FRANKFURT UNIVERSITY OF APPLIED SCIENCES

MPI Special Challenge 3

Matrices multiplication using MPI

Cloud Computing | Computer Science and Engineering | WS 2017/18

14.12.2017, 10:00 – 13:15

Project Team

Karen Gharslyan

Introduction and Coordination

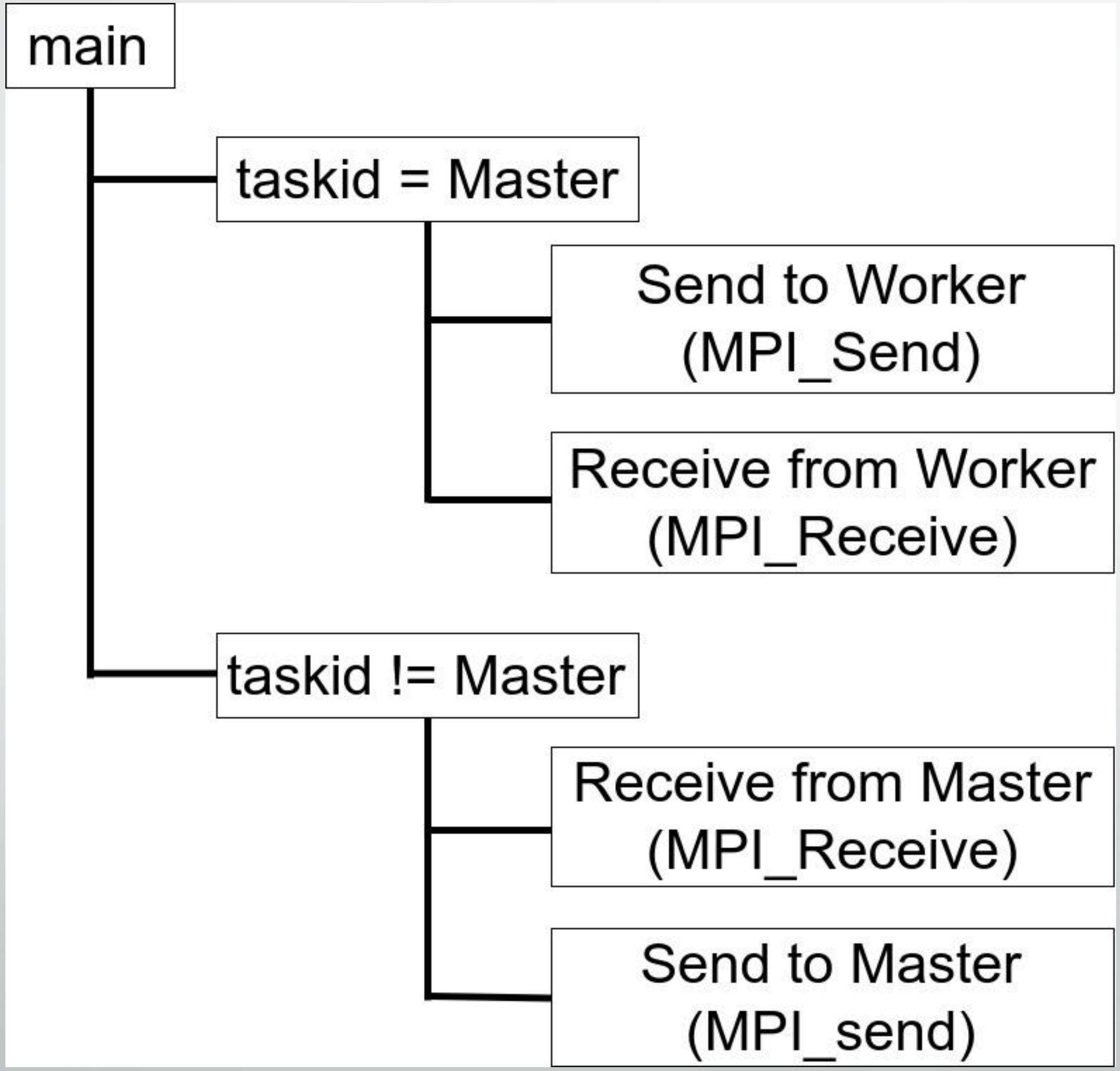
Rodion Slepnev

Theory and realization of algorithm

Minh Nguyen

Test cases

Same program for Master and Worker



main

taskid = Master

Send to Worker (MPI_Send)

Receive from Worker (MPI_Receive)

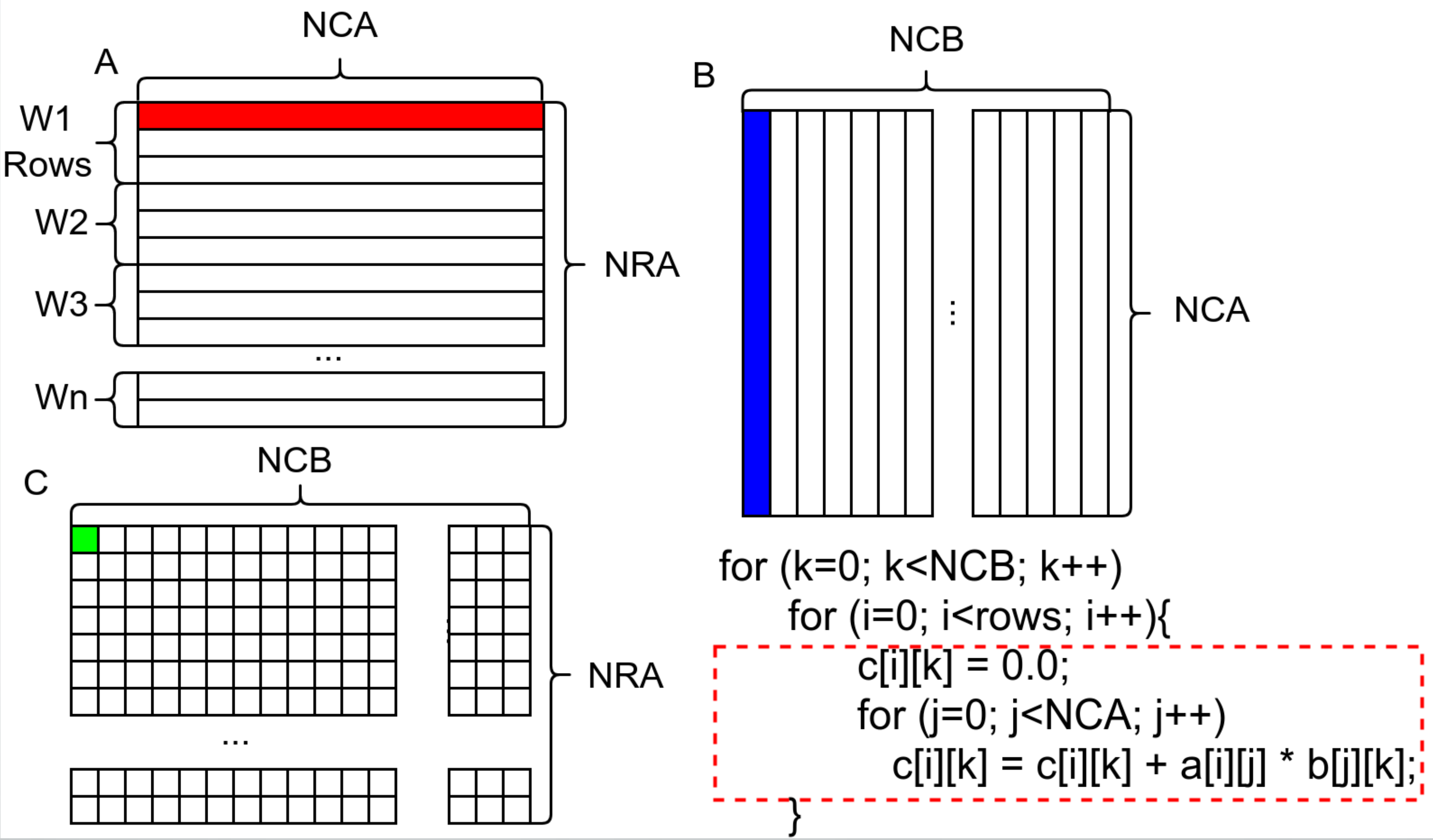
taskid != Master

Receive from Master (MPI_Receive)

Send to Master (MPI_send)

Matrix Multiplication (1-3)

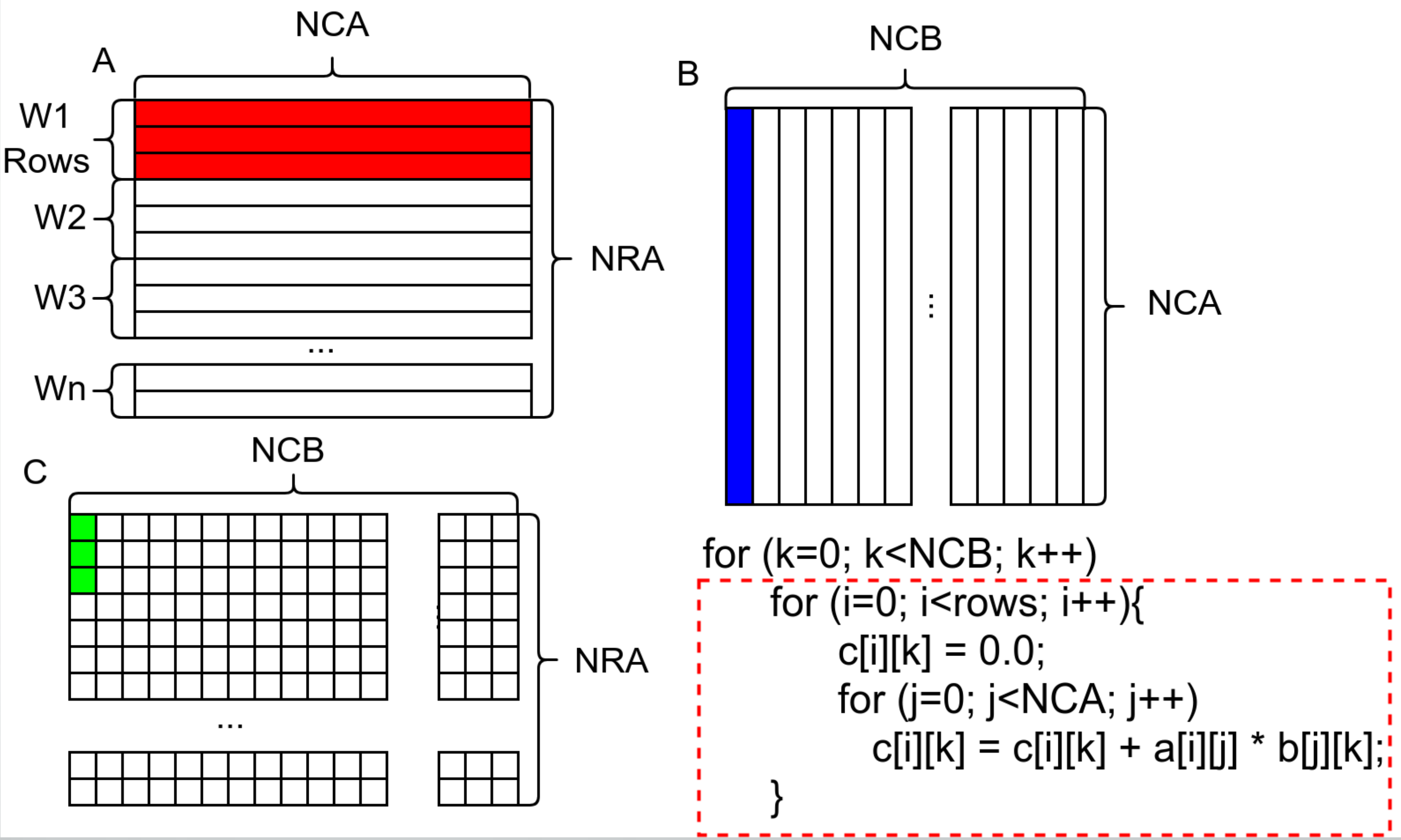
Row on column



```
for (k=0; k<NCB; k++)
     for (i=0; i<rows; i++){
          c[i][k] = 0.0;
          for (j=0; j<NCA; j++)
             c[i][k] = c[i][k] + a[i][j] * b[j][k];
     }
```

Matrix Multiplication (2-3)

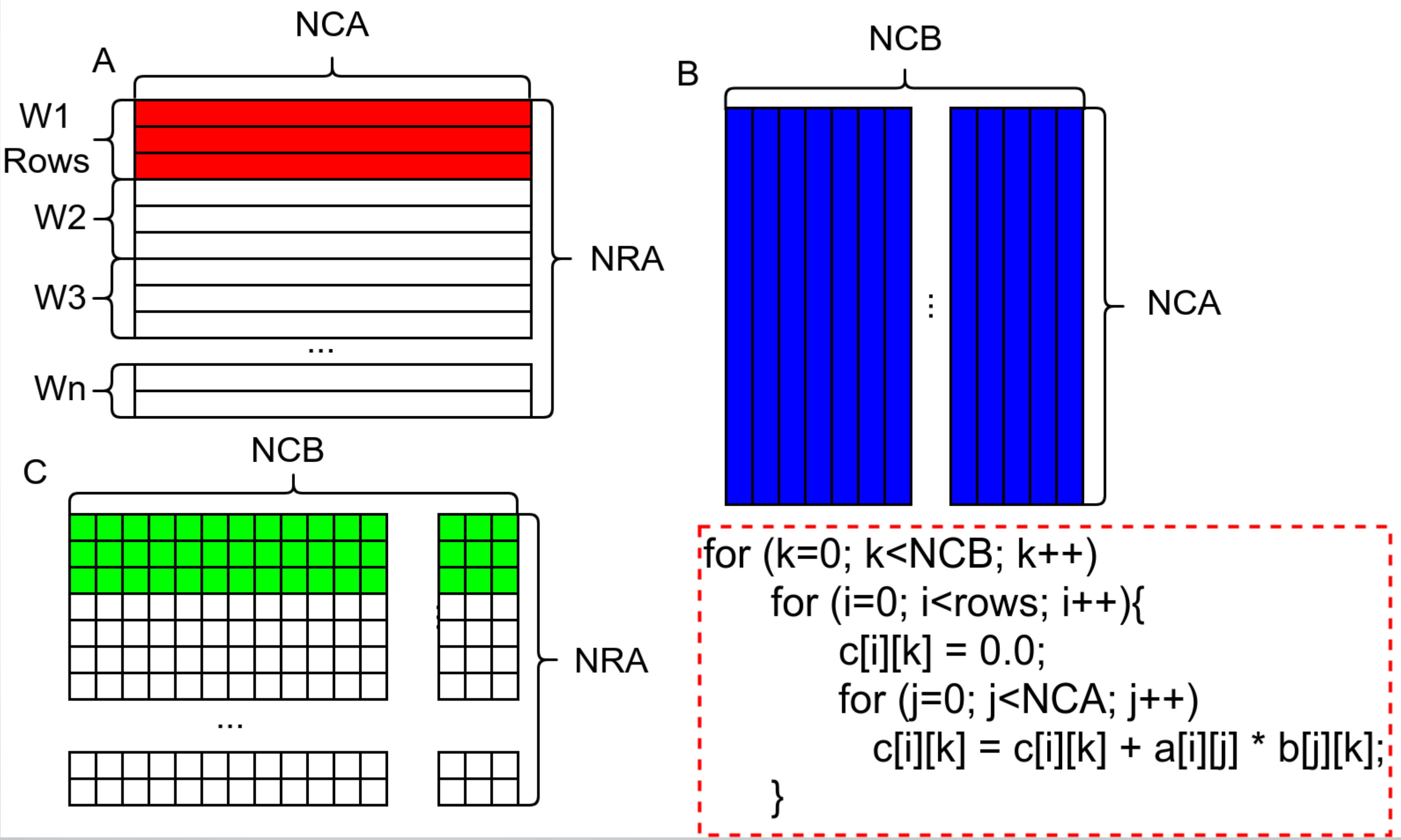
Rows on column



```
for (k=0; k<NCB; k++)
    for (i=0; i<rows; i++){
        c[i][k] = 0.0;
        for (j=0; j<NCA; j++)
           c[i][k] = c[i][k] + a[i][j] * b[j][k];
    }
```

Matrix Multiplication (3-3)

Rows on matrix



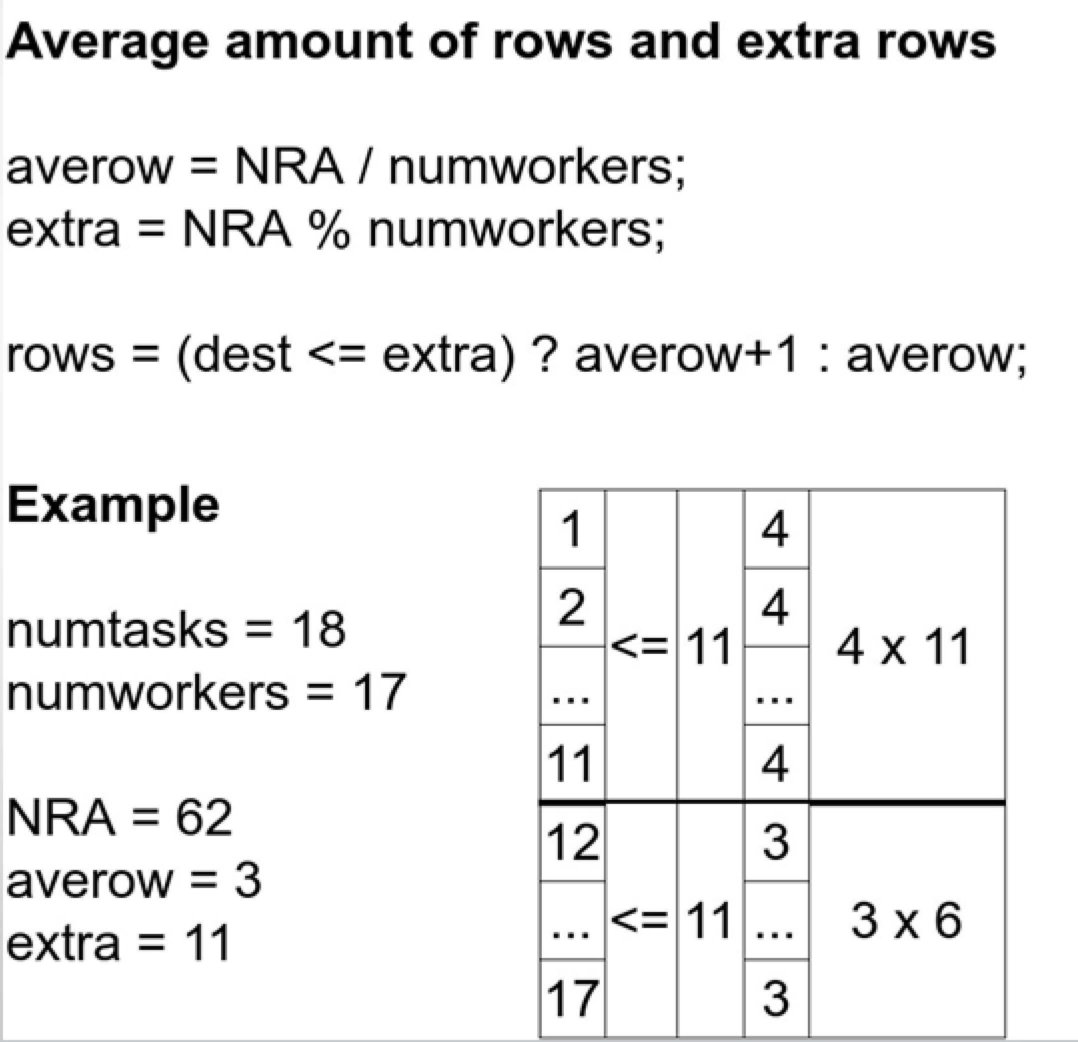
```
for (k=0; k<NCB; k++)
    for (i=0; i<rows; i++){
        c[i][k] = 0.0;
        for (j=0; j<NCA; j++)
            c[i][k] = c[i][k] + a[i][j] * b[j][k];
    }
```

How tasks are assigned to Workers

- Ideas to Parallel Matrix Multiplication:
  - 1 single task: Multiplication of 1 row in matrix A to 1 column in matrix B
  - A x B with A (a x c) and B (c x b) will have in total a*b tasks
- Simple approach:
  - Divide tasks based on number of rows in matrix A
  - Rows are divided equally to each workers in cluster

How tasks are assigned to Workers

- Implementation



Average amount of rows and extra rows

```
averow = NRA / numworkers;
extra = NRA % numworkers;

rows = (dest <= extra) ? averow+1 : averow;
```

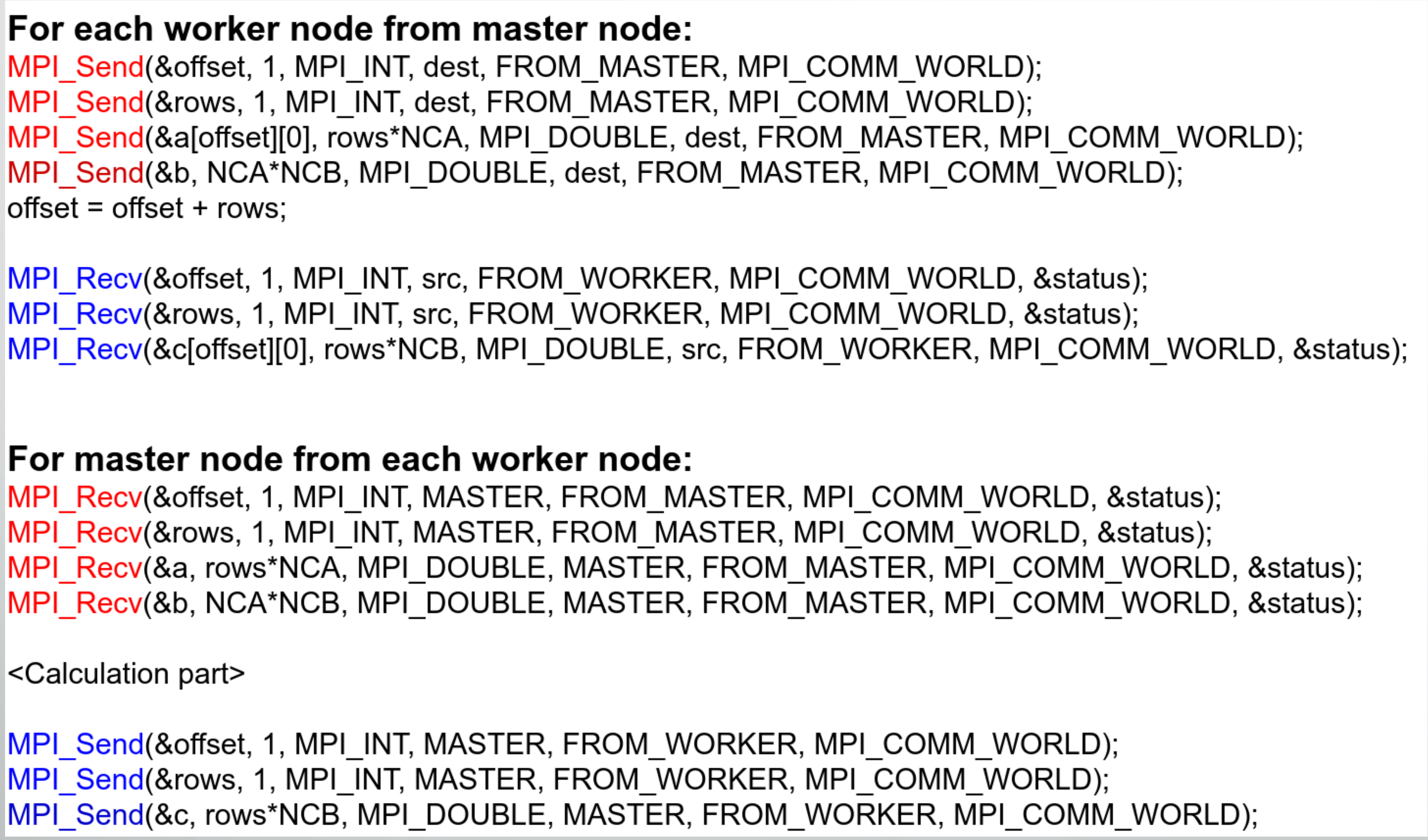
Example

numtasks = 18
numworkers = 17

NRA = 62
averow = 3
extra = 11

1	<=	11	4	4 x 11
2			4	
...			...	
11			4	
12	<=	11	3	3 x 6
...			...	
17			3	

Send and Receive MPI functions



For each worker node from master node:

```
MPI_Send(&offset, 1, MPI_INT, dest, FROM_MASTER, MPI_COMM_WORLD);
MPI_Send(&rows, 1, MPI_INT, dest, FROM_MASTER, MPI_COMM_WORLD);
MPI_Send(&a[offset][0], rows*NCA, MPI_DOUBLE, dest, FROM_MASTER, MPI_COMM_WORLD);
MPI_Send(&b, NCA*NCB, MPI_DOUBLE, dest, FROM_MASTER, MPI_COMM_WORLD);
offset = offset + rows;

MPI_Recv(&offset, 1, MPI_INT, src, FROM_WORKER, MPI_COMM_WORLD, &status);
MPI_Recv(&rows, 1, MPI_INT, src, FROM_WORKER, MPI_COMM_WORLD, &status);
MPI_Recv(&c[offset][0], rows*NCB, MPI_DOUBLE, src, FROM_WORKER, MPI_COMM_WORLD, &status);
```

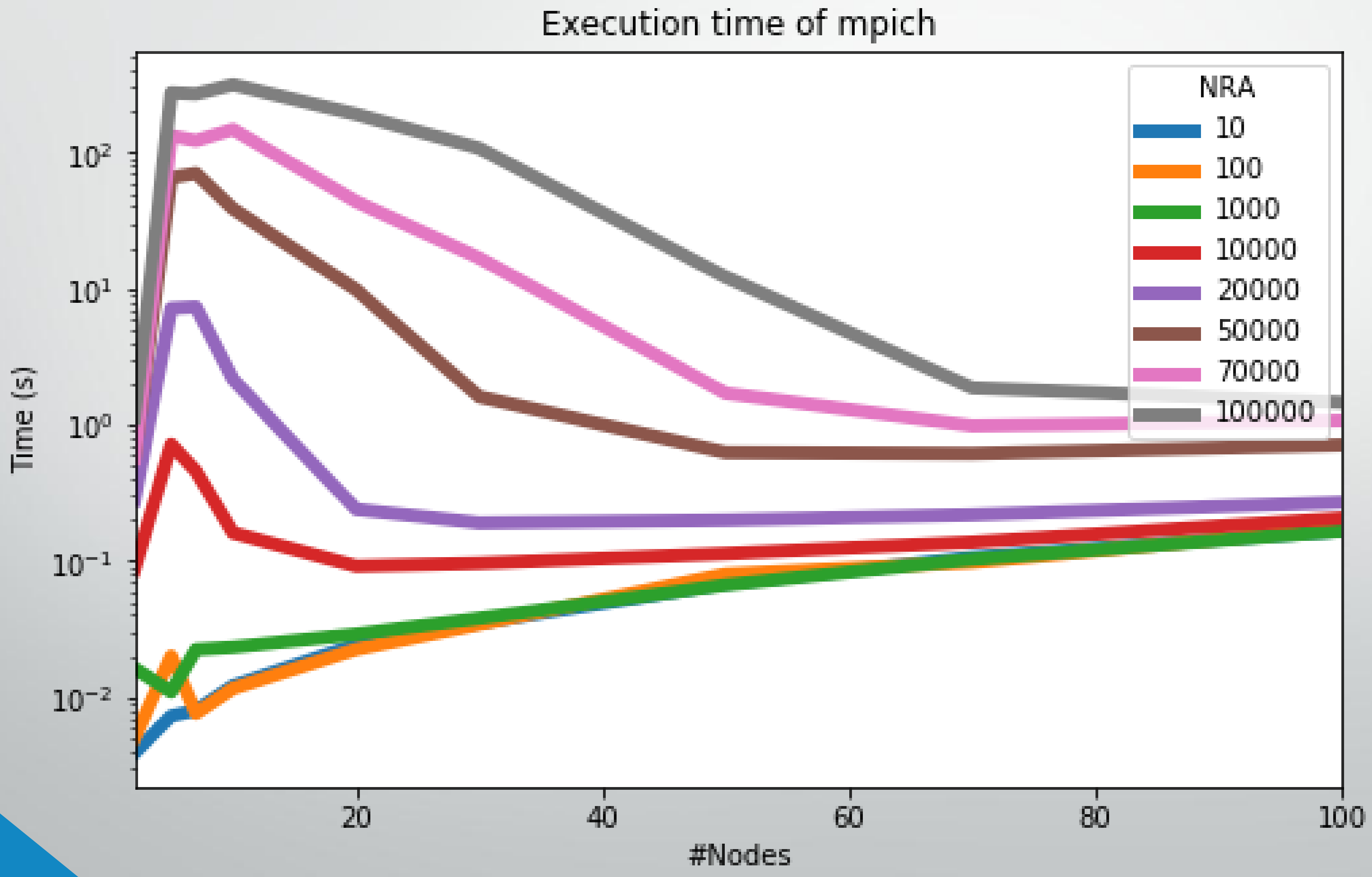
For master node from each worker node:

```
MPI_Recv(&offset, 1, MPI_INT, MASTER, FROM_MASTER, MPI_COMM_WORLD, &status);
MPI_Recv(&rows, 1, MPI_INT, MASTER, FROM_MASTER, MPI_COMM_WORLD, &status);
MPI_Recv(&a, rows*NCA, MPI_DOUBLE, MASTER, FROM_MASTER, MPI_COMM_WORLD, &status);
MPI_Recv(&b, NCA*NCB, MPI_DOUBLE, MASTER, FROM_MASTER, MPI_COMM_WORLD, &status);

<Calculation part>

MPI_Send(&offset, 1, MPI_INT, MASTER, FROM_WORKER, MPI_COMM_WORLD);
MPI_Send(&rows, 1, MPI_INT, MASTER, FROM_WORKER, MPI_COMM_WORLD);
MPI_Send(&c, rows*NCB, MPI_DOUBLE, MASTER, FROM_WORKER, MPI_COMM_WORLD);
```

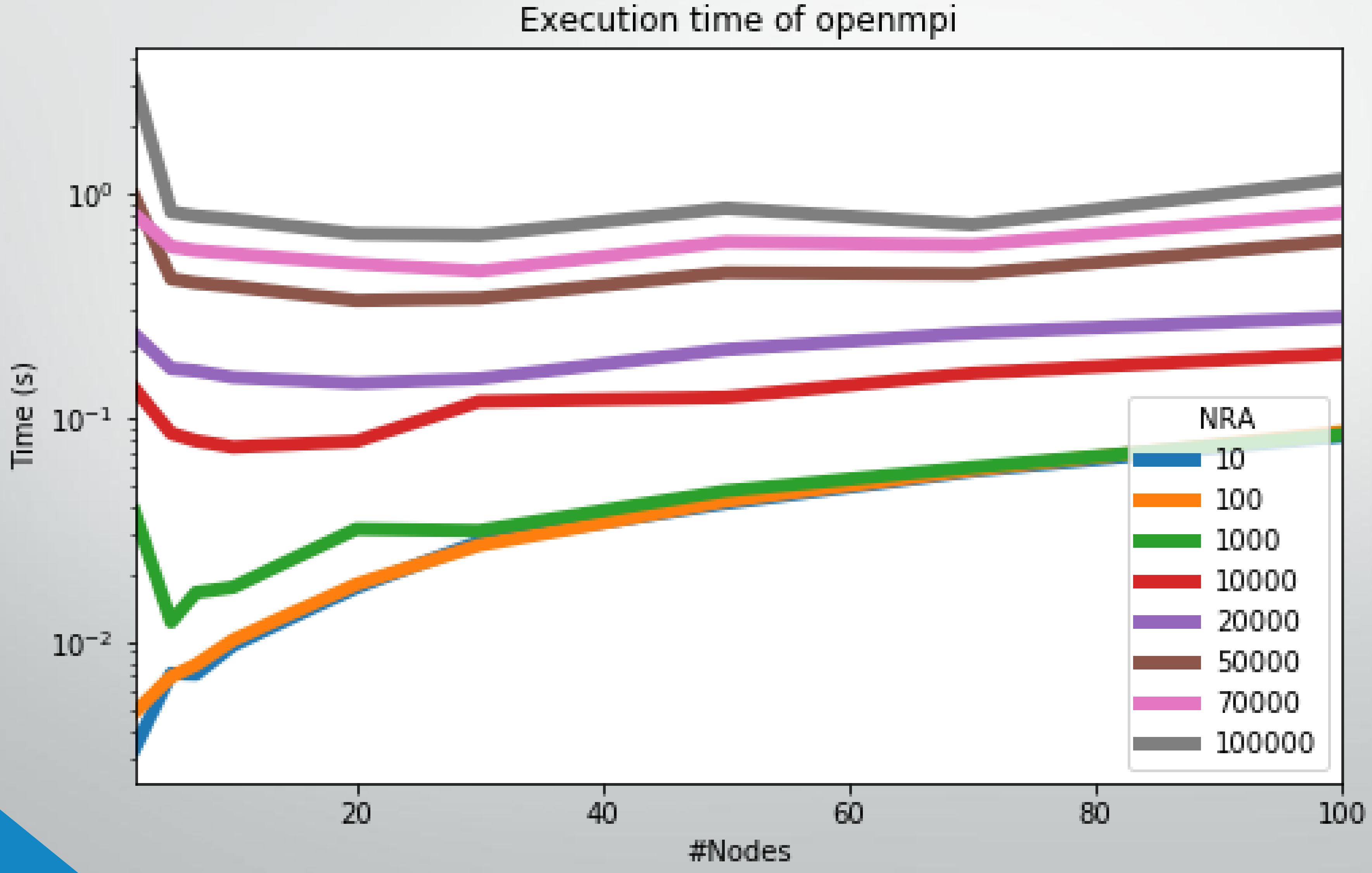
Test Case

- Matrix A x Matrix B
  Size matrix A: NRA x 10
  Size matrix B: 10 x 10
- Test case varies 3 variables:
  - NRA: 10,100,1000,10000,20000,50000,70000,100000
  - Number of Nodes: 2,5,7,10,20,30,50,70,100
  - MPI Variant: mpich, openmpi
- Execution time and memory consumption of each test case is captured

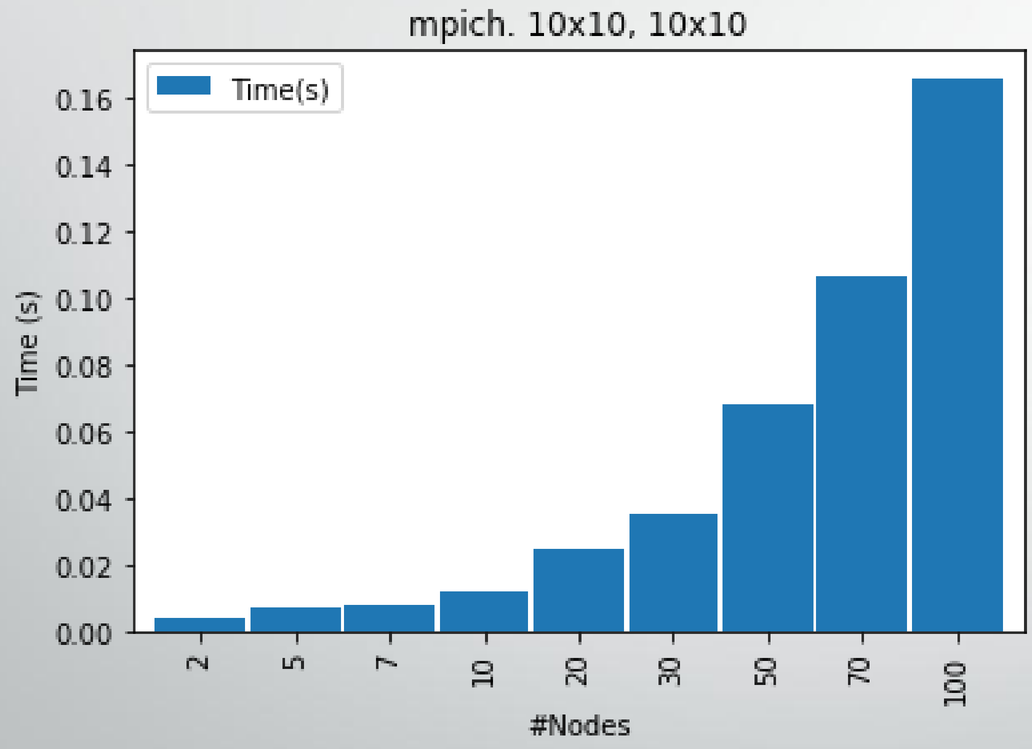
Execution time (log scaled)

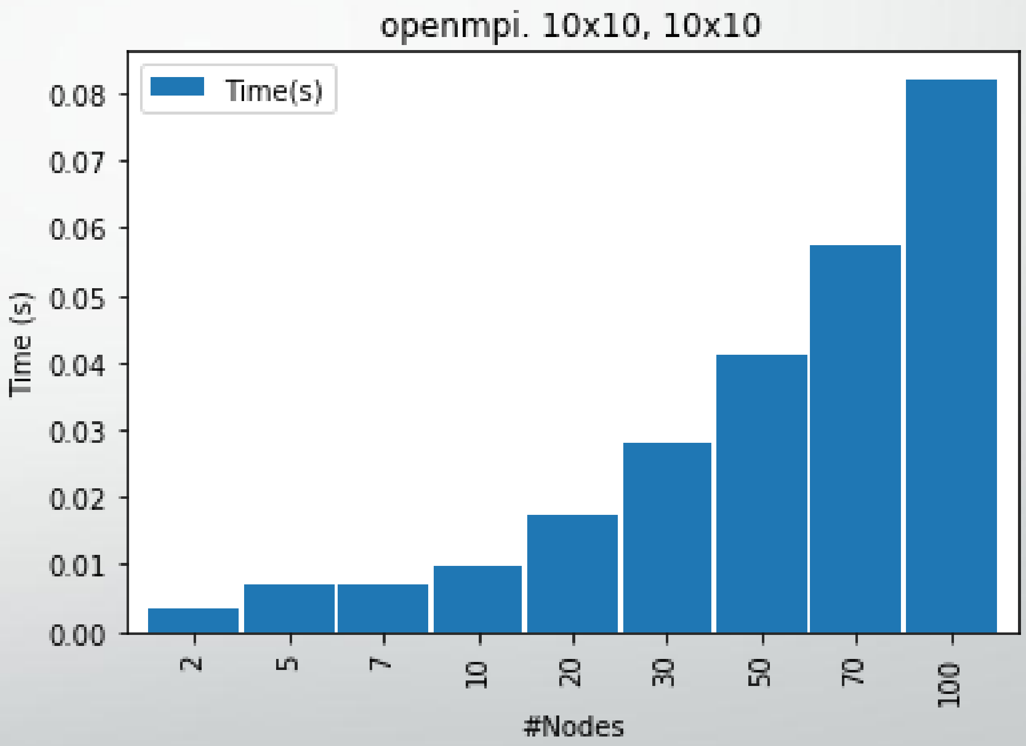


Execution time (log scaled)

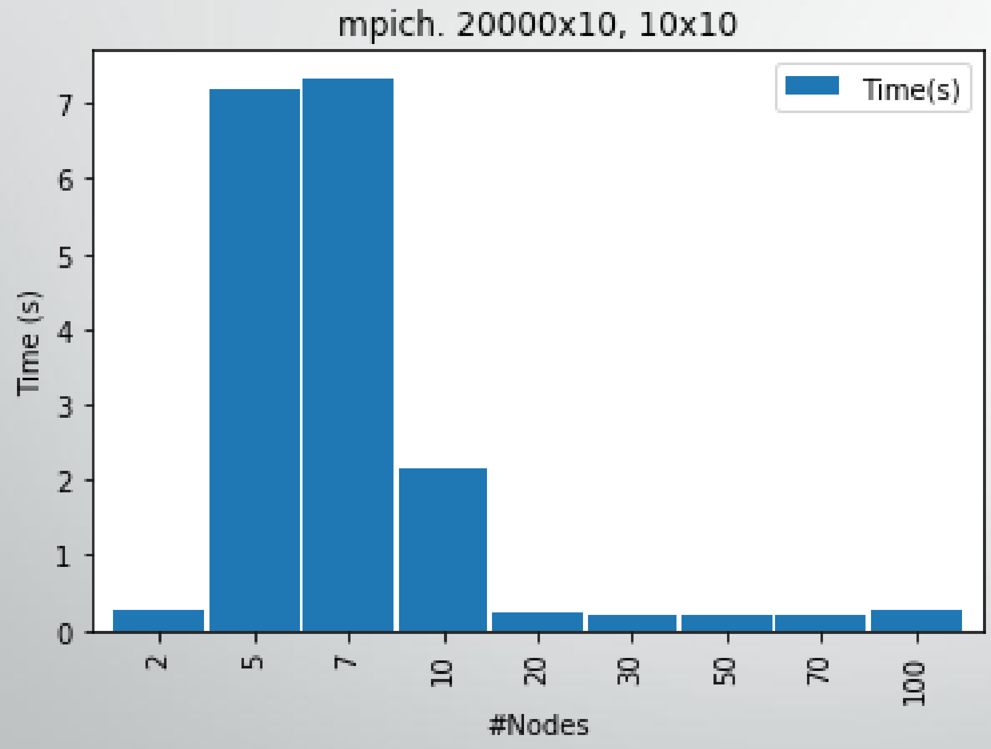


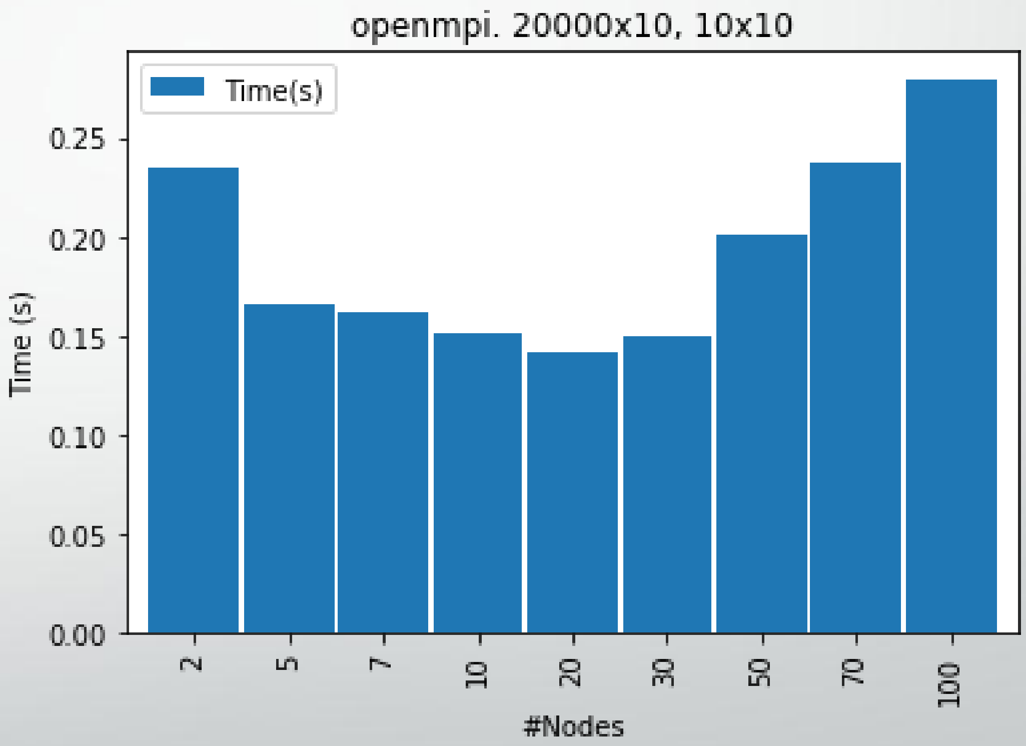
Execution time: Small problem size



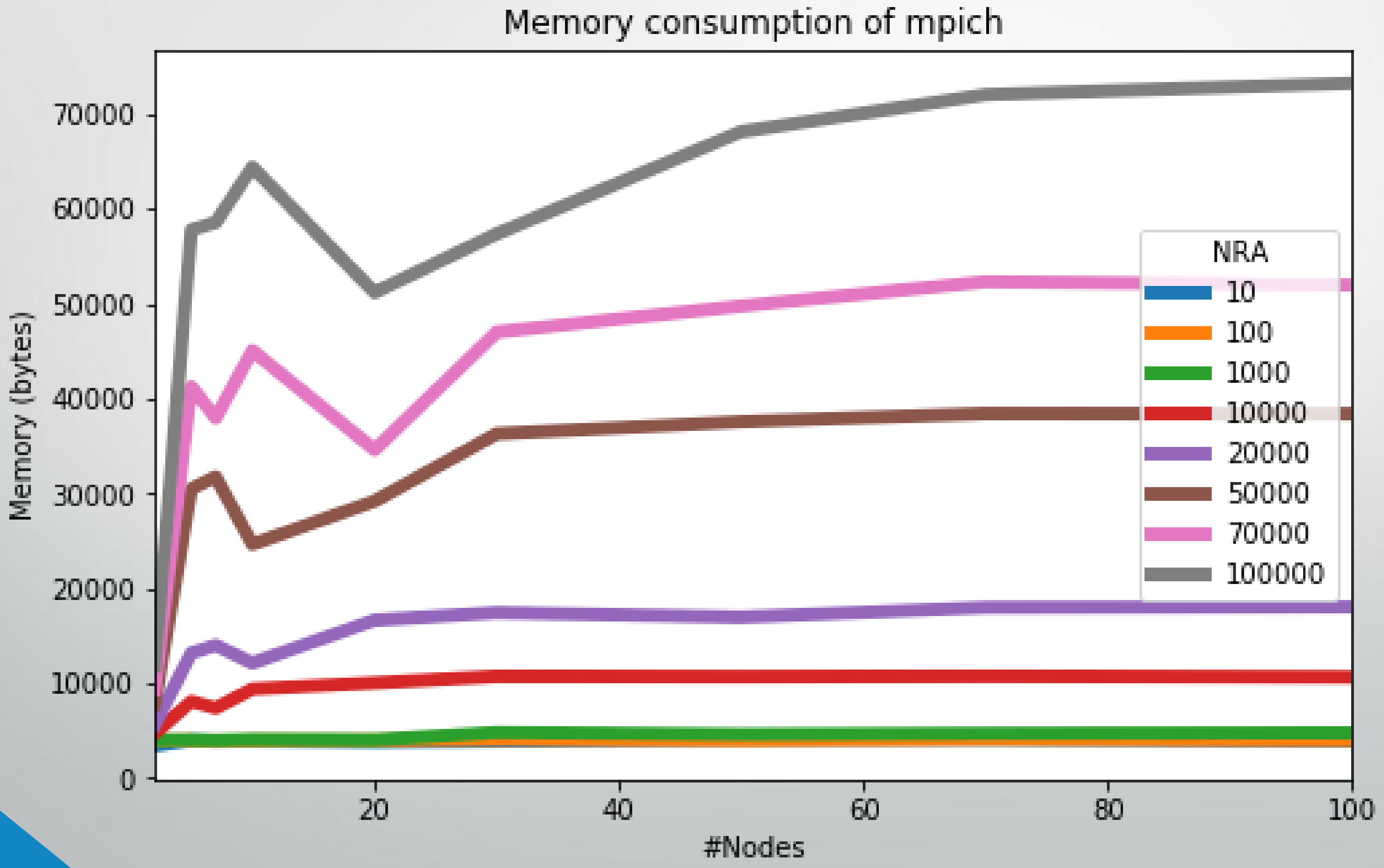


Execution time: Medium problem size

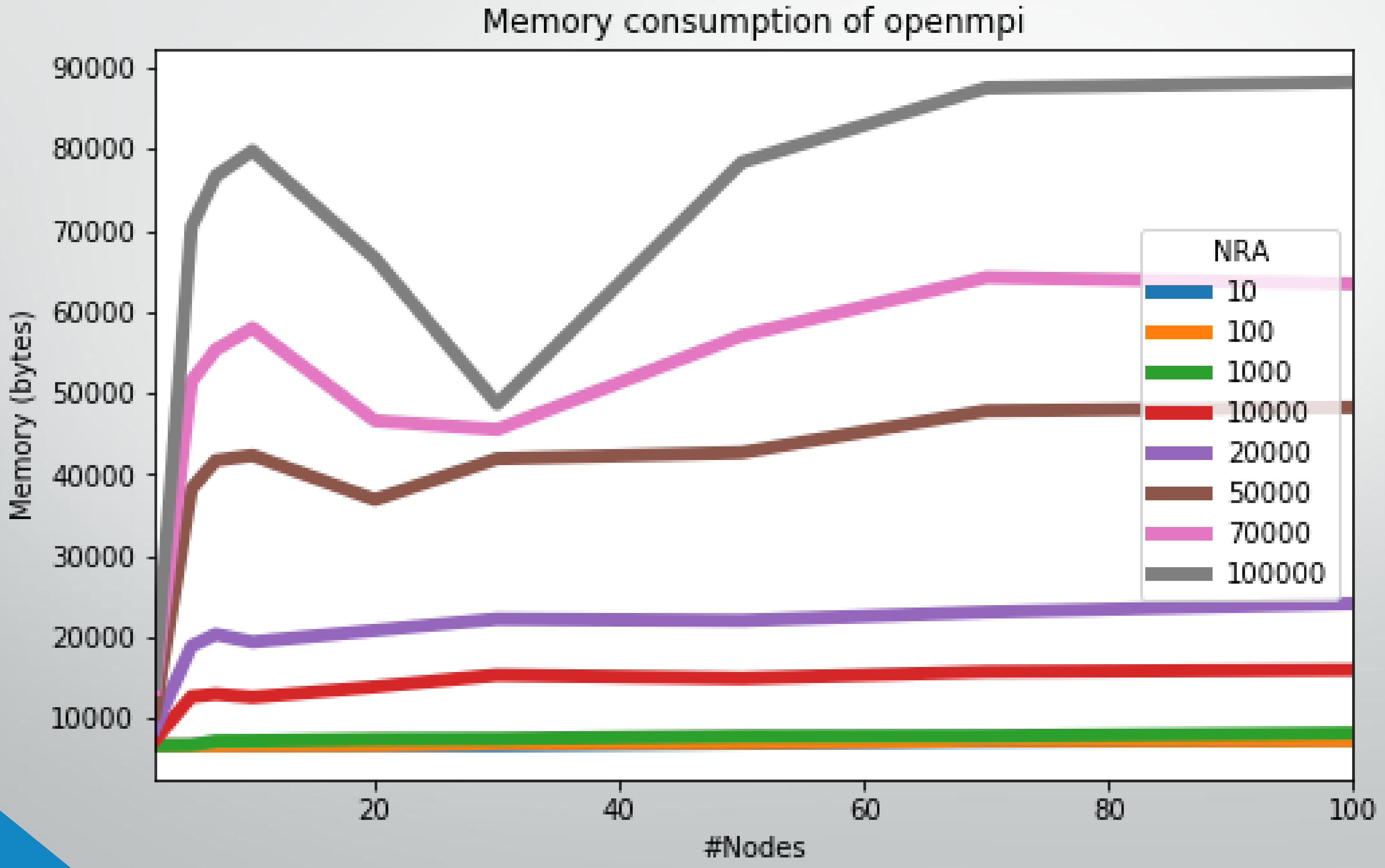

mpich. 20000x10, 10x10


openmpi. 20000x10, 10x10

Memory consumption



Memory consumption



Takeaway

- Same code, different MPI library produces different result
- OpenMPI is faster than mpich in general
- mpich is unexpectedly slow for nodes range: 5~10

  -> Results not useful

- Optimal number of nodes for case 20000 x 10, 10 x 10

  -> 20 nodes (OpenMPI)

Thank you for listening !