Parallel programming for HPC

Final project

Isac Pasianotto, 2024-06-08

Matrix-Matrix multiplication

Requirements

- Implement an algorithm to solve $C = AB,\ A, B, C \in \mathbb{R}^{N \times N}$
- Three different implementations are required:
  - *Naive*: 3-nested loop.
  - *Blas:* $\mathrm{cblas\_dgemm}$ function
  - *Cuda:* $\mathrm{cublasDgemm}$ function
- *Multi-node* computation with MPI
- *Scaling test*

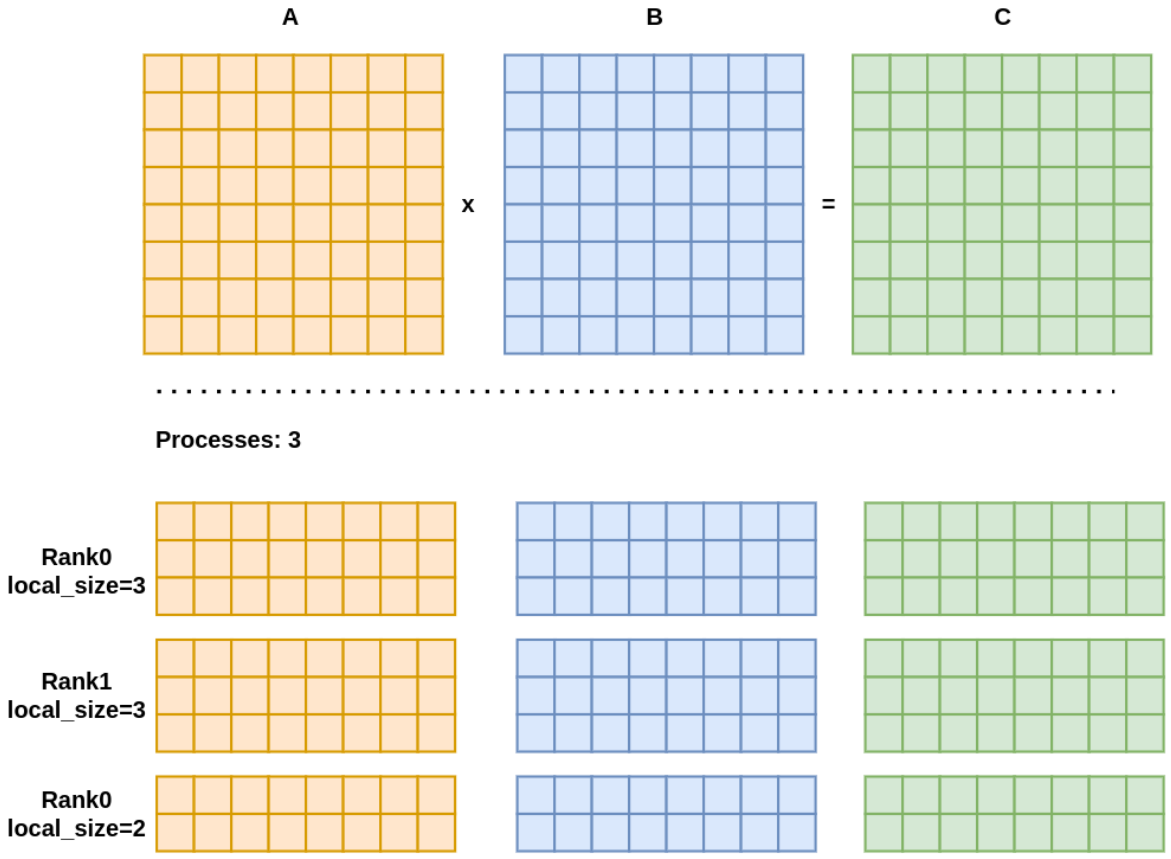
Implementation

```
#include <...>

int main(int argc, char** argv)
{
  initialize_data_distributely(A, B, C);
  for (int i = 0; i < n_processes; i++)
  {
    create_block(B, B_block);
    MPI_Allagatherv(B_block, ... );
    mat_mult(A, B_block, C_block);
    copy_cblock_back(C_block, C);
  }
  // print_matrix(C);
  return 0;
}
```

Distributed initialization

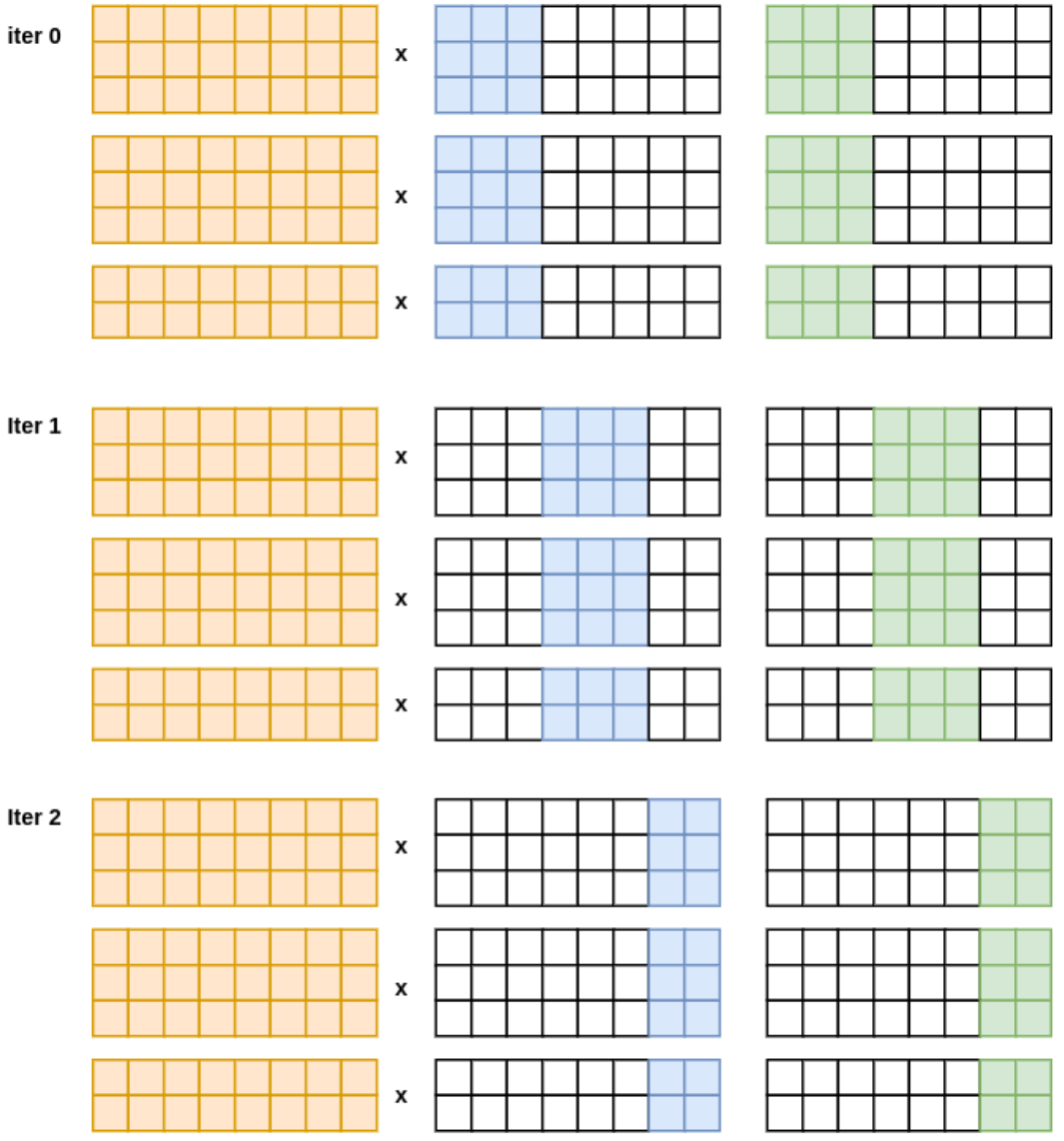
- Matrices divided by rows
- (Almost) homogeneous distribution of the workload between processes



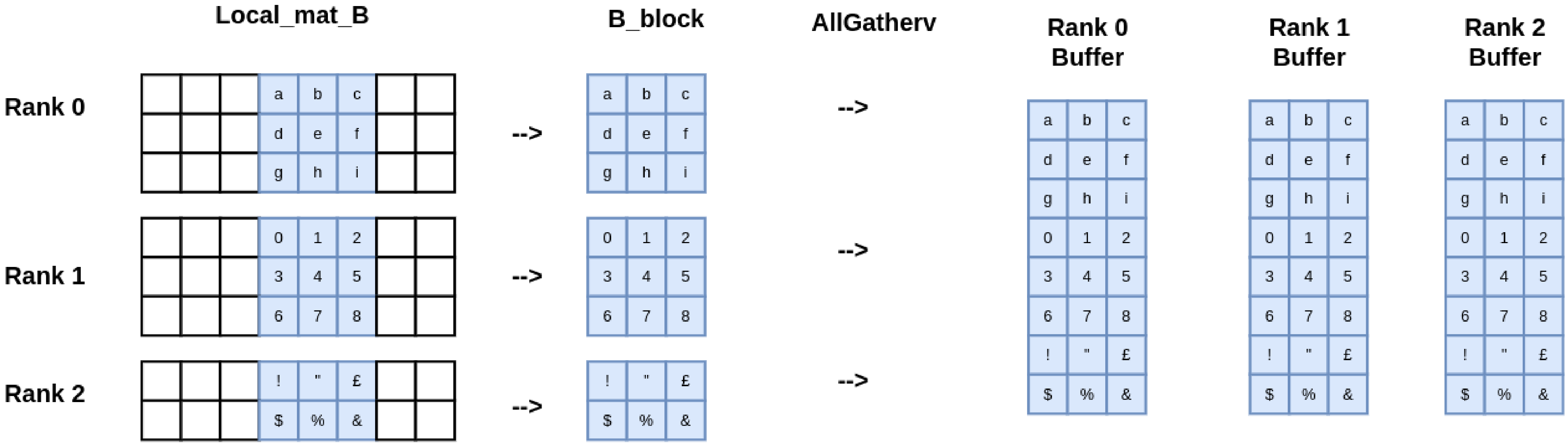
$$\text{local\_size} = (\text{rank} < \text{N} \,\%\, \text{size}) \;?\; \text{N} \;/\; \text{size} + 1 \;:\; \text{N} \;/\; \text{size};$$

Loop over the # of processes

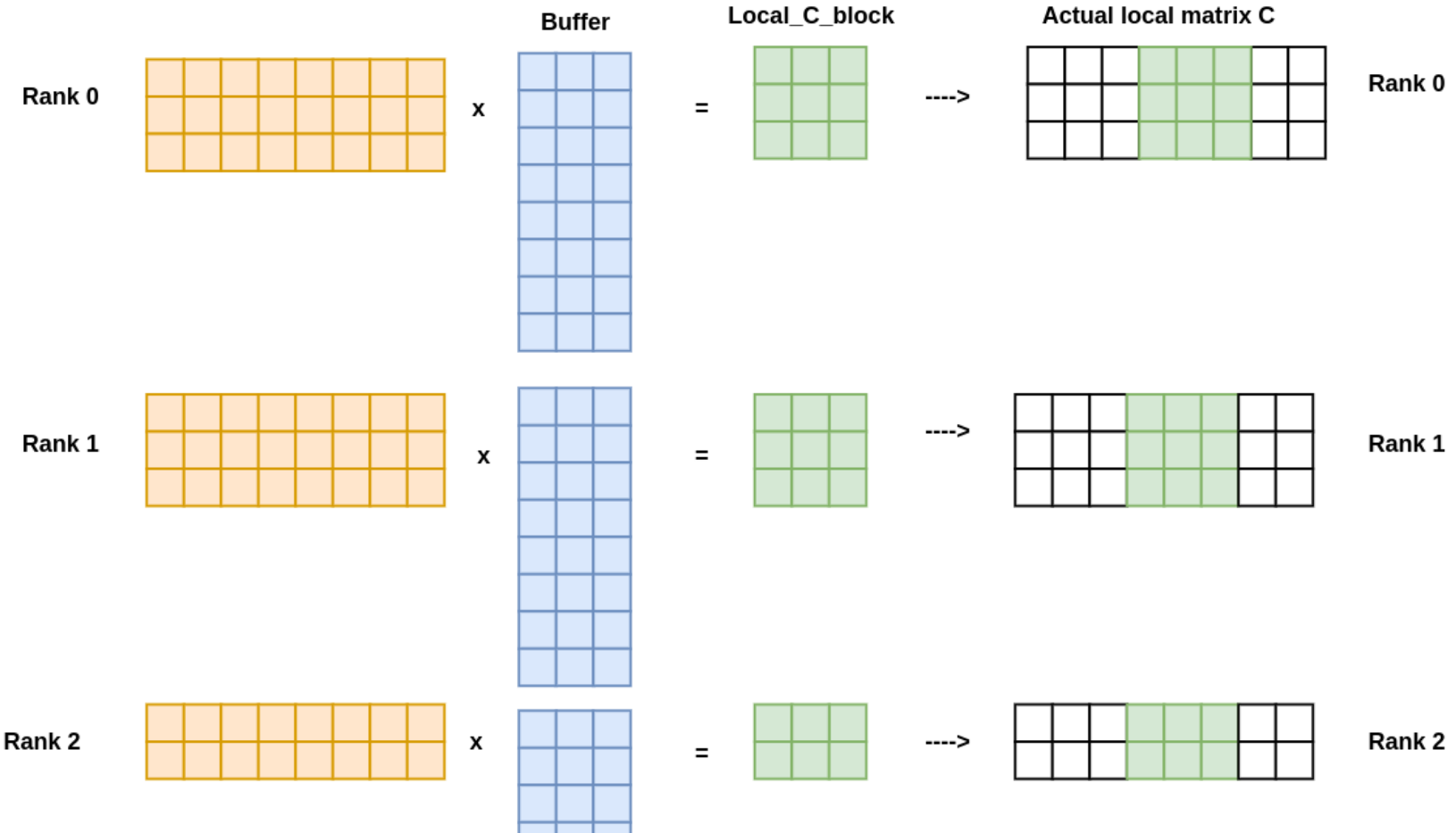
- Create $\mathrm{B\_block}$
- Communicate it to other processes
- Compute $\mathrm{C\_block}$



Gather the column block



Result: C_block



Results: a initial disclaimer

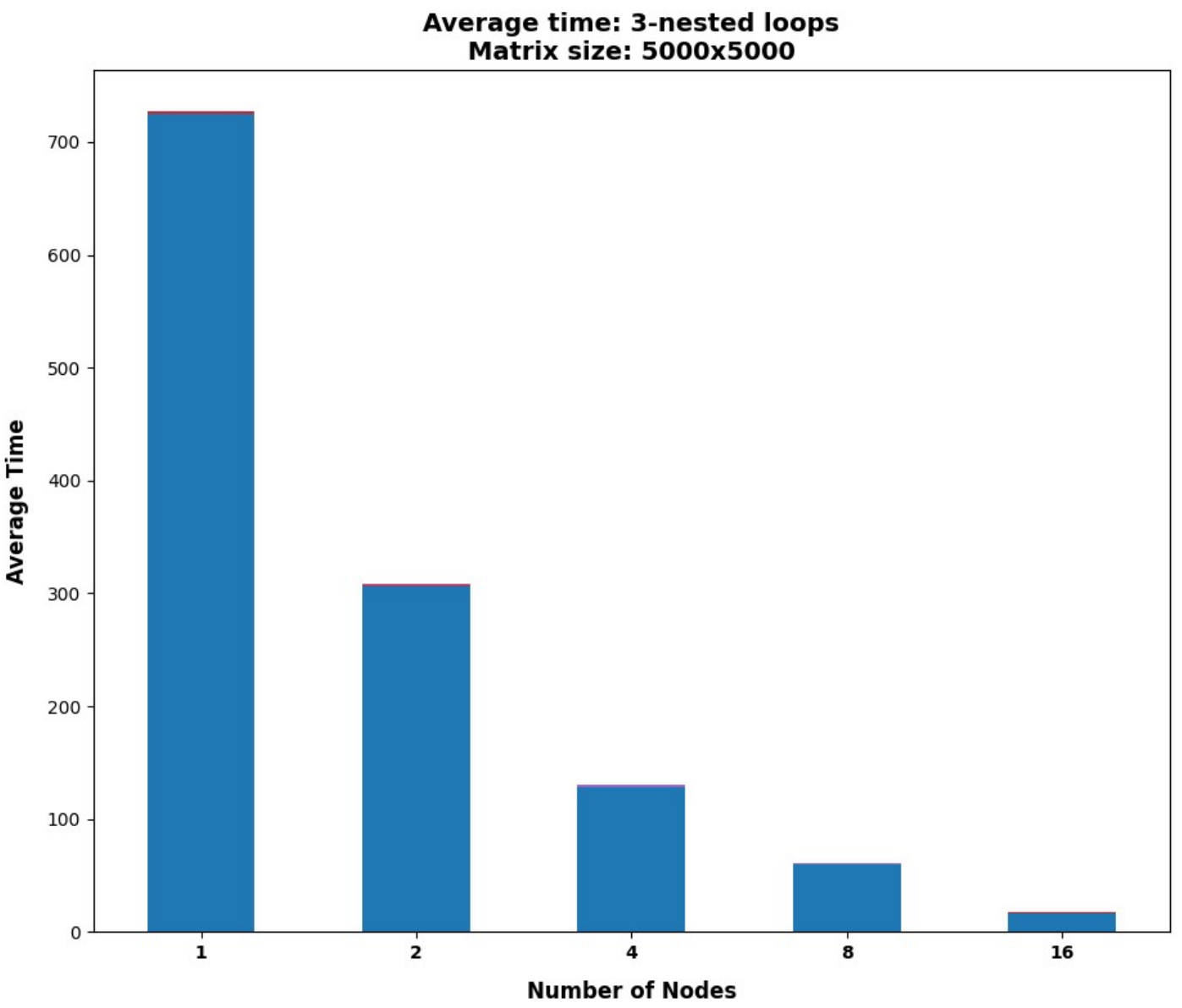
mpirun -np $nproc ./main.x

	Time (s)
Initialize A, B, C	1.925907
Compute B_block	0.071962
Perform MPI_Allgatherv	0.085323
Compute C_block	725.593634

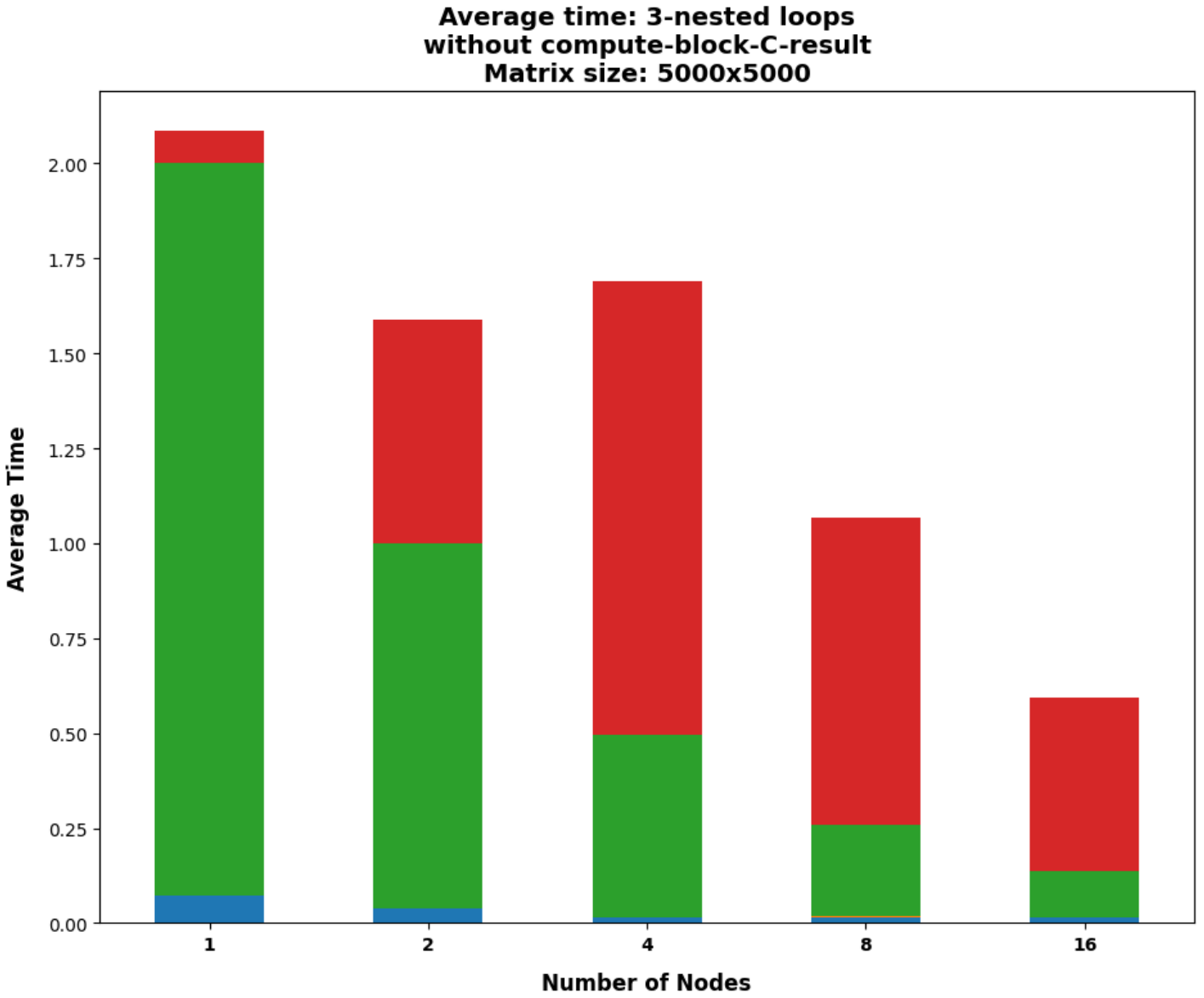
srun -N $nproc ./main.x

	Time (s)
Initialize A, B, C	36.501742
Compute B_block	0.016431
Perform MPI_Allgatherv	0.088304
Compute C_block	7.639606

Results: 3-nested loop

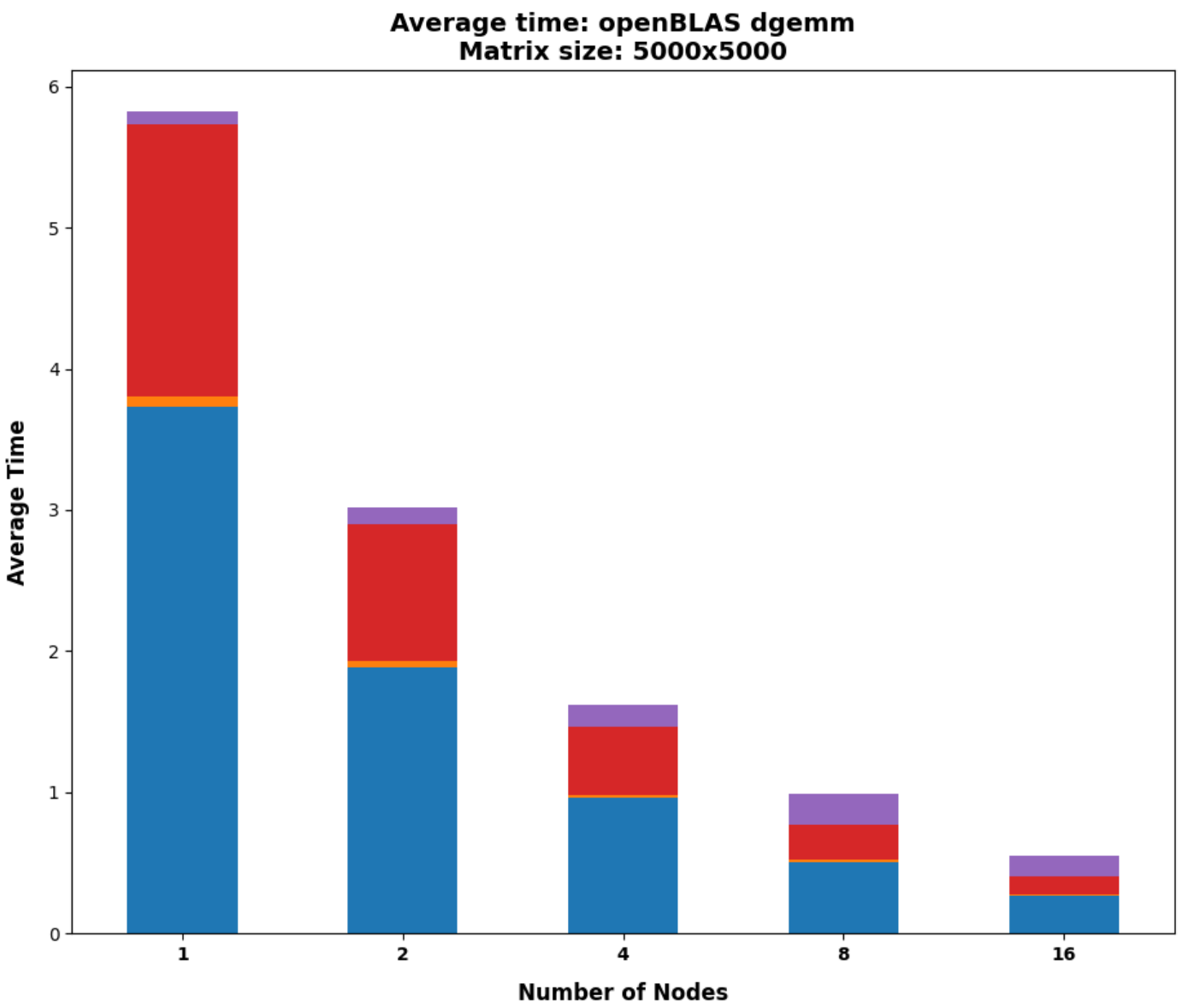


Results: 3-nested loop (Cont'd)

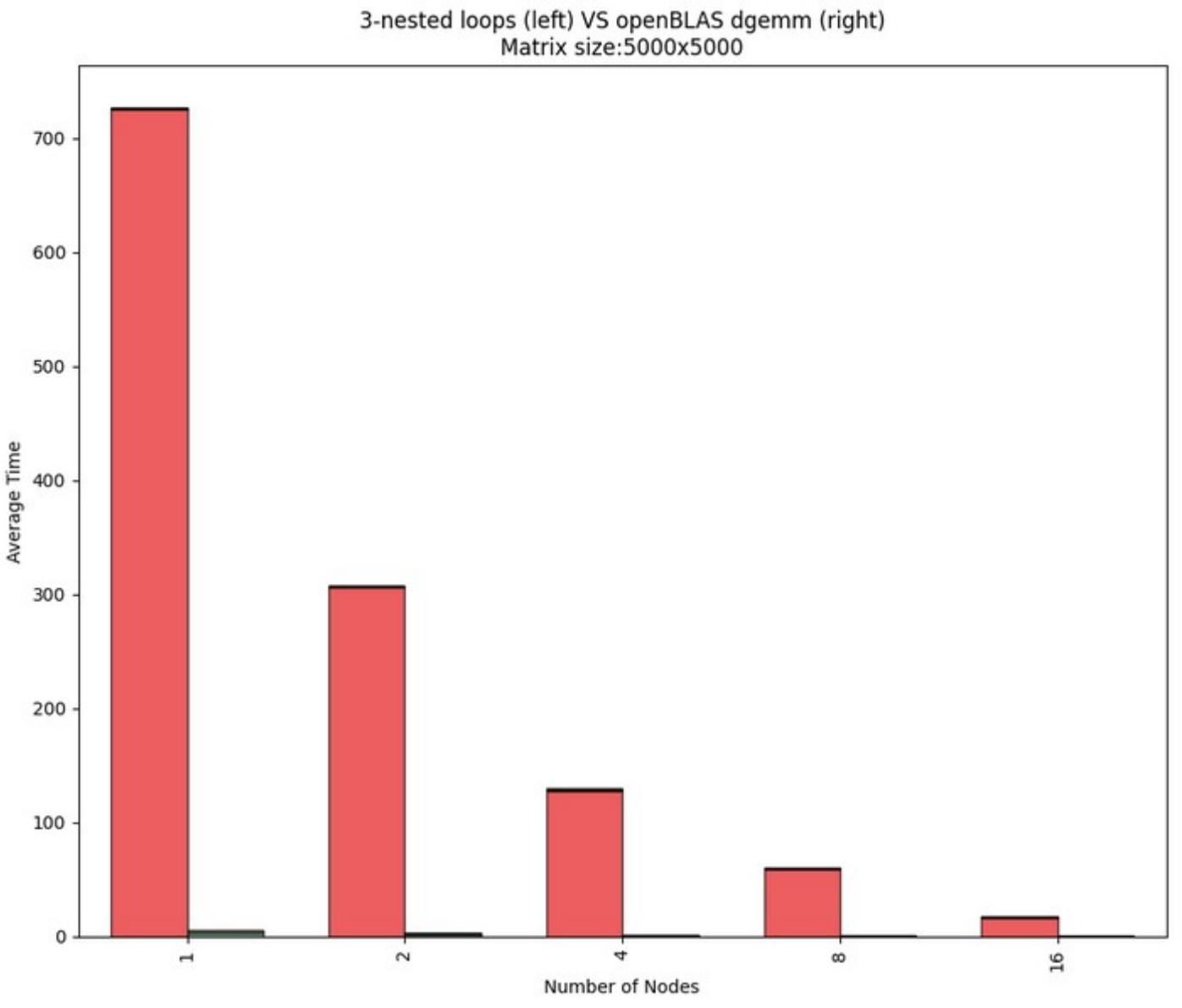


Results: BLAS

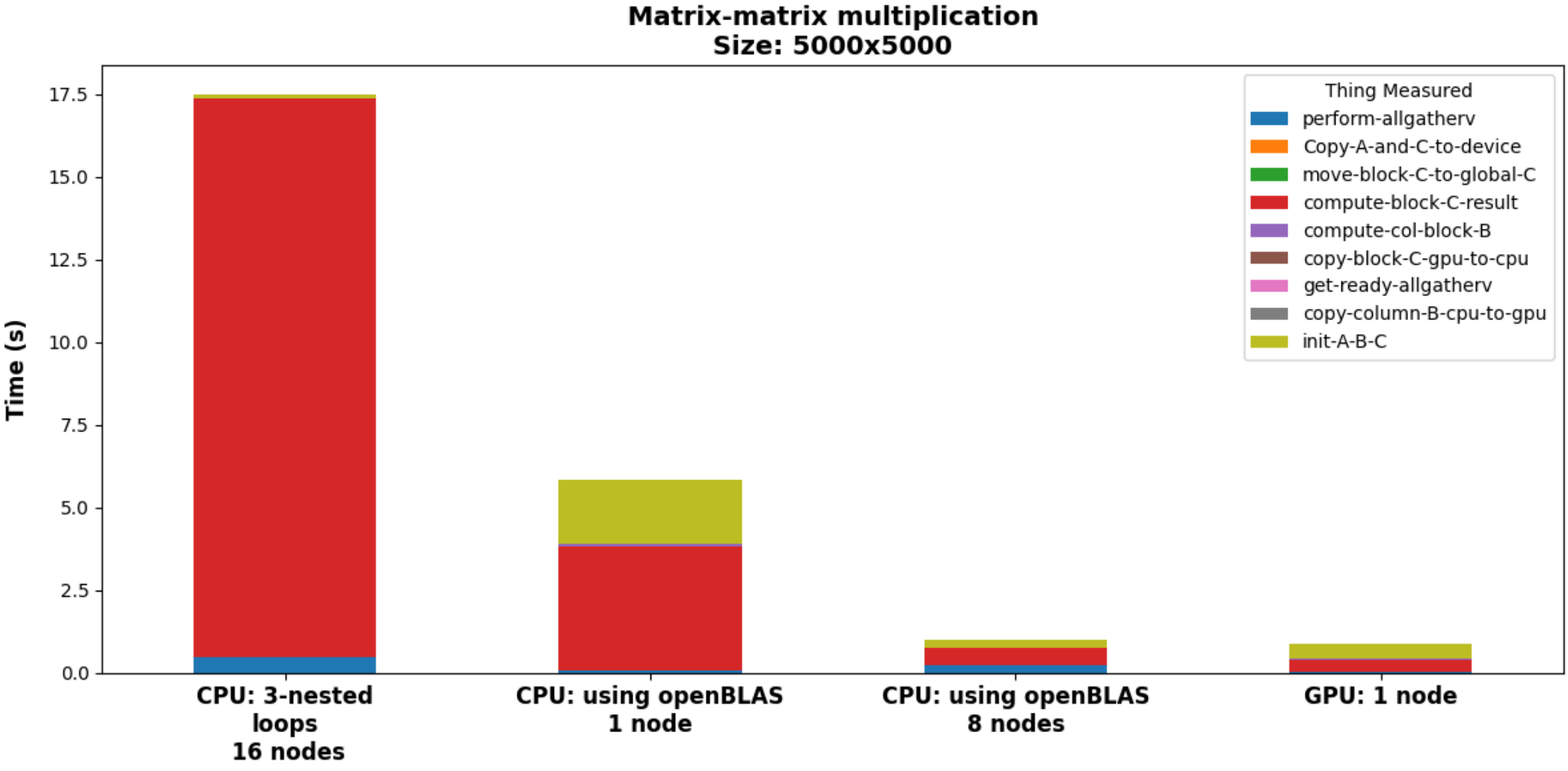
- In theory it should not scale so good



Results: CPU

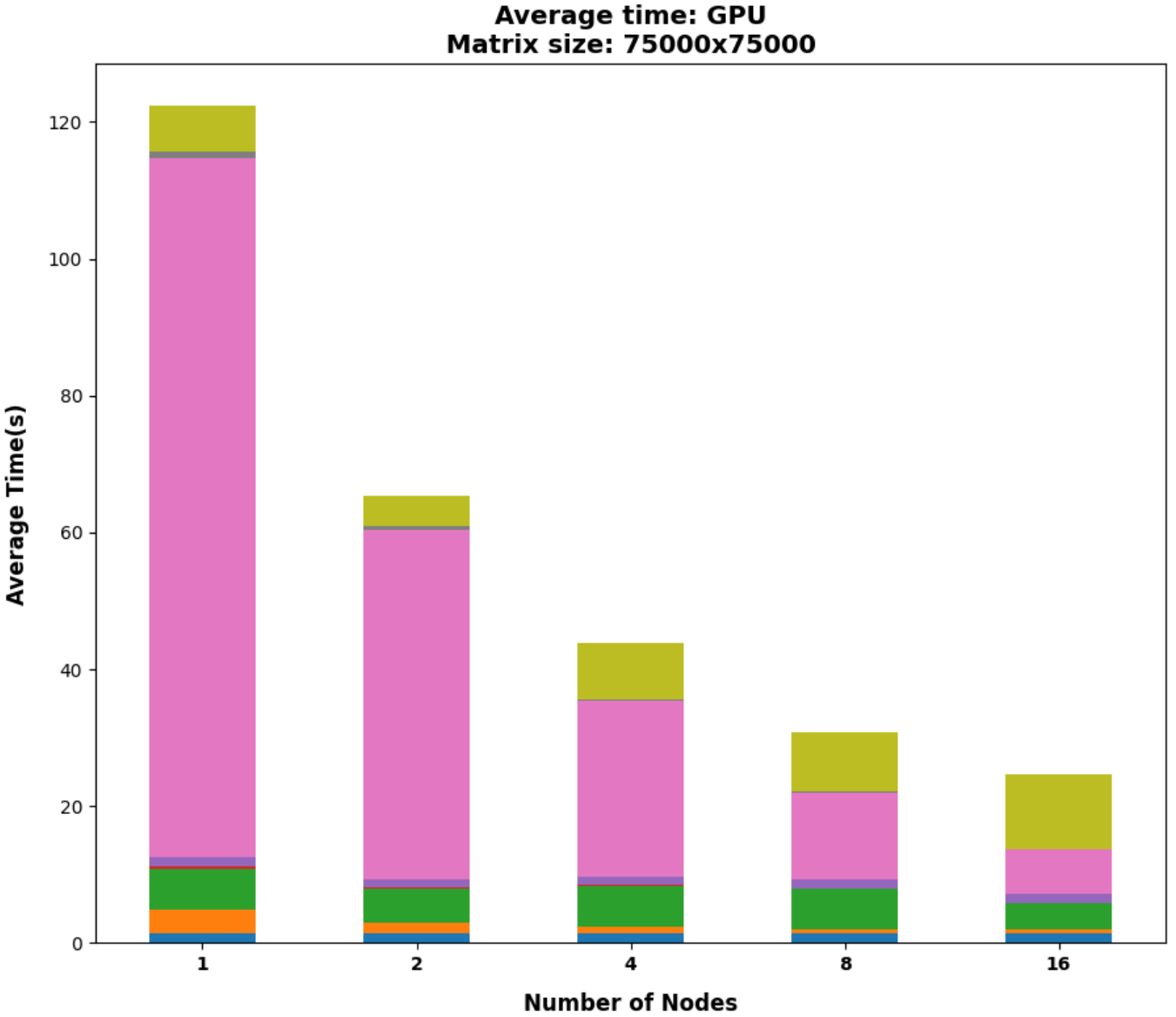

3-nested loops (left) VS openBLAS dgemm (right)
Matrix size:5000x5000
Average Time
0
100
200
300
400
500
600
700
1
2
4
8
16
Number of Nodes

Results: cuda



Results: cuda (Cont'd)

- Increased the matrix size to 75,000



Jacobi

Requirements

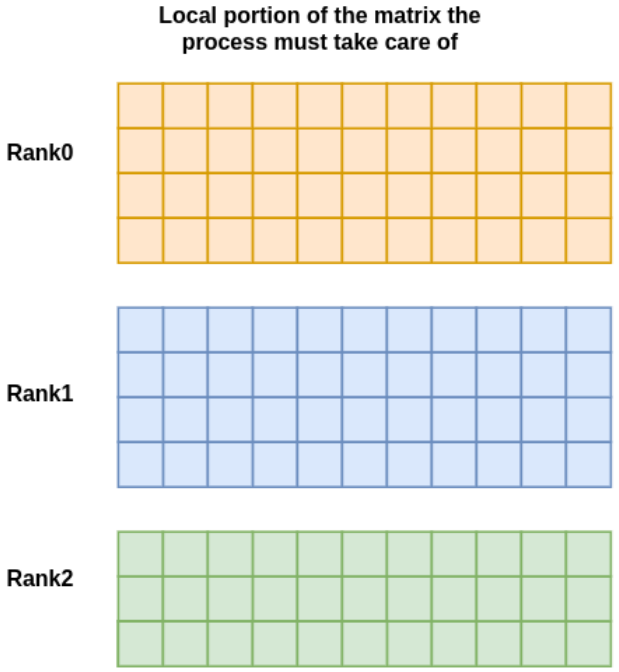
- Take an *already existing* code $\mathrm{jacobi.c}$
- *Make it parallel* using MPI
- *Port the code on GPUs* using $\mathrm{openACC}$
- Reduce as much as possible $CPU \leftrightarrow GPU$ data transfer
- *Scaling test* :
  - 1,200 x 1,200 -- 10 iteration
  - 12,000 x 12,000 -- 10 iterations

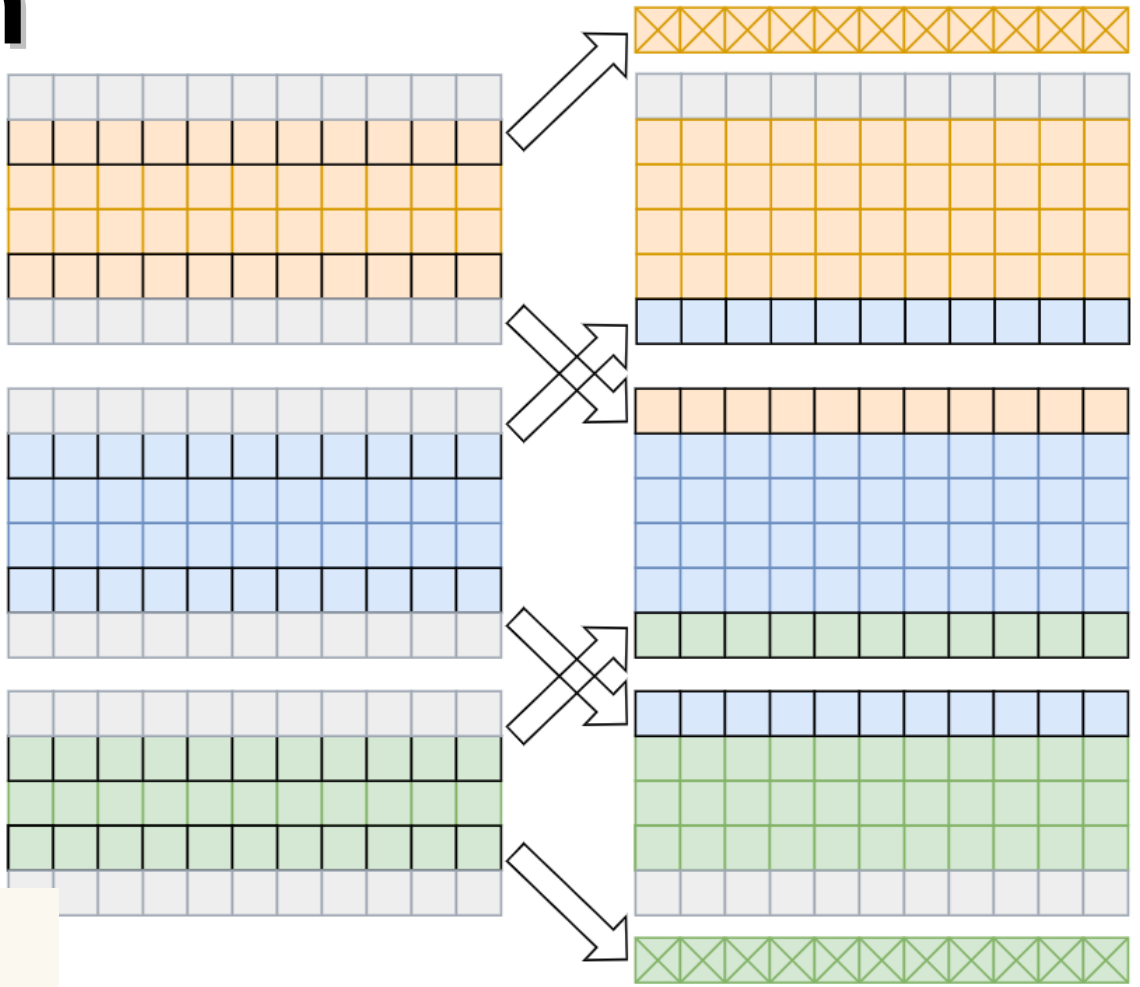
Implementation

```
#include <...>

int main(int argc, char** argv)
{
  initialize_local_matrix(local_matrix);                    //cpu
  move_local_mat_to_device(local_matrix, mat_dev);          //cpu->gpu
  for (int i = 0; i < n_iter; i++)
  {
    Communicate_with_neighbours(mat_dev, rank, size);       //gpu
    evolve(local_matrix);                                   //gpu
  }
  bring_back_local_mat_to_host(mat_dev, local_matrix);      //gpu->cpu
  save_resluts(local_matrix);                               //cpu
  return 0;
}
```

Communication





```
//send up,recv bottom
int send_to = (rank - 1) >= 0 ? rank - 1 : MPI_PROC_NULL;
int recv_from = (rank + 1) < size ? rank + 1 : MPI_PROC_NULL;
```

openACC

```
//To copy the two needed matrices from host to device and vice-versa:
#pragma acc enter data copyin(matrix[:(dimension+2)*(local_size+2)],matrix_new[:(dimension+2)*(local_size+2)])
{
  for (it = 0; it < iterations; ++it)
  {
    /*
     *    Actual computation ...
     */
  }
}
#pragma acc exit data copyout(matrix[:(dimension+2)*(local_size+2)],matrix_new[:(dimension+2)*(local_size+2)])
```

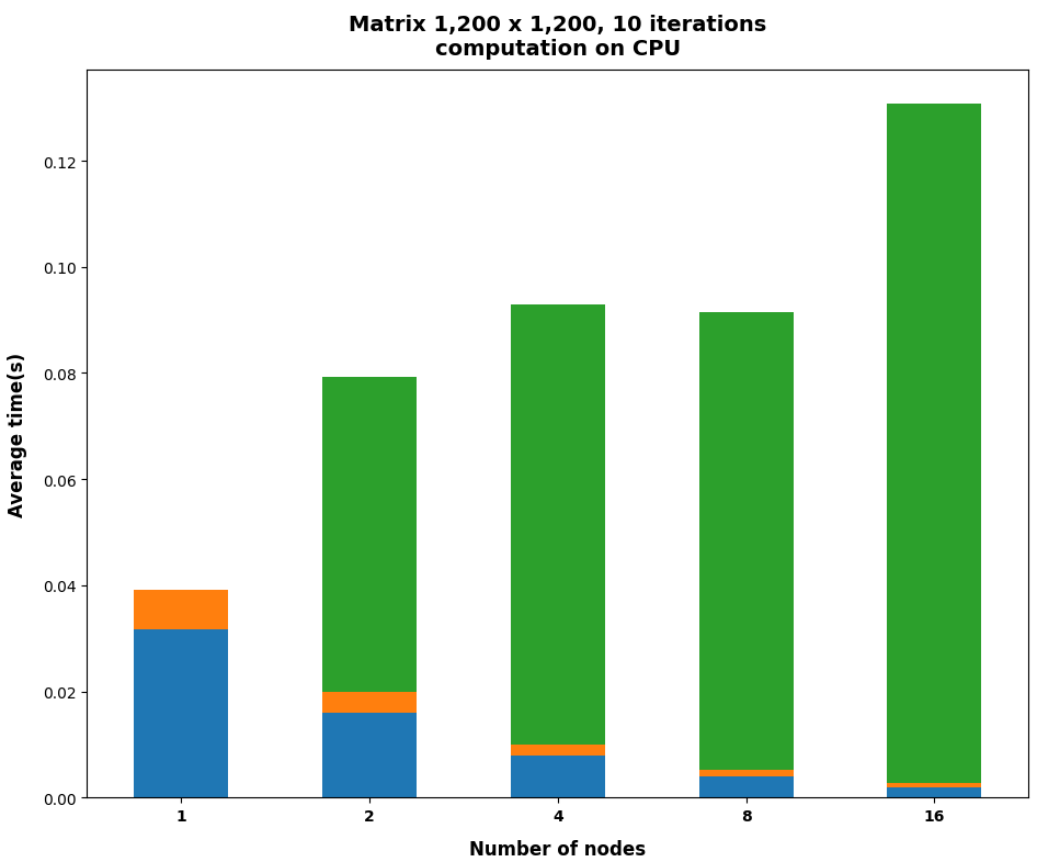
```
//gpu-gpu communication with no host involvement
#pragma acc host_data use_device(matrix)
{
  MPI_Sendrecv( matrix + (dimension + 2), dimension + 2, MPI_DOUBLE,
                send_to, 0, matrix + (dimension+2) * (local_size + 1),
                dimension+2, MPI_DOUBLE, recv_from, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  // MPI_Sendrecv( ... );
}
```

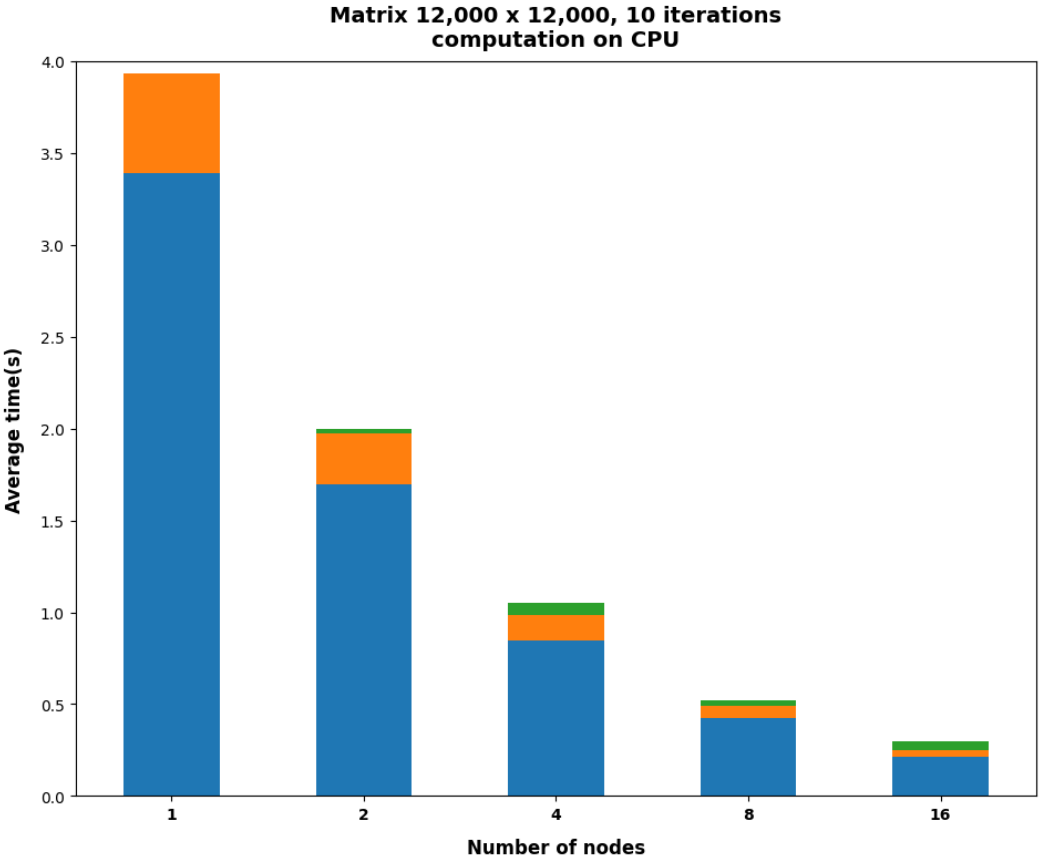
OpenACC (Cont'd)

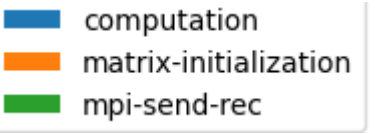
```
//The actual computation is done on the device

#pragma acc  data present(matrix[:(dimension+2)*(local_size+2)], matrix_new[:(dimension+2)*(local_size+2)])
{
#pragma acc parallel loop collapse(2)
  for (int i = 1; i <= local_size; ++i)
      for (int j = 1; j <= dimension; ++j)
        compute_new_value(matrix, matrix_new, i, j);
}
//swap the pointers on the device
#pragma acc serial present(matrix[:(dimension+2)*(local_size+2)],matrix_new[:(dimension+2)*(local_size+2)])
{
  double* tmp_matrix = matrix;
  matrix = matrix_new;
  matrix_new = tmp_matrix;
}
// swap pointer on the host, needed to preserve data consistency
tmp_matrix = matrix;
matrix = matrix_new;
matrix_new = tmp_matrix;
```

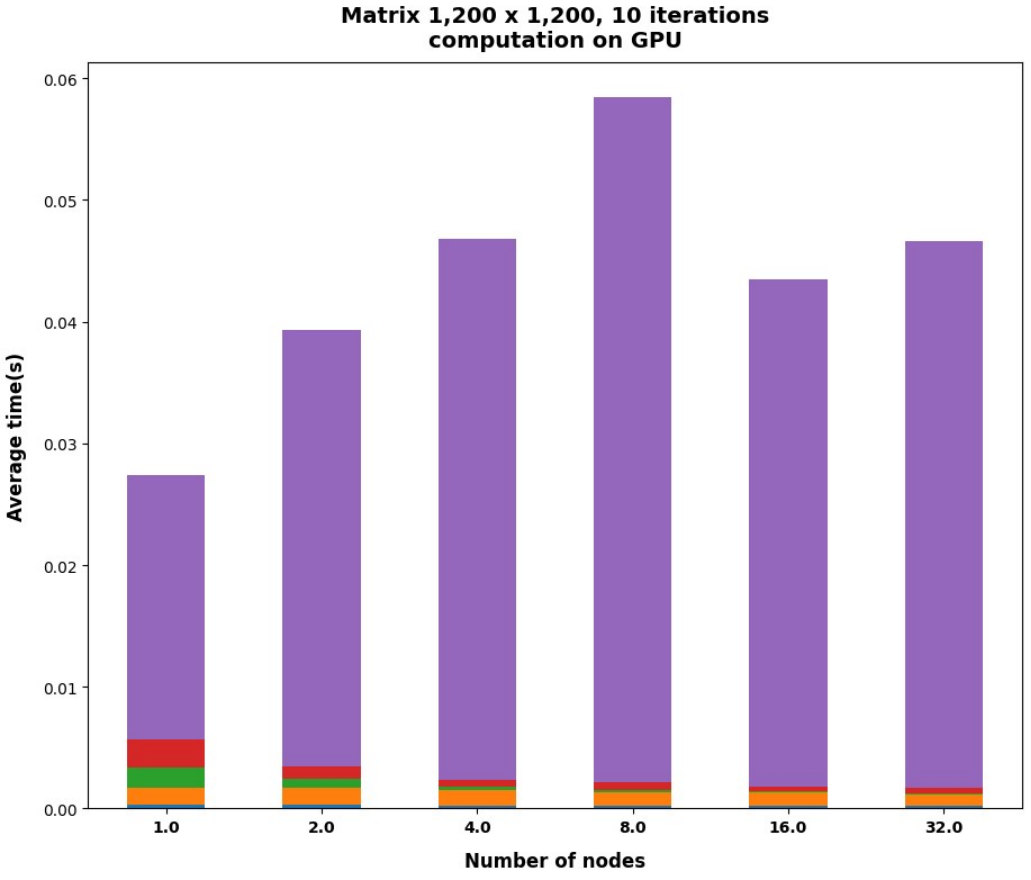
Results: CPU

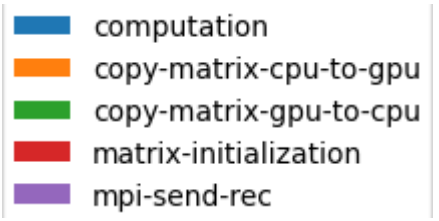

Matrix 1,200 x 1,200, 10 iterations
computation on CPU
Average time(s)
0.00
0.02
0.04
0.06
0.08
0.10
0.12
1
2
4
8
16
Number of nodes

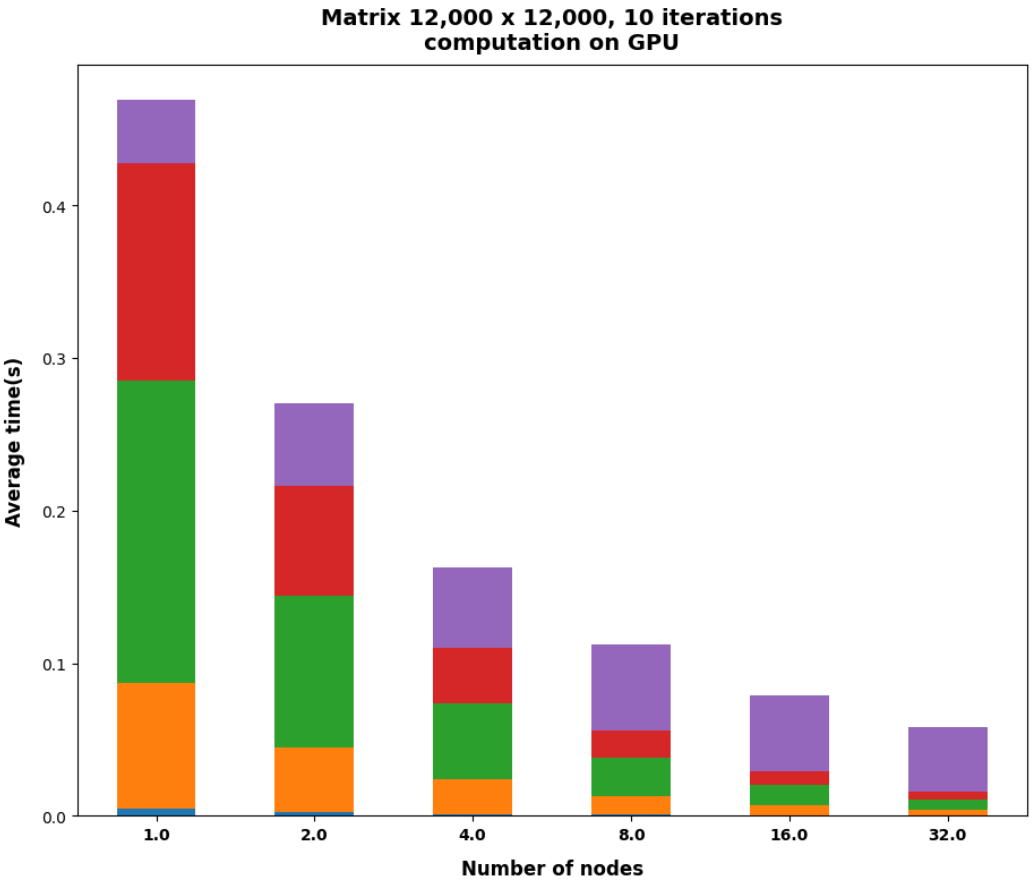

Matrix 12,000 x 12,000, 10 iterations
computation on CPU
Average time(s)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
1
2
4
8
16
Number of nodes


computation
matrix-initialization
mpi-send-rec

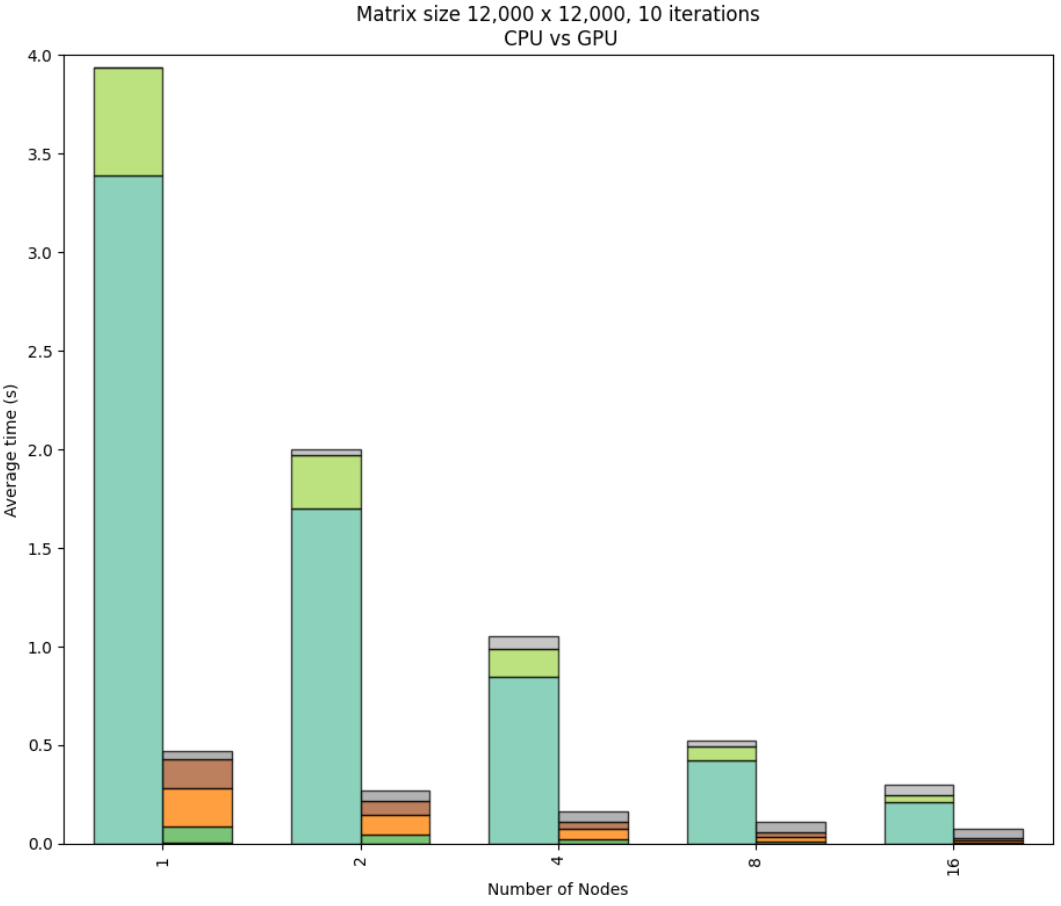
Results: GPU

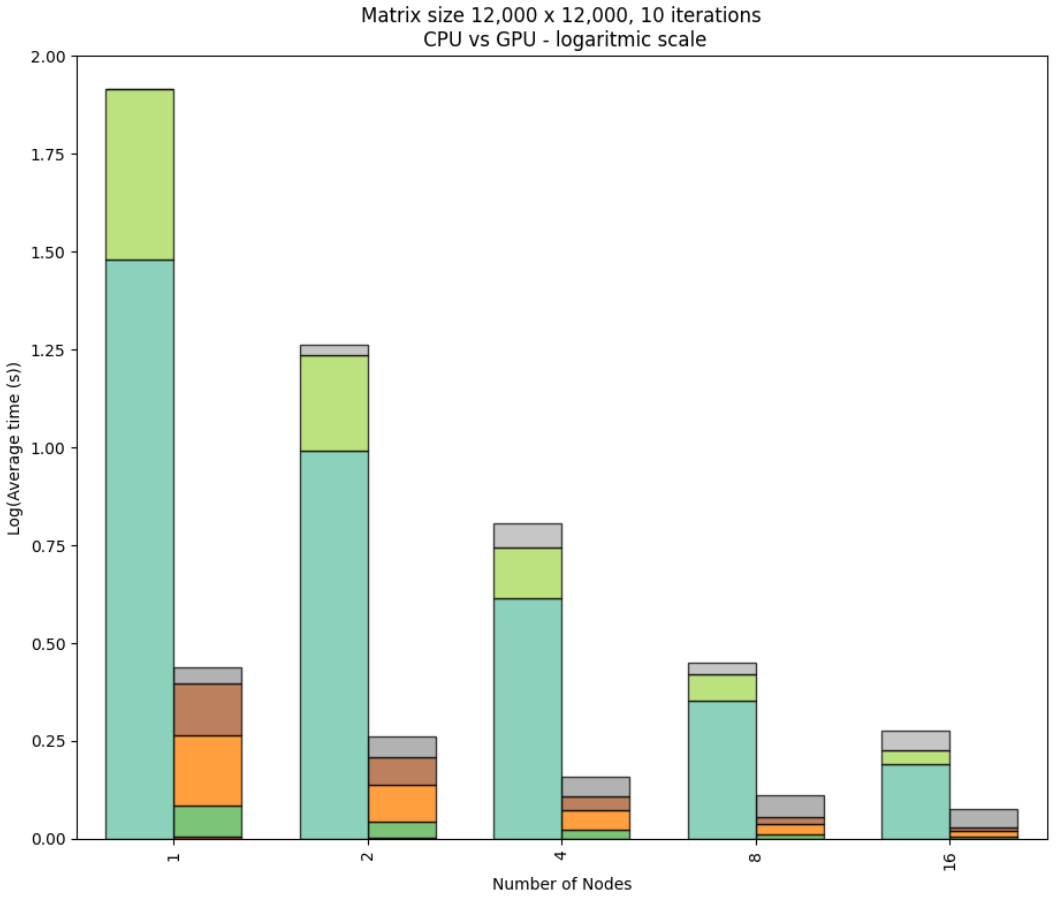

Matrix 1,200 x 1,200, 10 iterations
computation on GPU
Average time(s)
0.00
0.01
0.02
0.03
0.04
0.05
0.06
1.0
2.0
4.0
8.0
16.0
32.0
Number of nodes


computation
copy-matrix-cpu-to-gpu
copy-matrix-gpu-to-cpu
matrix-initialization
mpi-send-rec


Matrix 12,000 x 12,000, 10 iterations
computation on GPU
Average time(s)
0.0
0.1
0.2
0.3
0.4
1.0
2.0
4.0
8.0
16.0
32.0
Number of nodes

Results: CPU vs GPU





Jacobi-one-side