

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: cblas_dgemm function
 - Cuda: cublasDgemm function
- Multi-node computation with MPI
- Scaling test

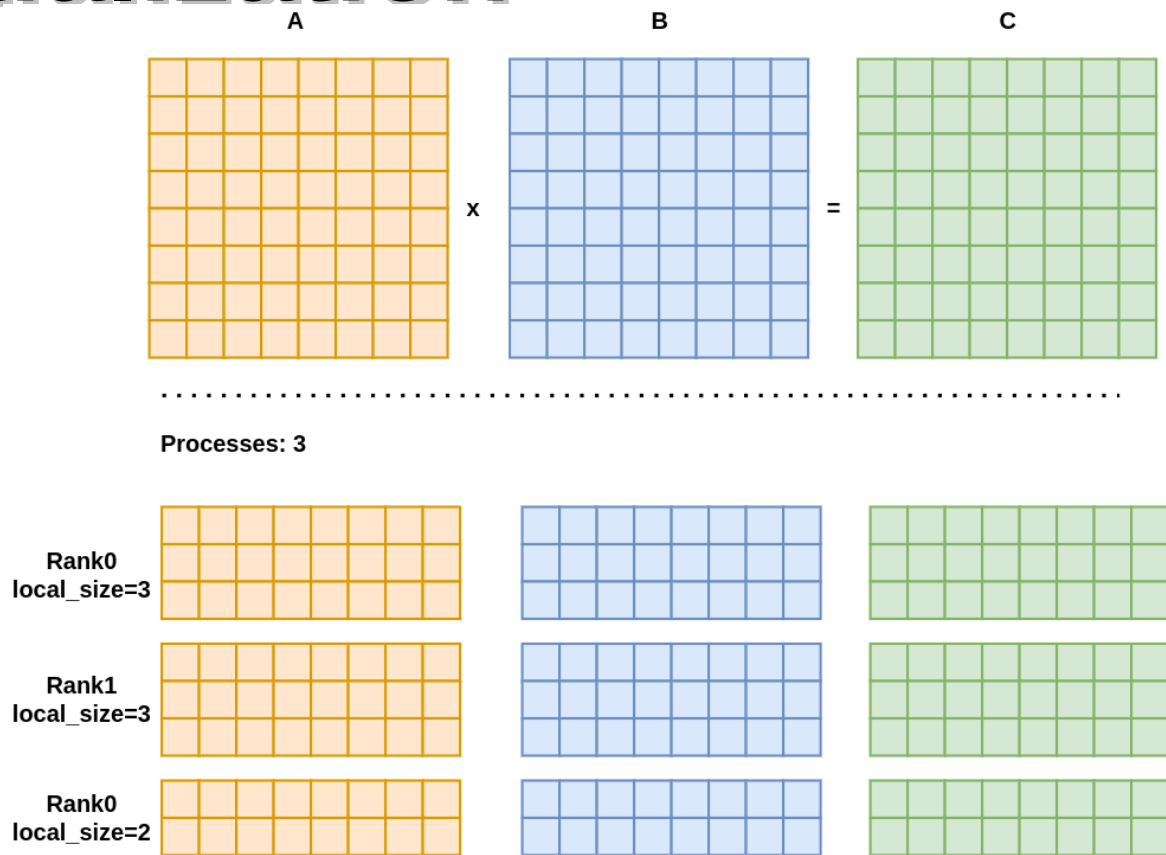
Implementation

```
#include <...>

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

Distributed initialization

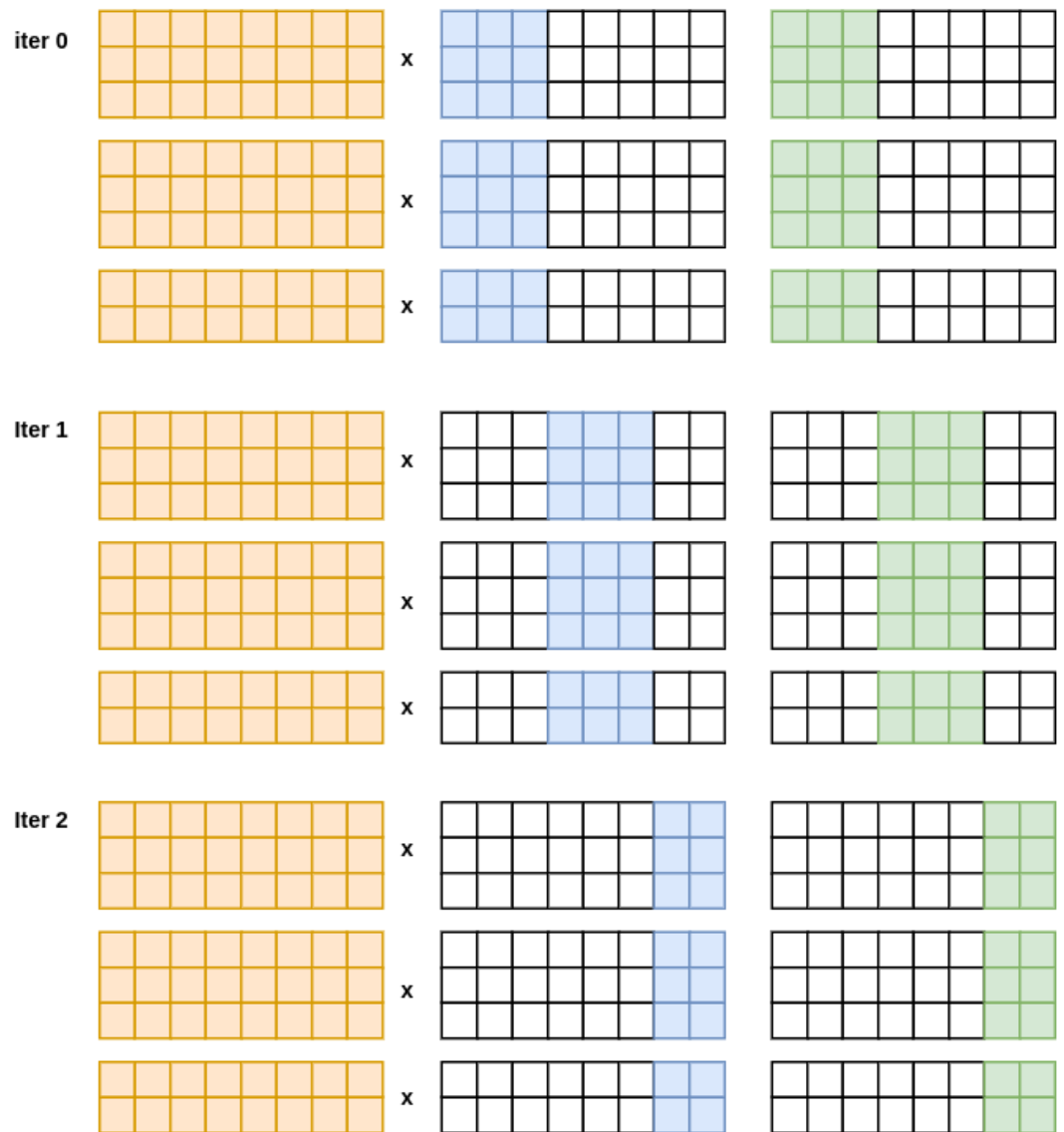
- Matrices divided by rows
- (Almost) homogeneous distribution of the workload between processes



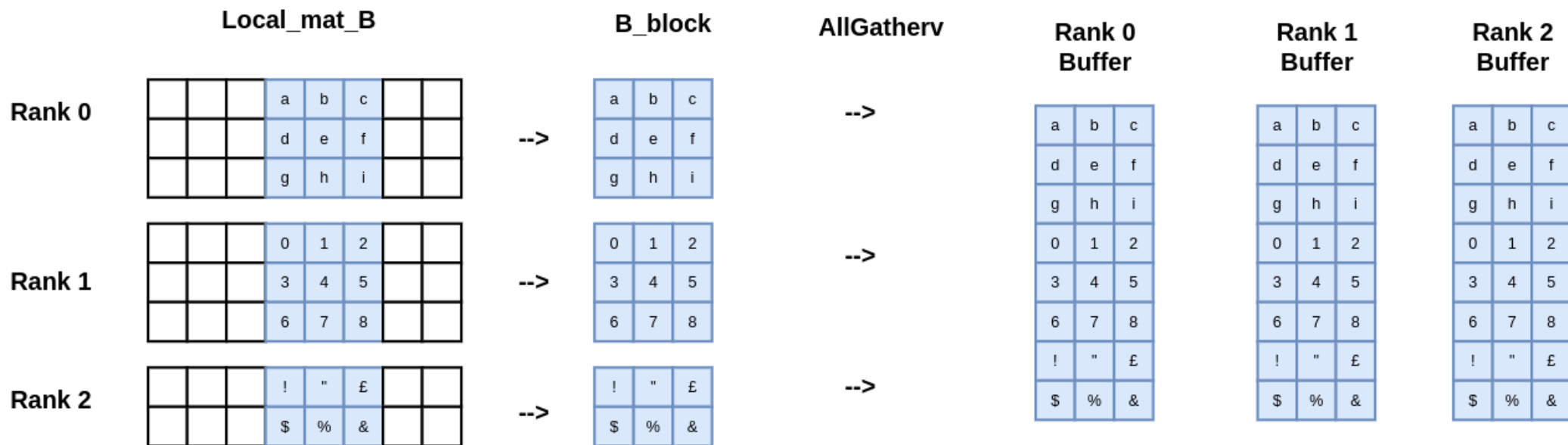
$$\text{local_size} = (\text{rank} < N \% \text{size}) ? N / \text{size} + 1 : N / \text{size};$$

Loop over the # of processes

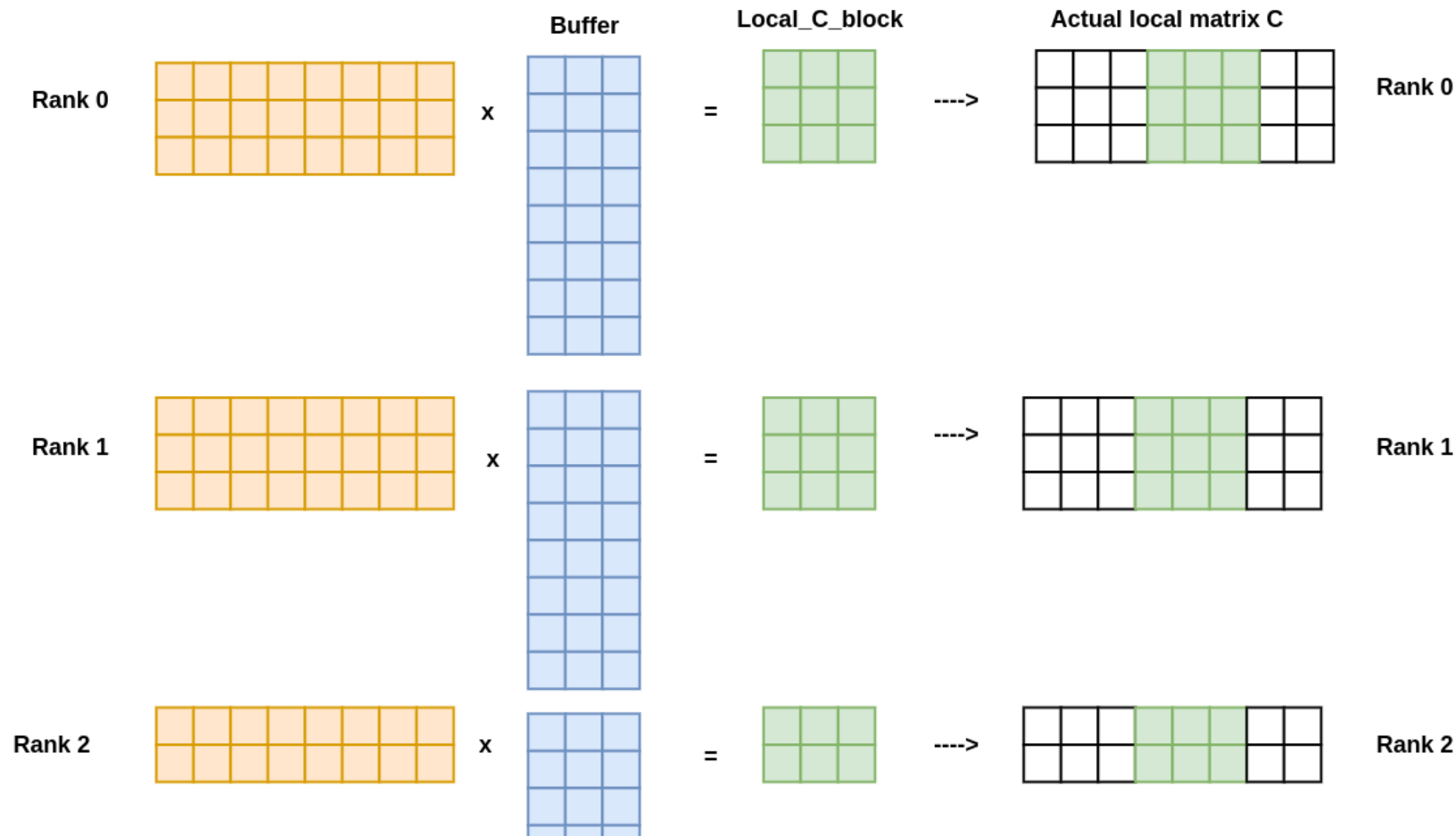
- Create B_{block}
- Communicate it to other processes
- Compute C_{block}



Gather the column block



Result: C_block



Results: a initial disclaimer

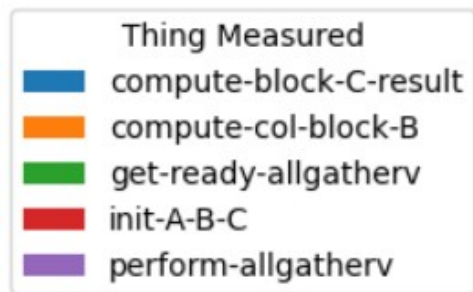
 `mpirun -np $nproc ./main.x`

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

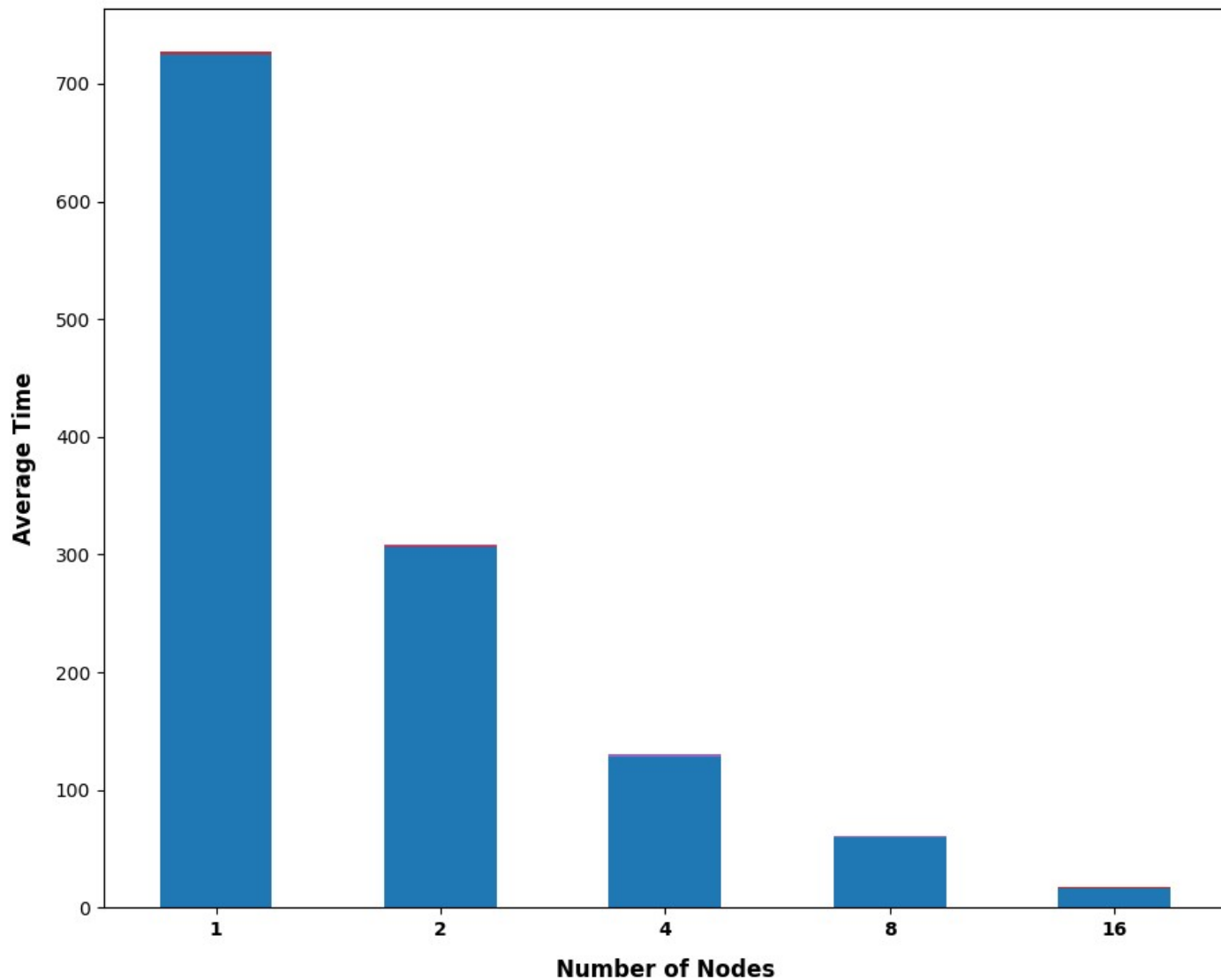
 `srun -N $nproc ./main.x`

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

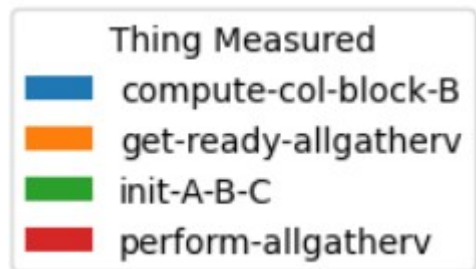
Results: 3-nested loop



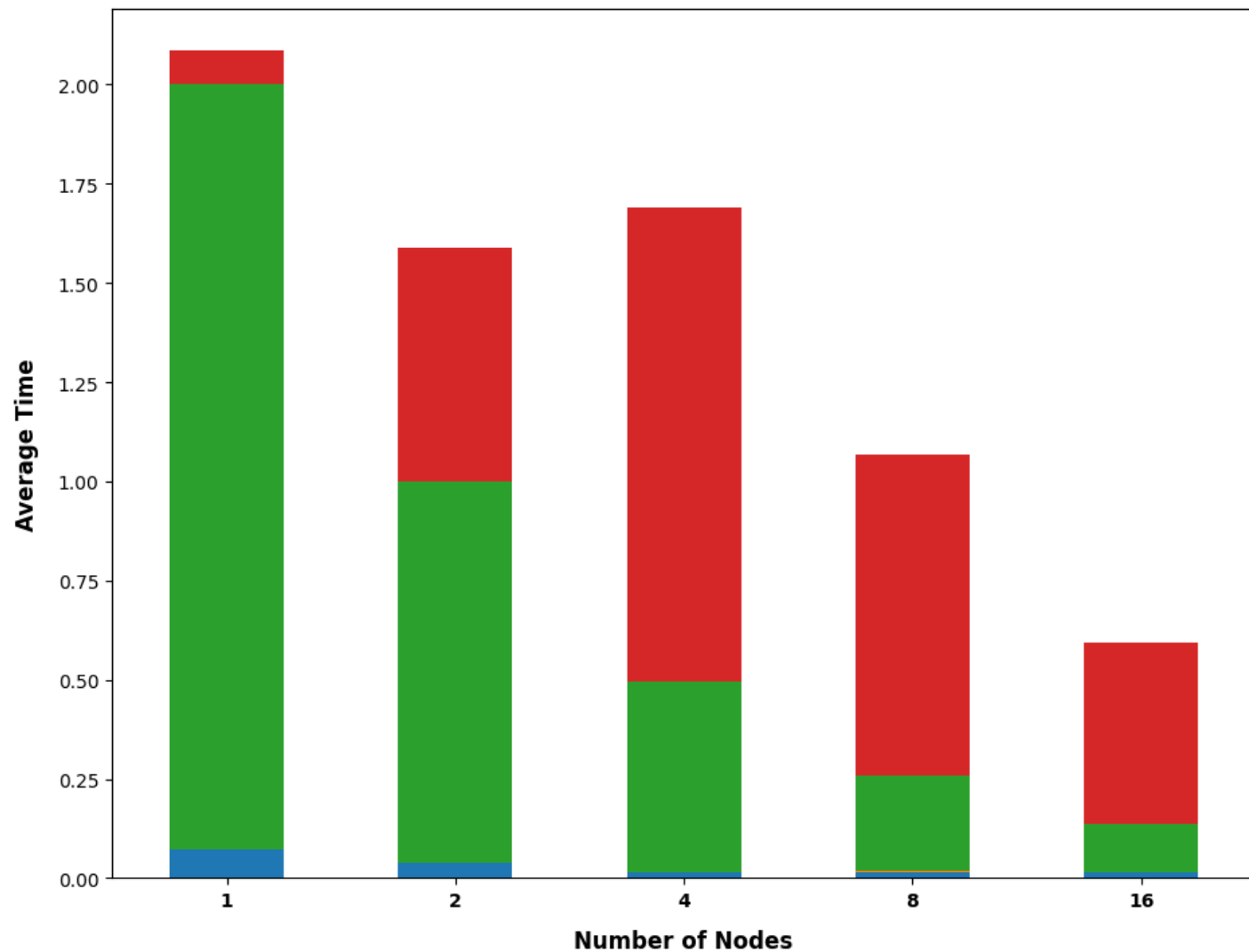
Average time: 3-nested loops
Matrix size: 5000x5000



Results: 3-nested loop (Cont'd)

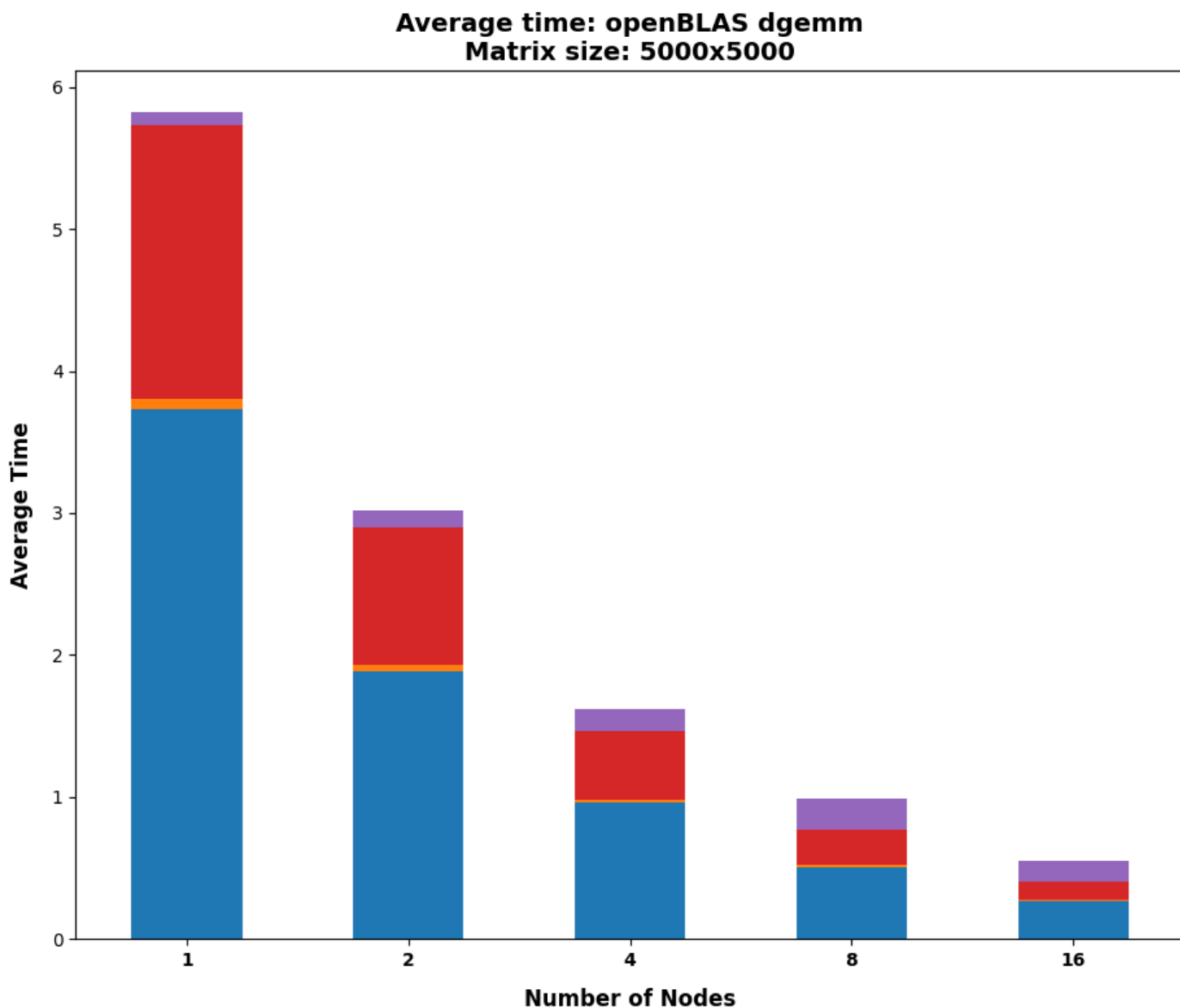
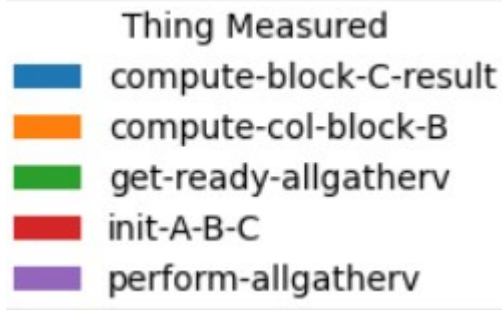


Average time: 3-nested loops
without compute-block-C-result
Matrix size: 5000x5000

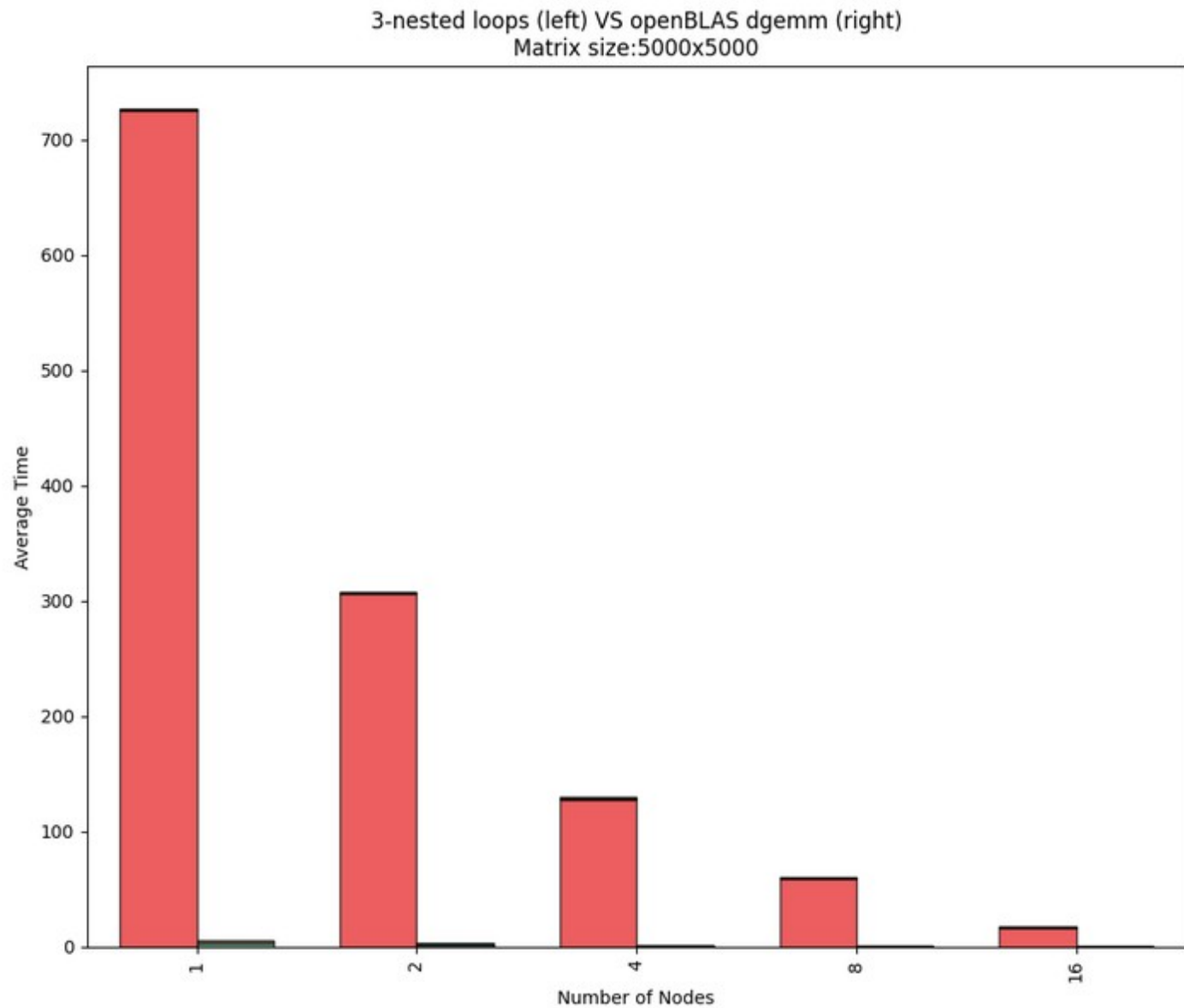


Results: BLAS

- In theory it should not scale so good

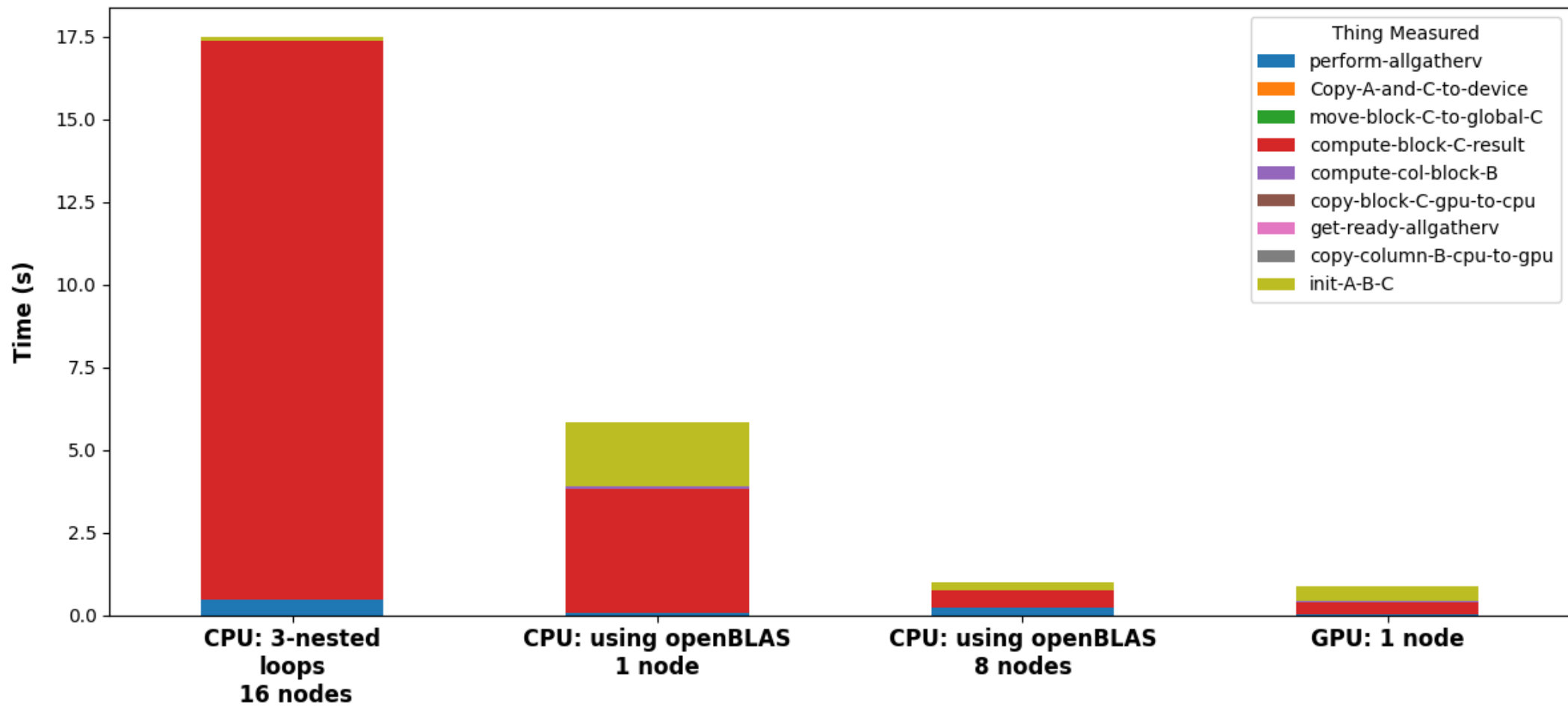


Results: CPU



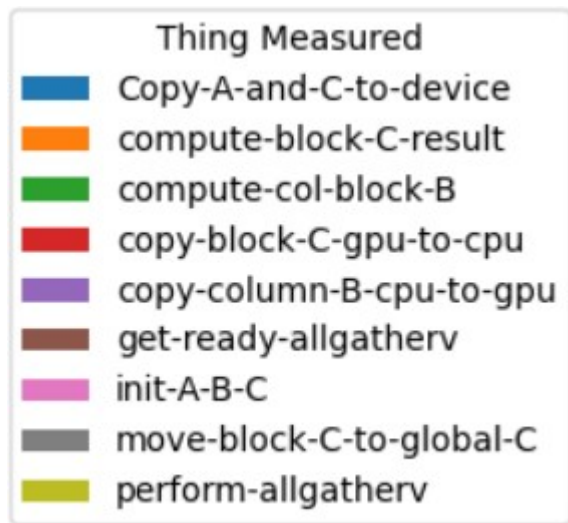
Results: cuda

Matrix-matrix multiplication
Size: 5000x5000

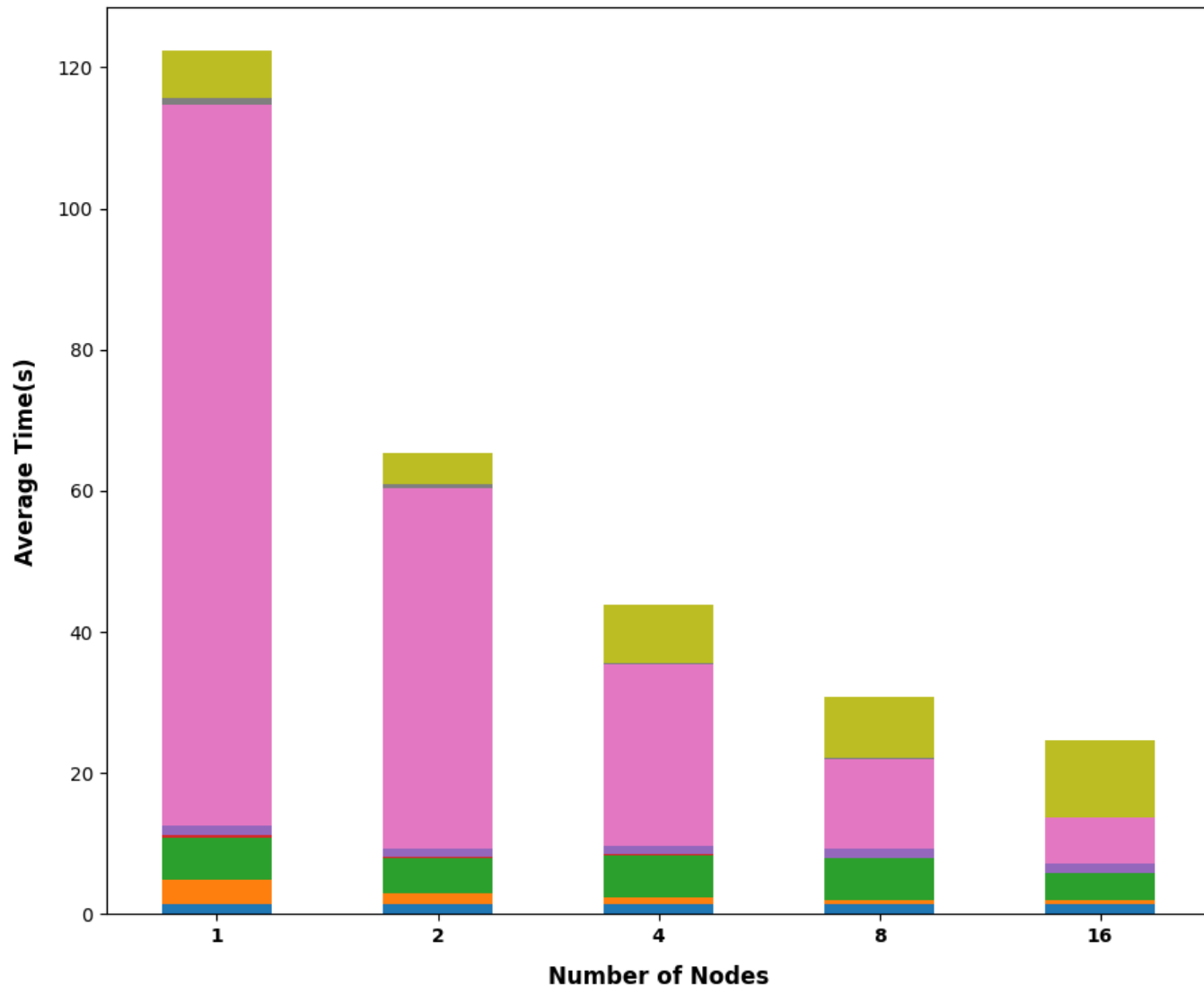


Results: cuda (Cont'd)

- Increased the matrix size to 75,000



Average time: GPU
Matrix size: 75000x75000



Jacobi

Requirements

- Take an *already existing* code `jacobi.c`
- Make it parallel using MPI
- Port the code on GPUs using 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);
```

```
    move_local_mat_to_device(local_matrix, mat_dev);
```

```
    for (int i = 0; i < n_iter; i++)
```

```
    {
```

```
        Communicate_with_neighbours(mat_dev, rank, size);
```

```
        evolve(local_matrix);
```

```
    }
```

```
    bring_back_local_mat_to_host(mat_dev, local_matrix);
```

```
    save_resluts(local_matrix);
```

```
    return 0;
```

```
}
```

```
//cpu
```

```
//cpu->gpu
```

```
//gpu
```

```
//gpu
```

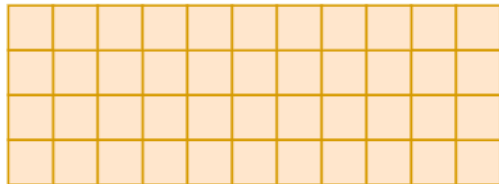
```
//gpu->cpu
```

```
//cpu
```

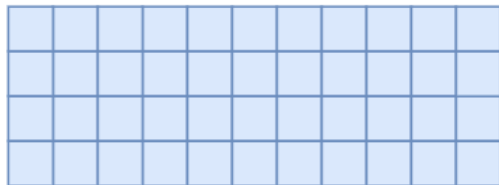
Communication

Local portion of the matrix the process must take care of

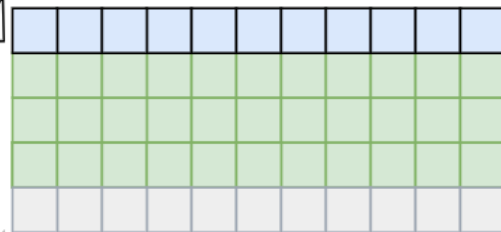
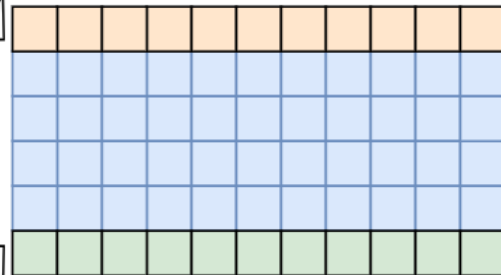
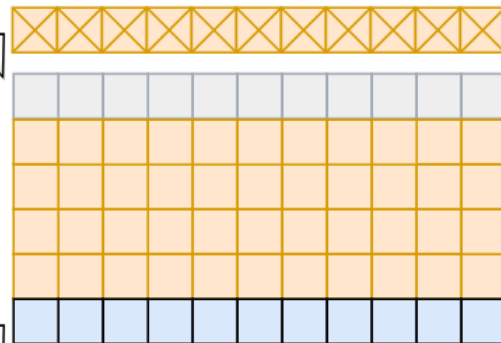
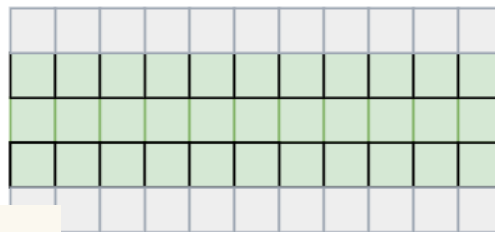
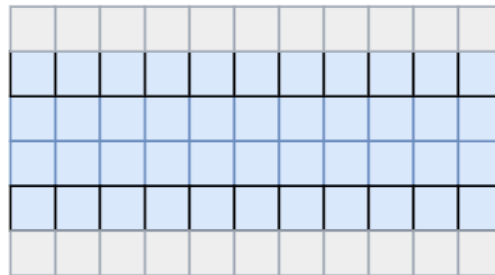
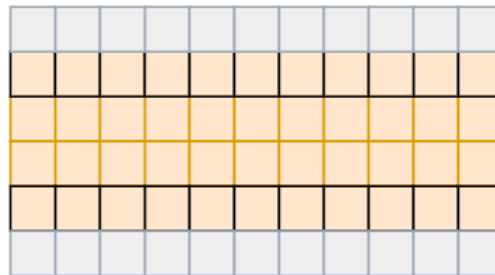
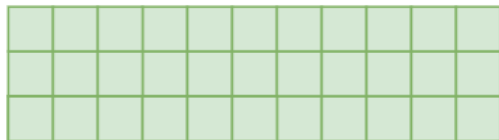
Rank0



Rank1



Rank2



```
//send up,recv bottom
int send_to = (rank - 1) >= 0 ? rank - 1 : MPI_PROC_NULL;
int rcv_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);  
}
```

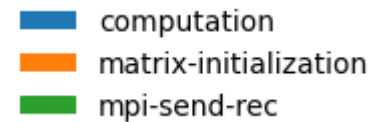
```
//swap the pointers on the device
```

```
#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;  
}
```

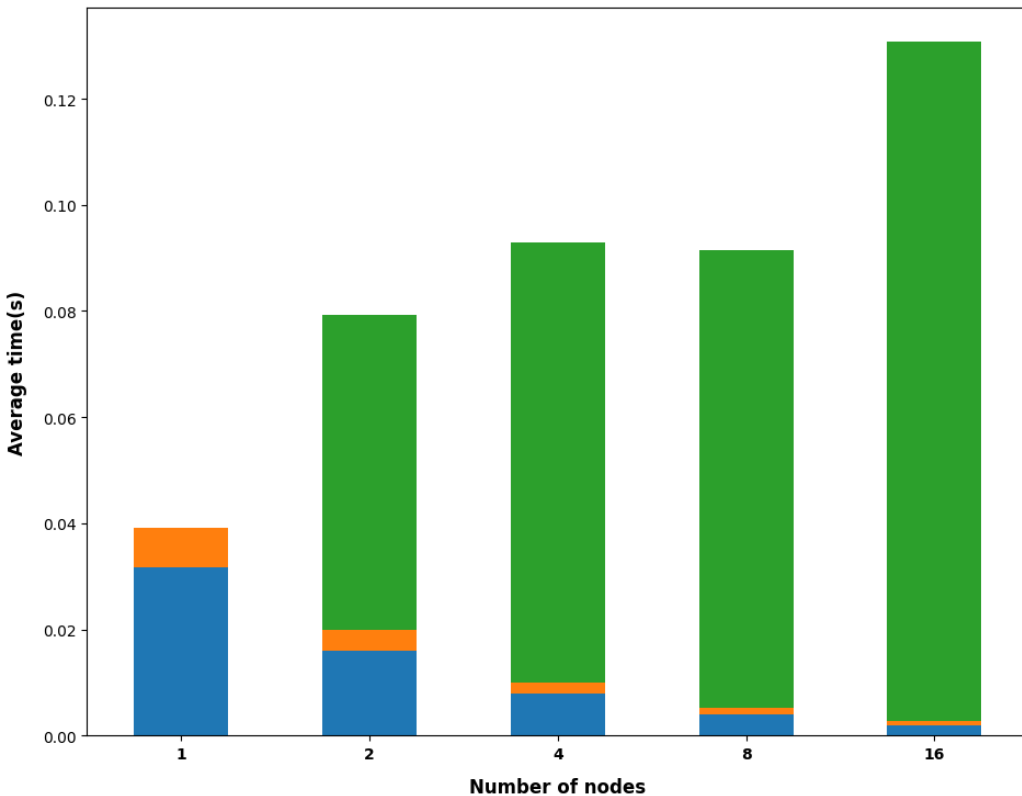
```
// swap pointer on the host, needed to preserve data consistency
```

```
tmp_matrix = matrix;  
matrix = matrix_new;  
matrix_new = tmp_matrix;
```

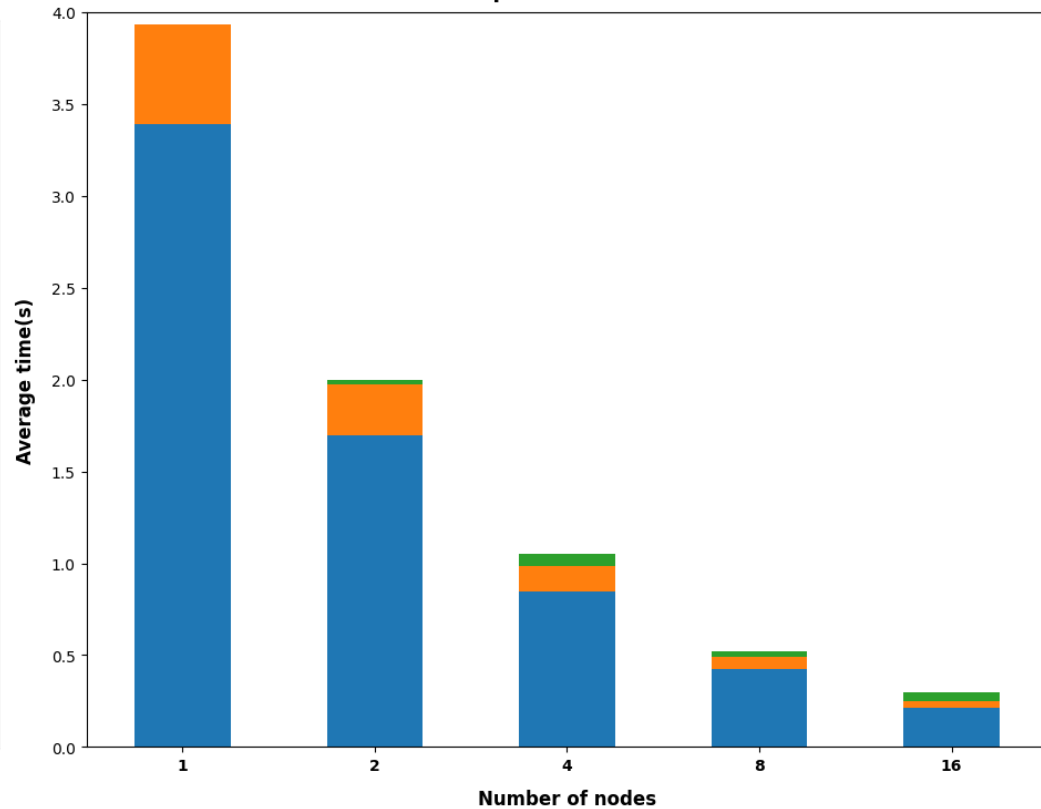
Results: CPU



Matrix 1,200 x 1,200, 10 iterations
computation on CPU



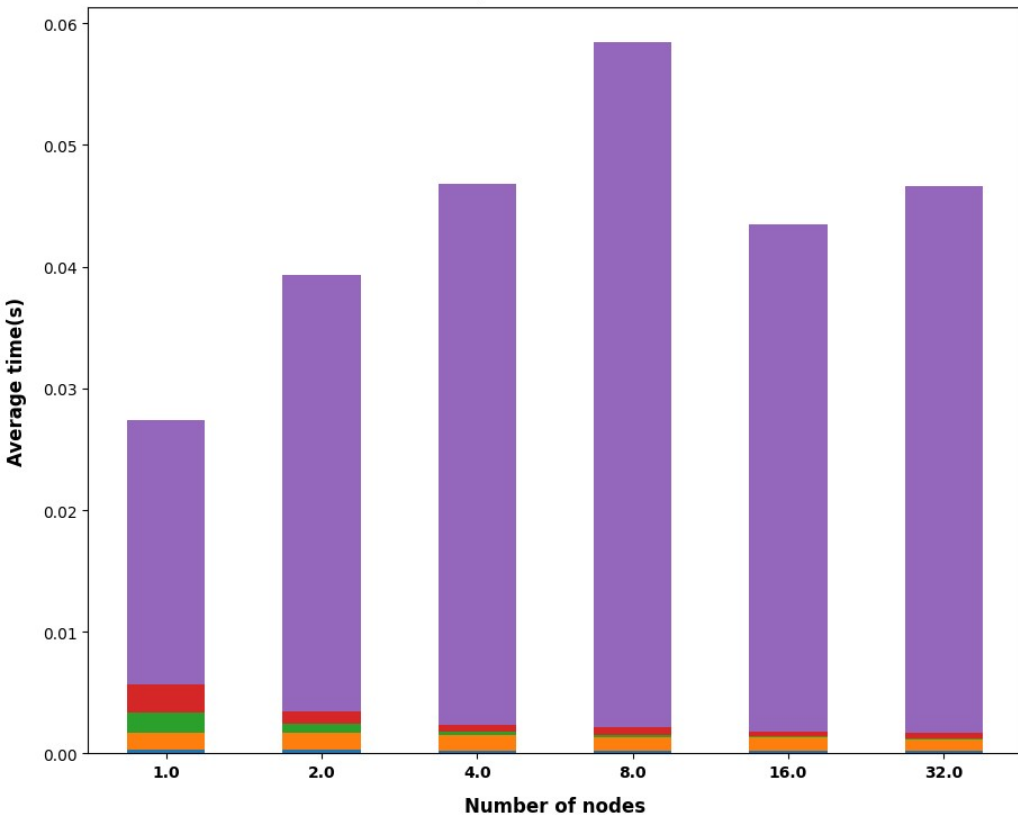
Matrix 12,000 x 12,000, 10 iterations
computation on CPU



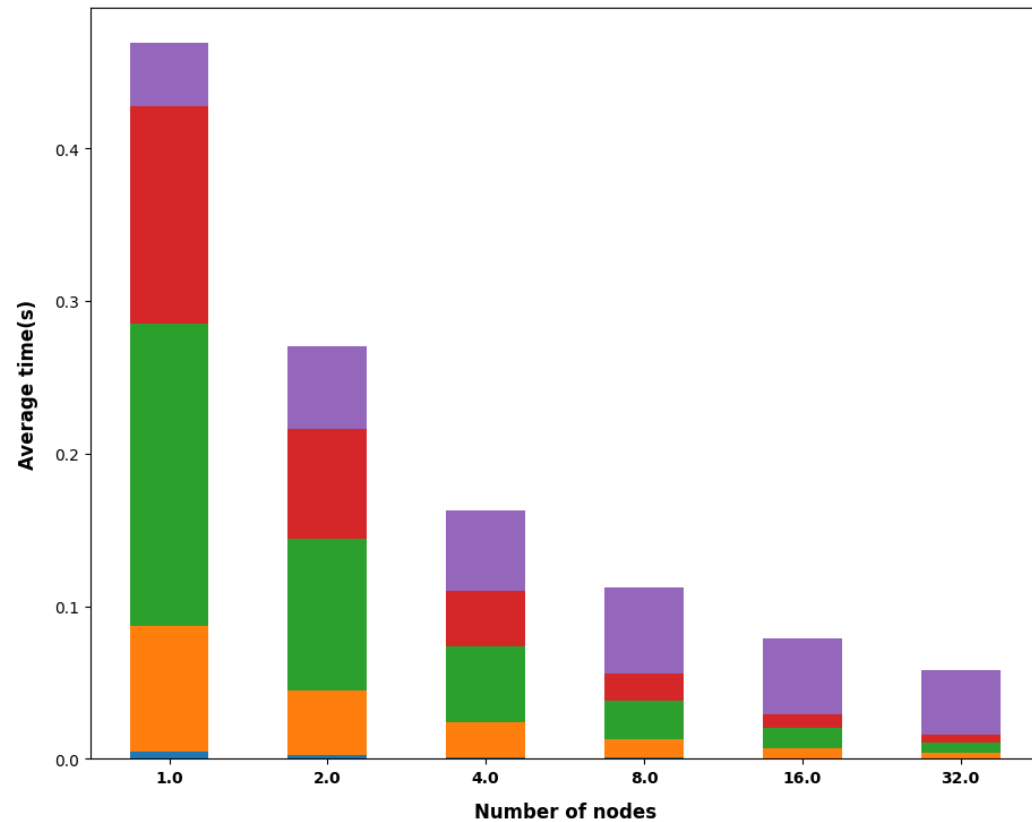
Results: GPU

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

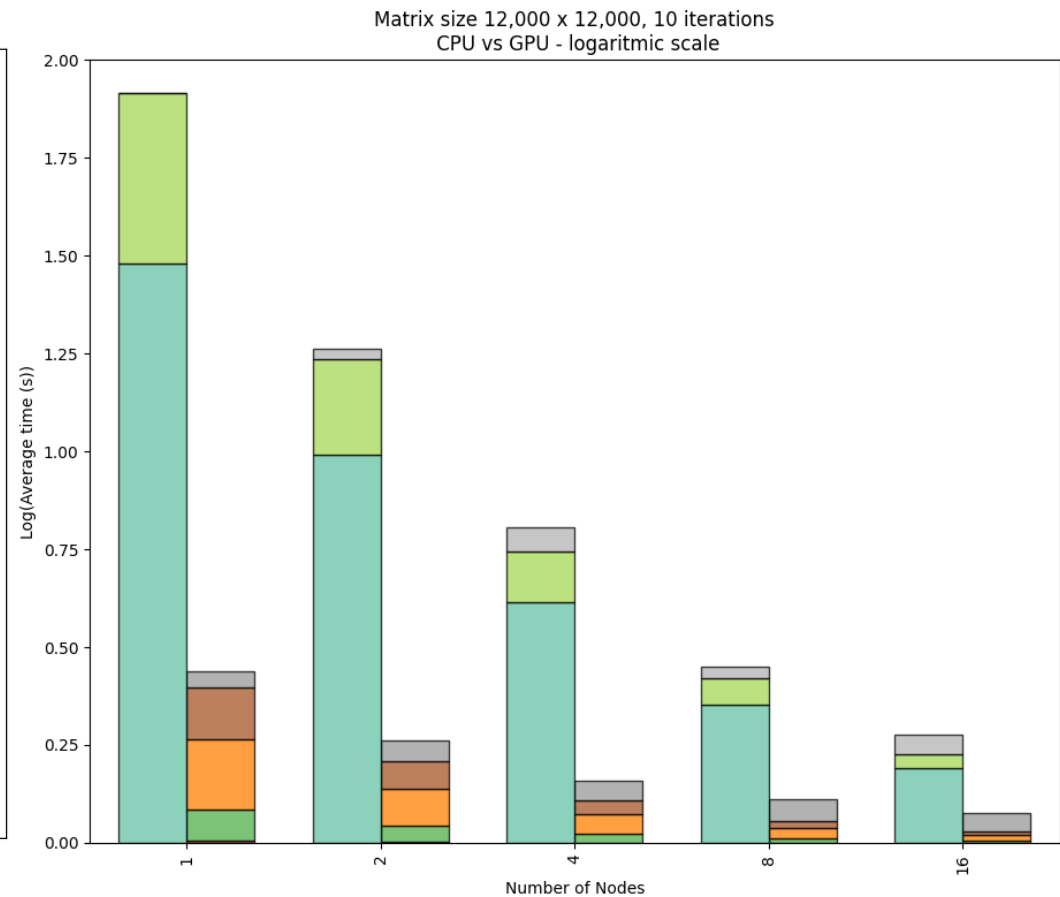
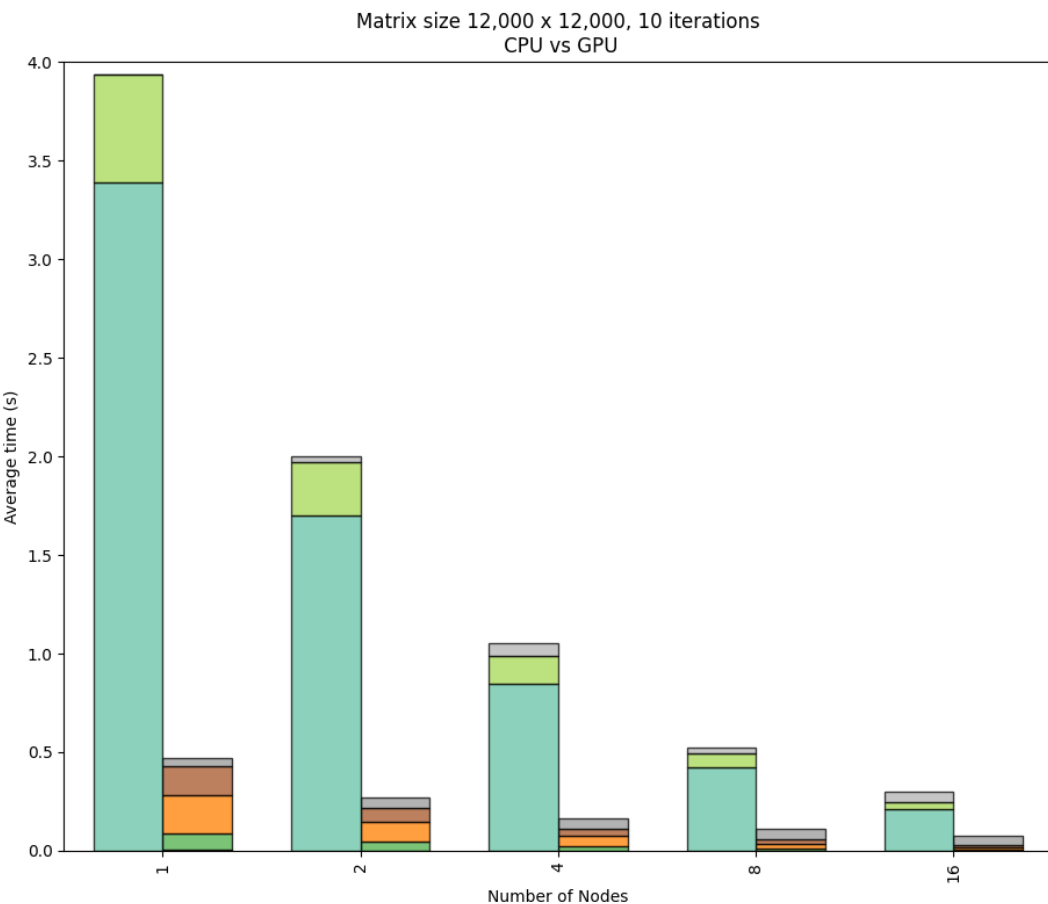
Matrix 1,200 x 1,200, 10 iterations
computation on GPU



Matrix 12,000 x 12,000, 10 iterations
computation on GPU



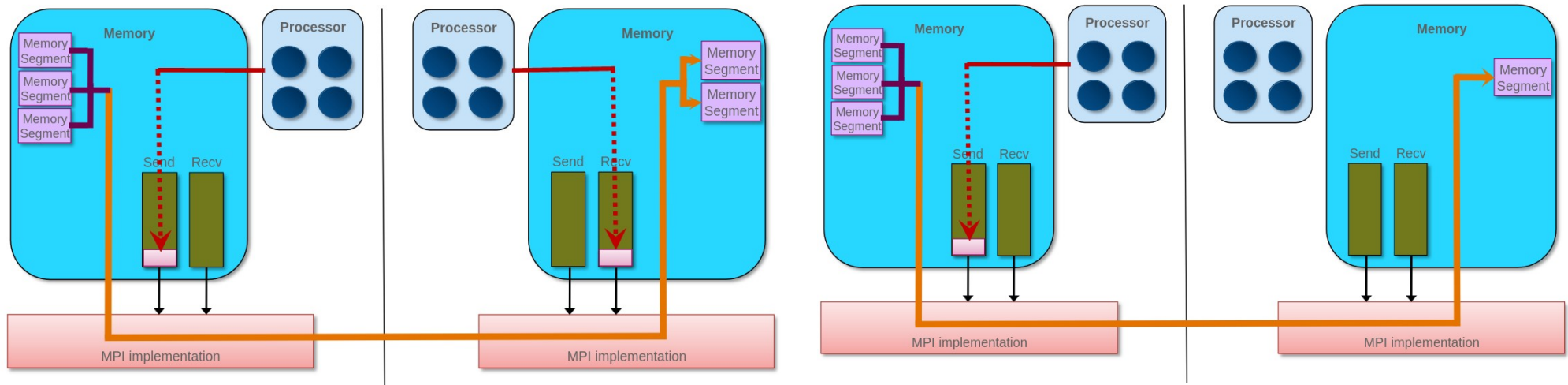
Results: CPU vs GPU



Jacobi-one-side

Requirements

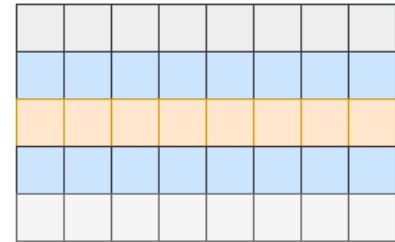
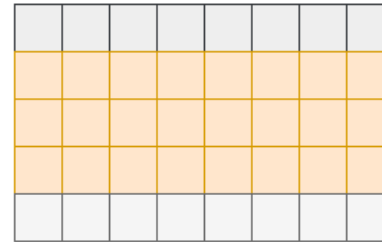
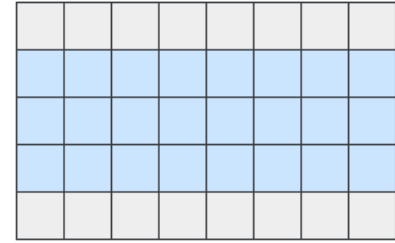
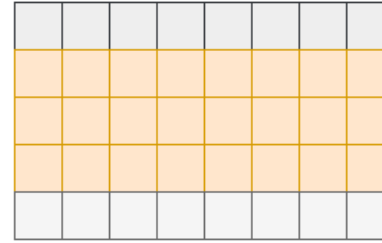
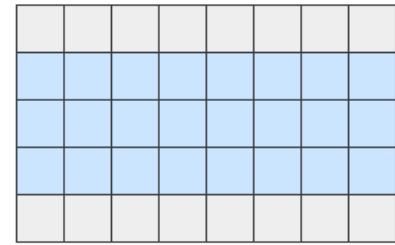
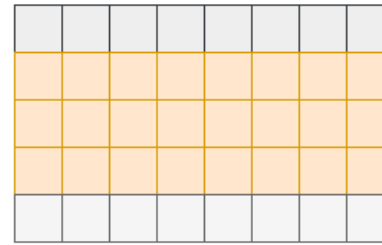
- Reimplement the *previous exercise* jacobi.c
- Use MPI-RMA approach to perform communications between processes



Implementation

<i>MPI operation used</i>	
1 MPI_Win MPI_PUT	2 MPI_Win MPI_GET
2 MPI_Win MPI_PUT	2 MPI_Win MPI_GET

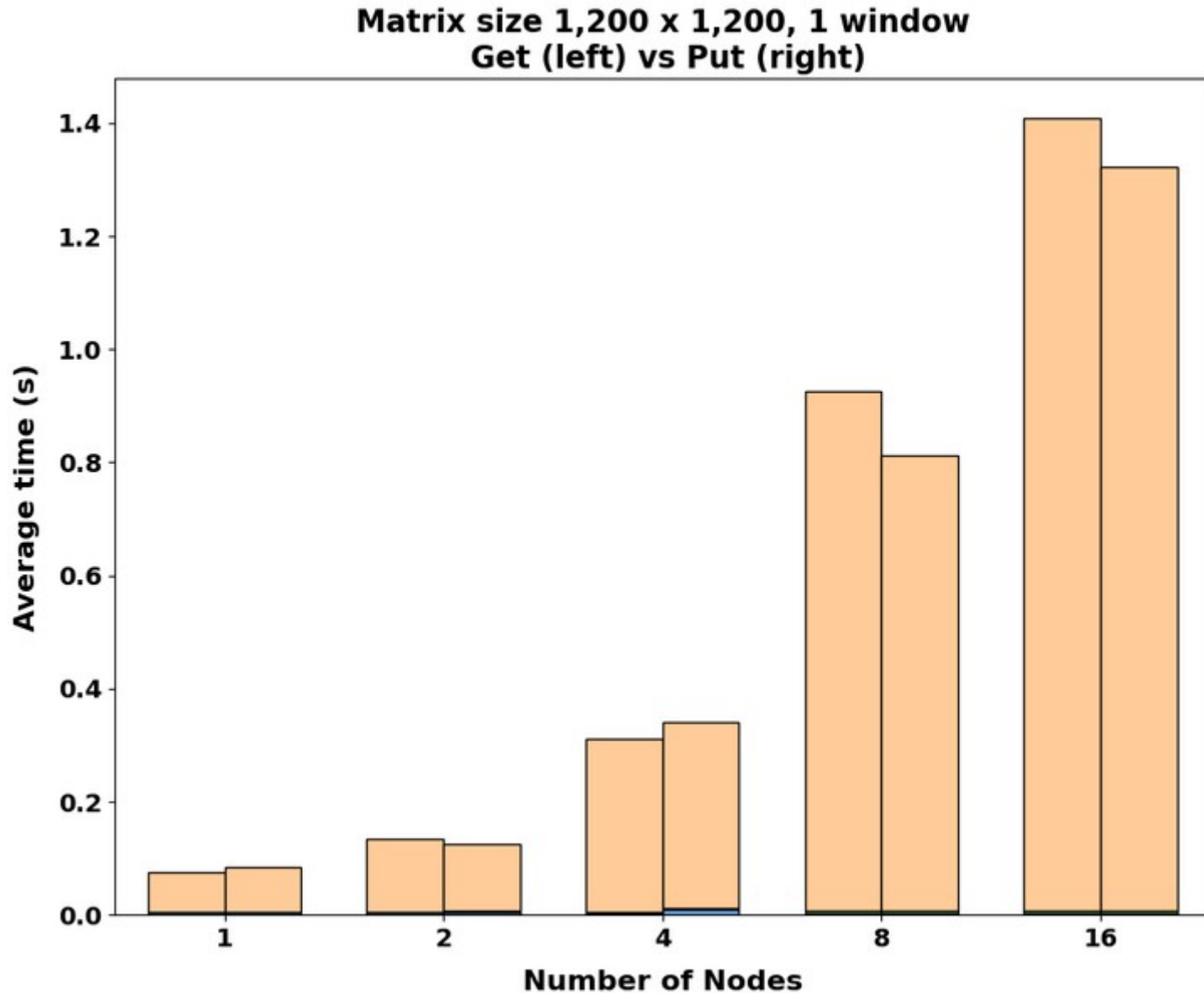
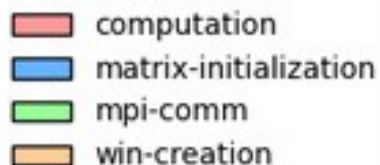
of Windows



Results:

GENERAL CONSIDERATION:

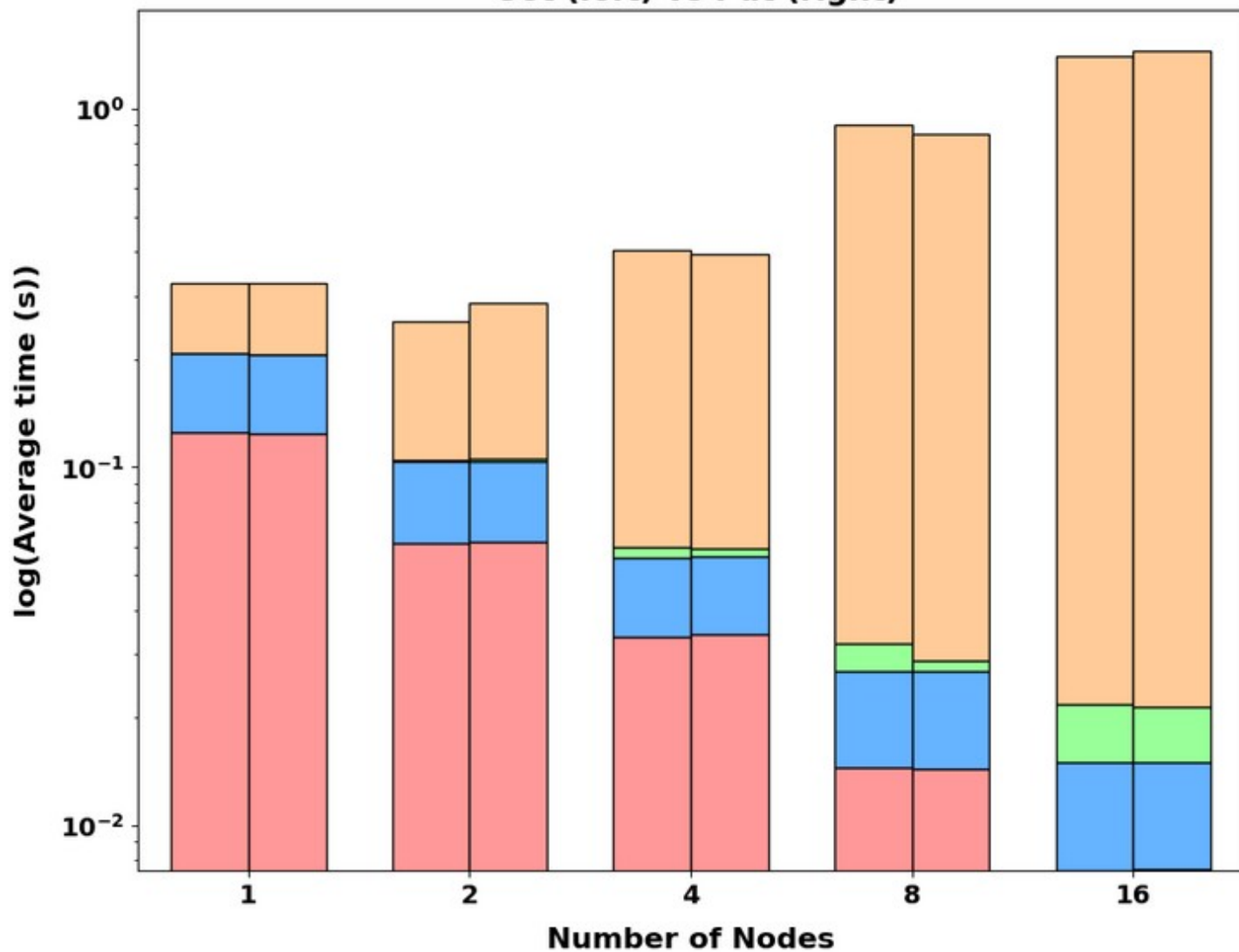
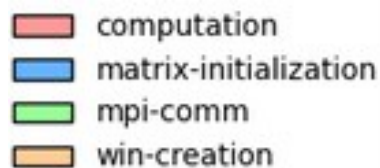
The time spent to create the memory window obliterates all the other components



Results:

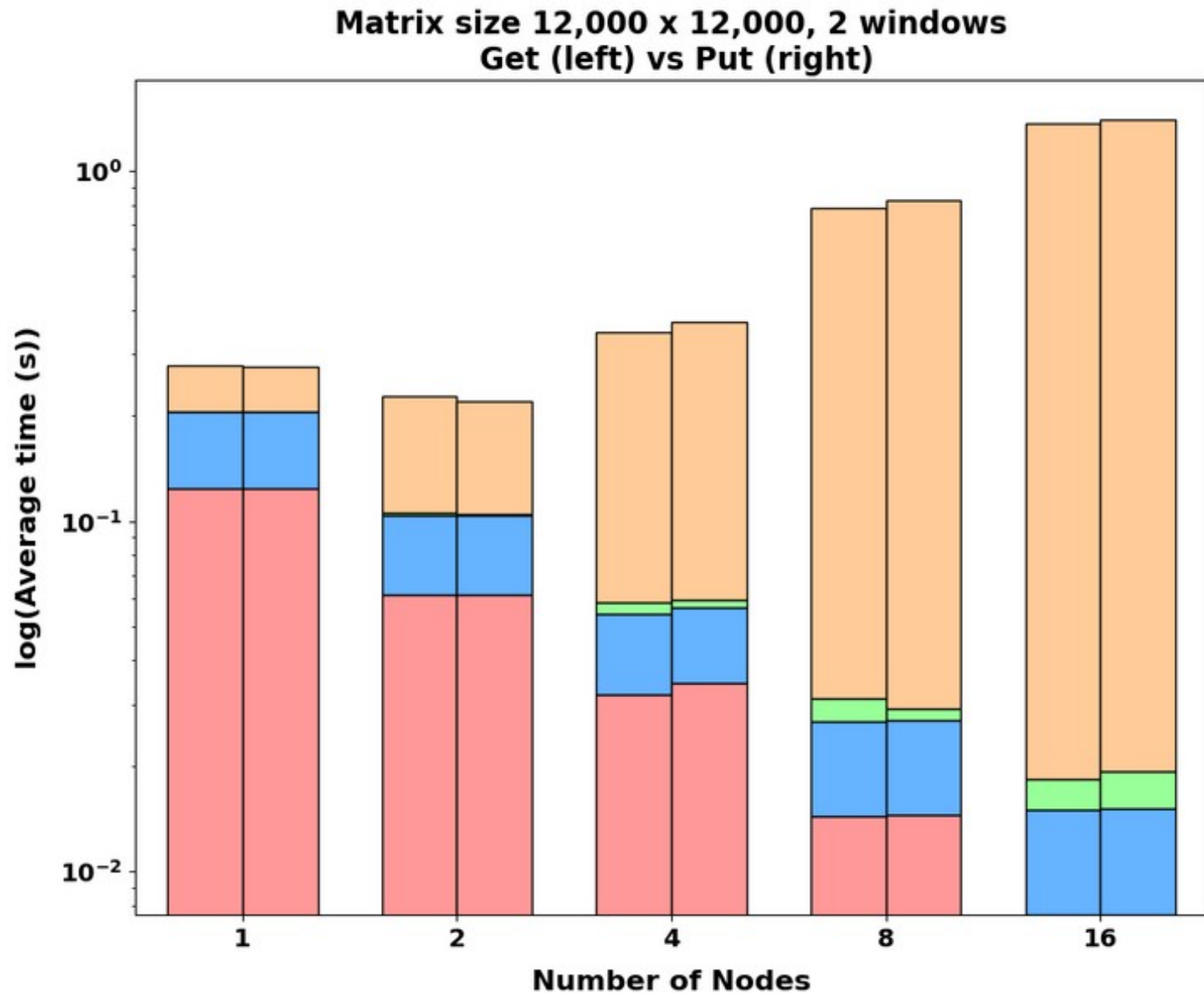
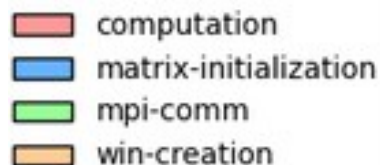
PUT VS GET

Matrix size 12,000 x 12,000, 1 window
Get (left) vs Put (right)



Results:

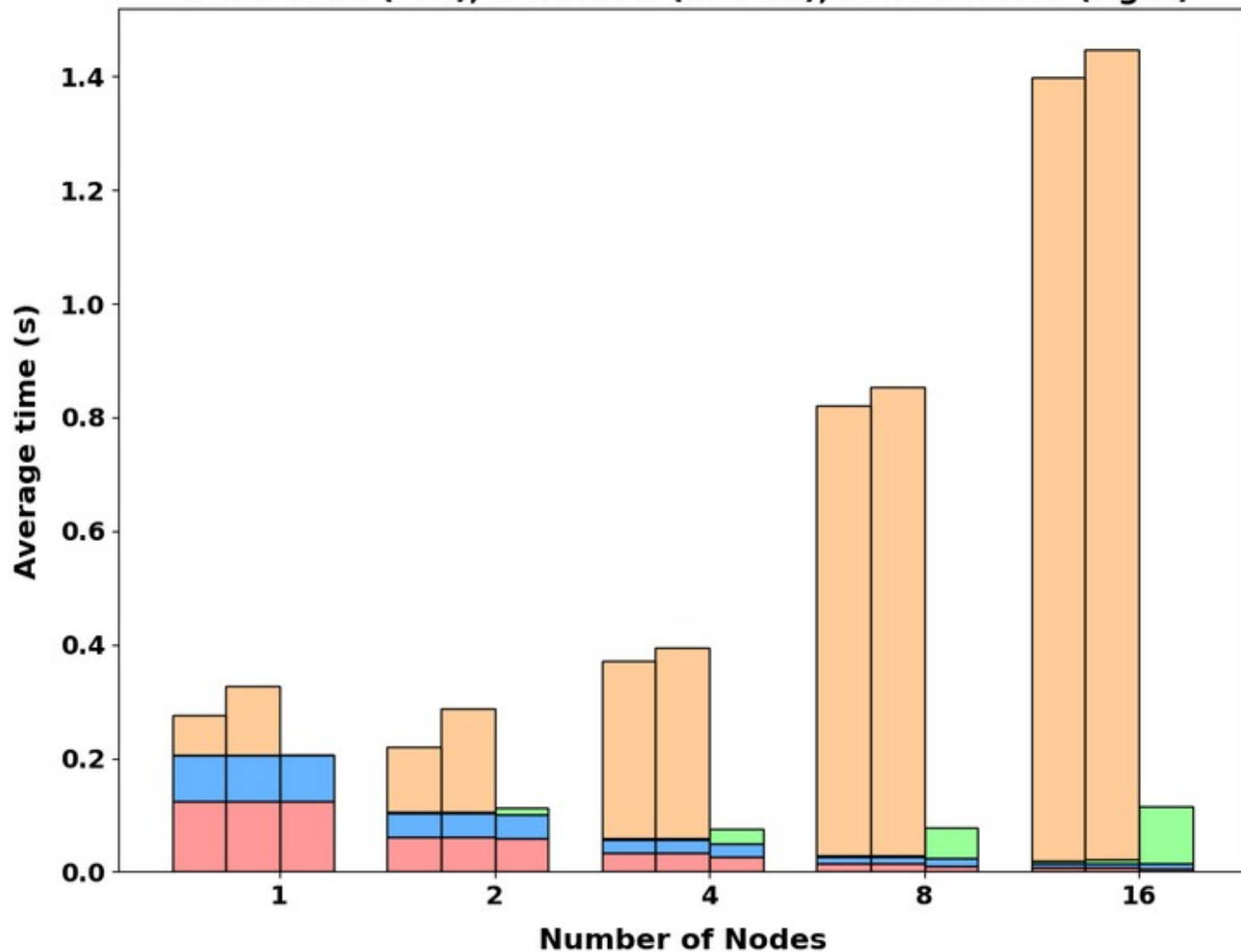
PUT VS GET (Cont'd)



Results:

1 vs 2 MPI_Win

Matrix size 12,000 x 12,000, Put
2 windows (left), 1 window (middle), send-receive (right)

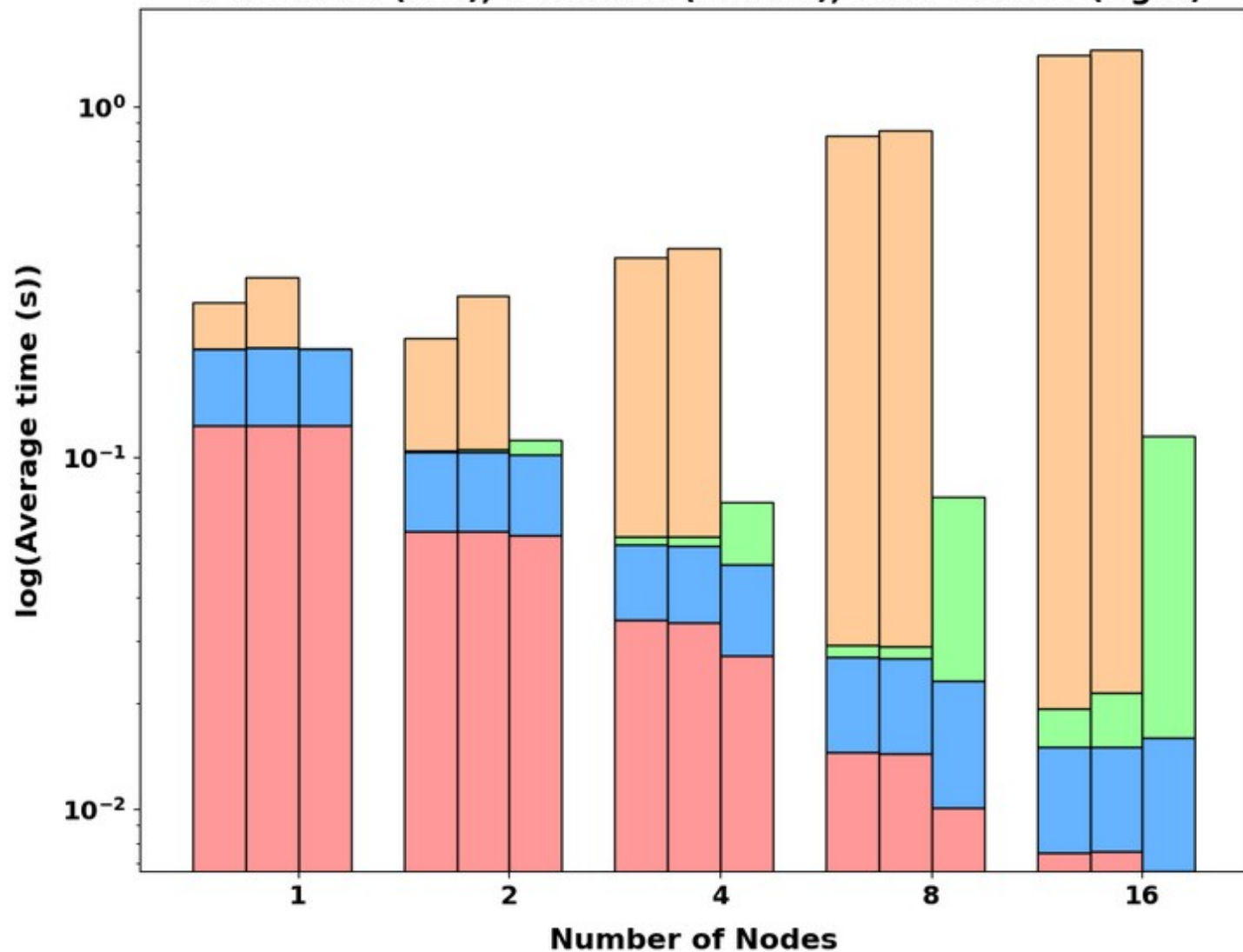
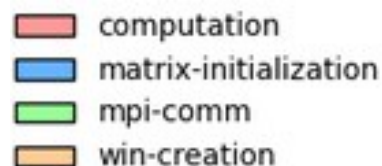


computation
matrix-initialization
mpi-comm
win-creation

Results:

1 vs 2 MPI_Win (Cont'd)

Matrix size 12,000 x 12,000, Put
2 windows (left), 1 window (middle), send-receive (right)



Thank you for the attention