Parallel Programming with MPI

Use case - Jacobi solver

HPC SCHOOL
COSTA RICA

CINECA

RISC2

CeNAT
Centro Nacional de Alta Tecnología

ALESSANDRO MARANI

a.marani@cineca.it || San José 2023

Jacobi solver

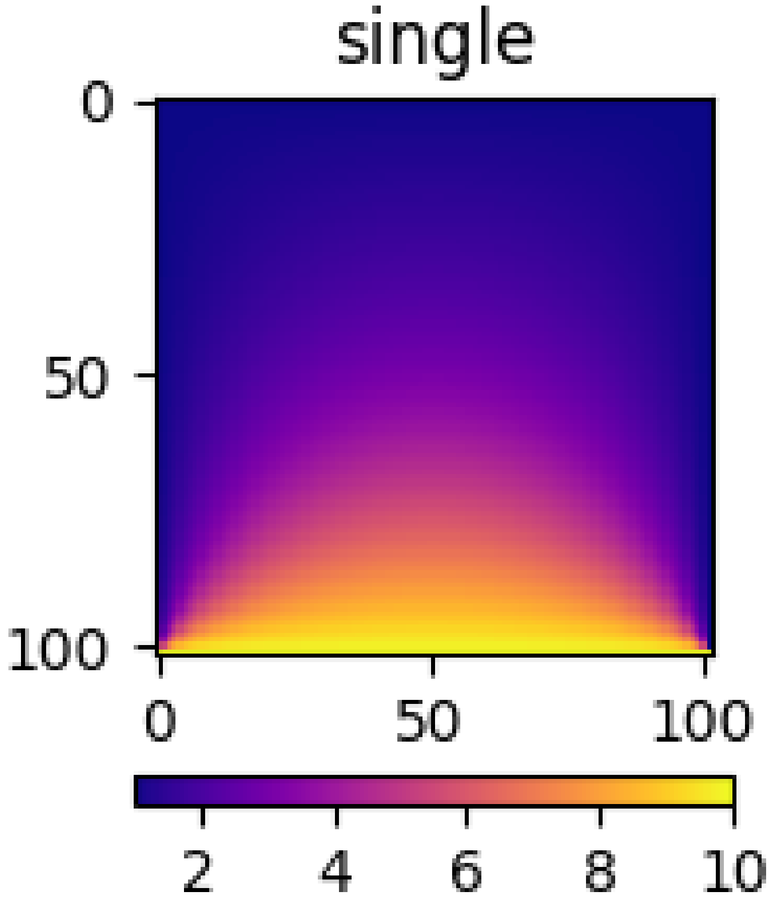
(also known as 2D stencil or Laplace 2D operator)

- The Jacobi method is an iterative algorithm for solving a system of linear equations.
- In the 2D model an approximation can be made by taking the average of the 4 neighbouring values (4 point stencil).

```
// Main loop
int iter;
for (iter=0; iter<MAX_ITER; iter++) {
    // Convergence check
    tmpnorm=0.0;
    for (i=1; i<=nx; i++) {
        for (j=1; j<=ny; j++) {
            k=(ny+2)*i+j;
           tmpnorm=tmpnorm+pow(grid[k]*4-grid[k-1]-
grid[k+1] - grid[k-(ny+2)] - grid[k+(ny+2)], 2);
        }
    }
    norm=sqrt(tmpnorm)/bnorm;
    if (norm < TOLERANCE) break;
    // Update new grid
    for (i=1; i<=nx; i++) {
        for (j=1; j<=ny; j++) {
            k=(ny+2)*i+j;
            grid_new[k]=0.25 * (grid[k-1]+grid[k+1]
+ grid[k-(ny+2)] + grid[k+(ny+2)]);
        }
    }
```

Practical use – 2D Heat Equation

Can be used as a simple model of heat flow.
For example, we can set the walls of the container to have different temperatures.

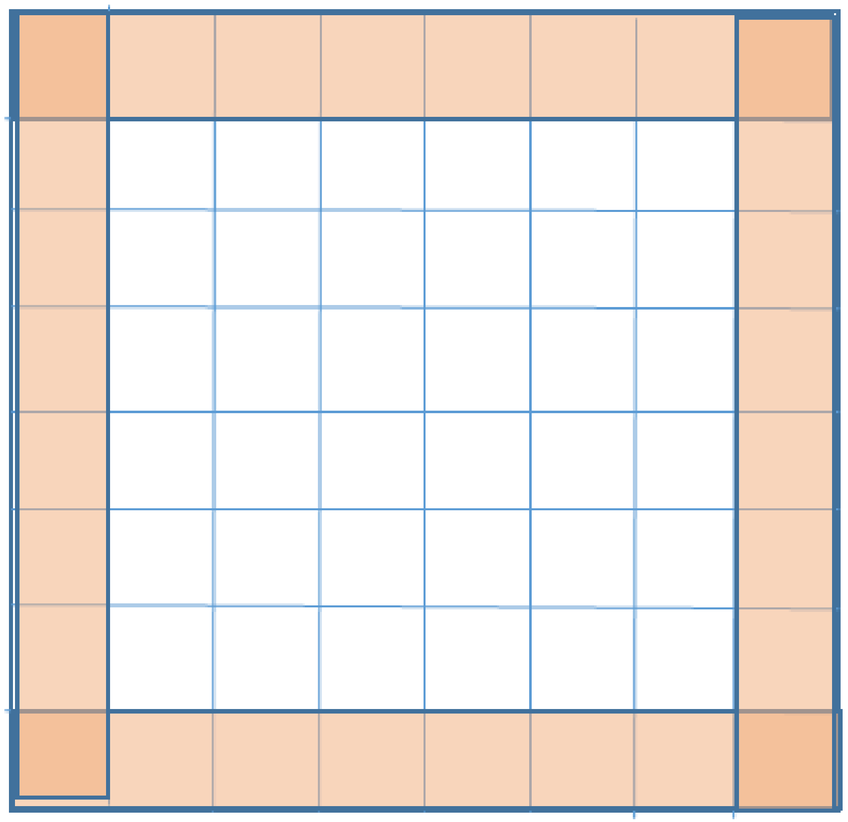


2D Jacobi solver on a 100x100 grid.

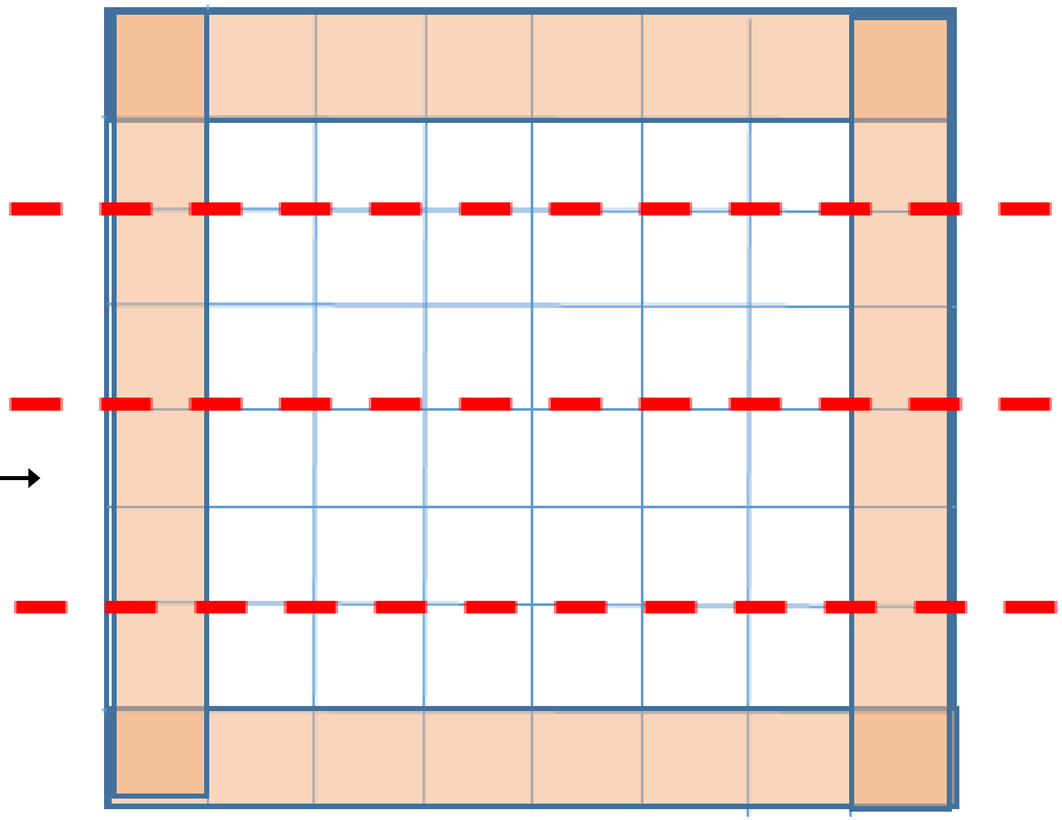
Jacobi in parallel

- For simplicity, we will use 1D domain decomposition, dividing rows or columns of the grid among the MPI ranks.
- Since each rank will need data from neighbouring domains need to set up halo regions.
- As a first version, can use MPI_Send and MPI_Recv to exchange the data.

1-D domain decomposition



divide grid amongst
available processors

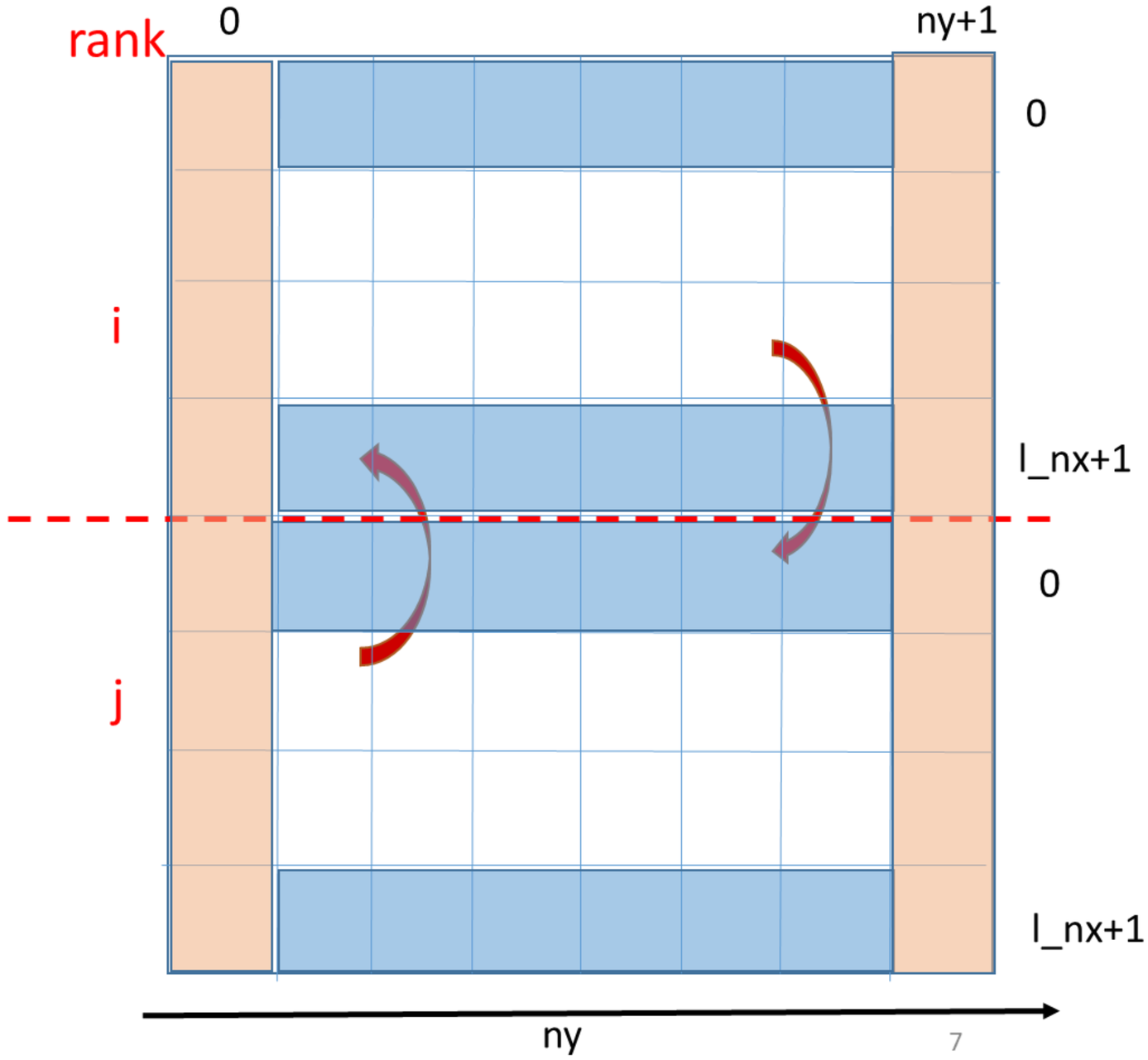


Halo exchange

We want to exchange rows, such that they are divided among the MPI ranks. Remember the MPI exercise about the ghost cells!

MPI ranks 0 and *size - 1* need only 1 halo region (and only 2 transfers) because they contain the top and bottom boundary conditions.

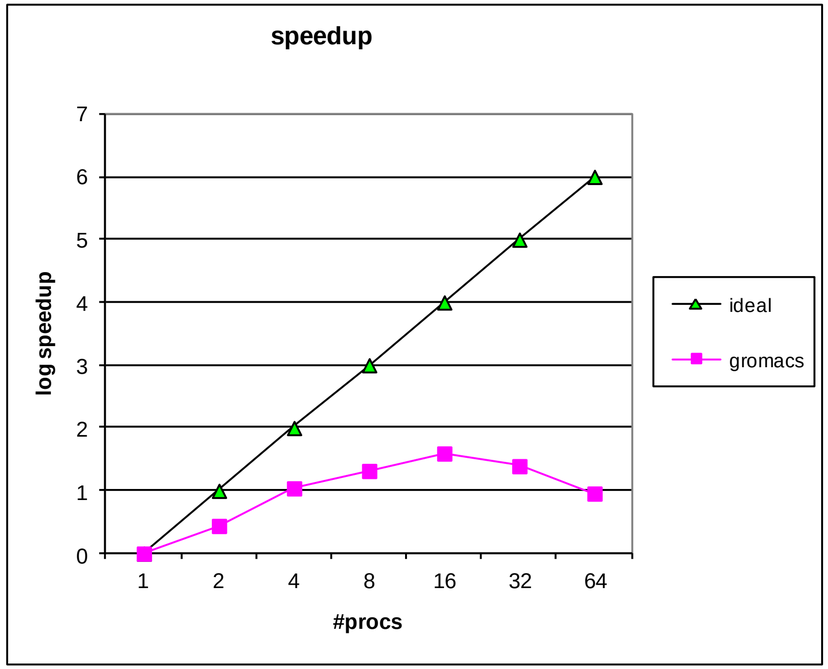
halo boundary grid

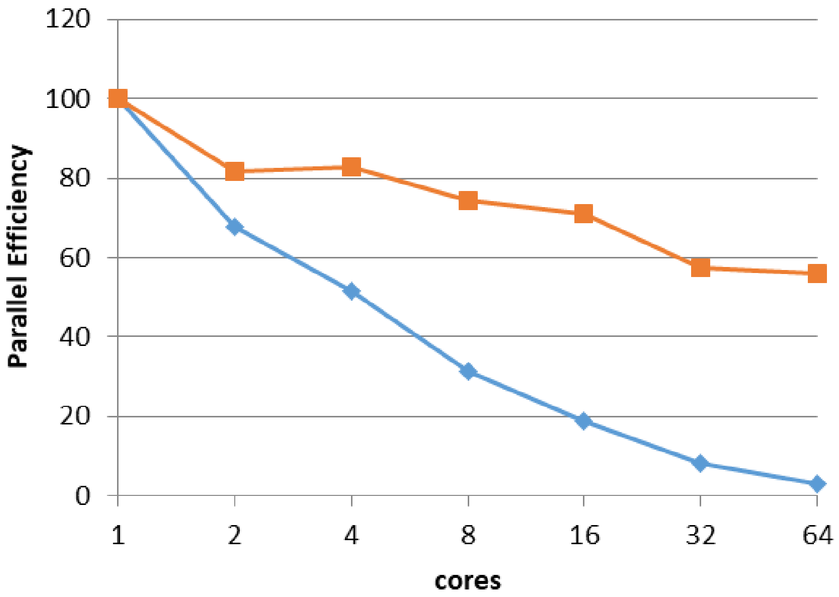


Parallel scaling

- An important feature of parallel programs is the parallel scaling, i.e. how the performance changes as the number of processors (or processor cores) is varied.
- To test the parallelization it is usual to plot a scaling curve showing this variation, adding also the “ideal” scaling, i.e. based on the assumption that if the number of cores double so does the performance.
- It is also possible to plot the parallel efficiency, defined as:

$$S = 100 \times \frac{P_N}{N \times P_1}$$





Exercise – MPI and scaling

1. Try first the serial code, for various grid sizes. ./jacobi_serial 100 100
2. Choose a grid size (for example 200x200) and run with 1,2,4,8, .. MPI processes.
3. Calculate the parallel efficiency depending on the number of processors.
4. (optional) Replace the blocking send/recv with non-blocking send/recv ot with MPI_Sendrecv and re-run. Check the results.

Exercise – OpenMP and Hybrid

1. Have a look at the source code and answer the following questions:
   - Which loop or loops would you try to parallelize with OpenMP?
   - Which OpenMP C pragmas would you use to do the parallelization?

Try to parallelize the serial code on the correct loops with OpenMP and check the results. They have to be *similar* to the serial version.

2. This code would be a good choice for hybrid MPI+OpenMP: use MPI and domain decomposition to distribute data over shared memory nodes, but use OpenMP to parallize with threads the loops over local data. Try that on the MPI version of the code and run it with different number of tasks and threads.

Current MPI Standard to be found at: https://www.mpi-forum.org/docs/mpi-4.0/mpi40-report.pdf
Credits to A. Emerson and many other colleagues from CINECA for the slides

To compile

Serial code:

```
gcc jacobi-serial.c -o <your_exec_name> -lm
```

MPI code*:

```
mpicc jacobi-mpi.c -o <your_exec_name> -lm
```

OpenMP code:

```
gcc <your_omp_code>.c -o <your_exec_name> -lm -fopenmp
```

Hybrid code*:

```
mpicc <your_hyb_code>.c -o <your_exec_name> -lm -fopenmp
```

*: *Remember to source environment.sh from the Environment directory before compiling with mpicc*

Job submission

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH -N 1
#SBATCH --ntasks-per-node=4    ###Edit that to decide the number of MPI tasks of your job
#SBATCH --cpus-per-task=4      ###Edit that to decide the number of OpenMP threads for any task
#SBATCH -p nu-wide
#SBATCH -t 1:00:00

#export OMP_PLACES=cores
#export OMP_NUM_THREADS=4
time mpirun -n 4 ./jacobi-mpi.x 300 300
```

Uncomment them if you are running the hybrid solution

```
real    0m32.206s
user    0m0.034s
sys     0m0.026s
```

For serial and OpenMP codes you can work on login node. Use «time» as well to compare the times!