, we avoided transpositions by computing $B_{col}^T \times A_{loc}^T$.

## Profiling

Times measured:

1. **Matrices initialization**, including:
   - random initialization of matrices $A_{loc}$ and $B_{loc}$.
2. **Communication**, including:
   - creation of all $B_{block}$'s;
   - all executions of `MPI_Allgatherv()`.
3. **Computation**, including:
   - all executions of `dgemm()`.
4. **Host-device data movements** (only for cuBLAS code), including:
   - initial transfer of $A_{loc}$ from host to device;
   - transfers of all $B_{col}$'s from host to device;
   - final transfer of $C_{loc}$ from device to host.

**Notice**: measured times were averaged over all processes.

## Jacobi method for Laplace equation

## List of implementations

1. Simple MPI.
2. **OpenACC with CUDA-unaware MPI**.
3. **OpenACC with CUDA-aware MPI**.
4. **MPI with remote memory access**.

**Notice** that all programs were farther parallelized using openMP.

## OpenACC - algorithm

1. Each process allocates its own portion of the grid. Also, an identical auxiliary matrix to store updated cells state is allocated.
2. Each process initializes its own cells.
3. Both main and auxiliary grids are moved to device.
4. For the required number of iterations, each process repeats the following:
    - communicates to the bordering processes the corresponding neighbor rows from host to host (CUDA-unaware) or device to device (CUDA-aware);
    - (only for CUDA-unaware) received boundaries are updated on device;
    - evolves its own cells on device on the auxiliary grid and then copies the updated system state to the main one;
    - (only for CUDA-unaware) updated border cells are copied back to host.
5. (Only for CUDA-unaware) the final grid is copied back to host.

## OpenACC - implementation details

## OpenACC - profiling

Times measured:

1. **System initialization**, including:
   - allocation and initialization of main and auxiliary matrices.
2. **One-time host-device data movements**, including:
   - initial transfer of matrices from host to device;
   - final transfer of main matrix from device back to host.
3. **Communication**, including:
   - all executions of `MPI_Sendrecv()`.
4. **Computation**, including:
   - all system updates on auxiliary matrix;
   - all copies from auxiliary to main matrix.
5. **In-loop host-device data movements**, including:
   - all incoming borders updates after communications (host to device);
   - all outgoing borders updates after computation (device to host).

## OpenACC - profiling

**Notice**: initialization and one-time host-device transfer times were averaged over all processes; while other times were averaged over iterations first and processes then.

## OpenACC - results

## MPI RMA - algorithm

1. Each process allocates its own portion of the grid and two additional arrays to store the upper and lower boundaries separately. Also, an auxiliary grid to store updated cells state is allocated.
2. Each process initializes its own cells.
3. For the required number of iterations, each process repeats the following:
    - opens two windows on its border arrays to allow neighbor processes to write directly;
    - writes directly to the neighbor processes' border arrays the corresponding neighbor rows (i.e. first and last), while evolves internal cells on the auxiliary grid;
    - closes the opened windows;
    - evolves cells on the first and last rows on the auxiliary grid and then swaps pointers to update main grid.

## MPI RMA - implementation details

## MPI RMA - profiling

Times measured:

1. **System initialization**, including:
    - allocation of main and auxiliary matrices and window allocation for boundaries;
    - main and auxiliary matrices and extreme boundaries initialization.
2. **Communication**, including:
    - all MPI direct put on neighbor processes's windows execution (i.e. `MPI_Win_start()`, `MPI_Put()` and `MPI_Win_complete()`);
    - all waits for MPI direct put on process's own windows to complete (i.e. `MPI_Win_wait()`).
3. **Computation**, including:
    - all system updates on auxiliary matrix (both internal and external rows);
    - all pointer swaps for main matrix update.

**Notice**: initialization time was averaged over all processes; while other times were averaged over iterations first and processes then.

## MPI RMA - results