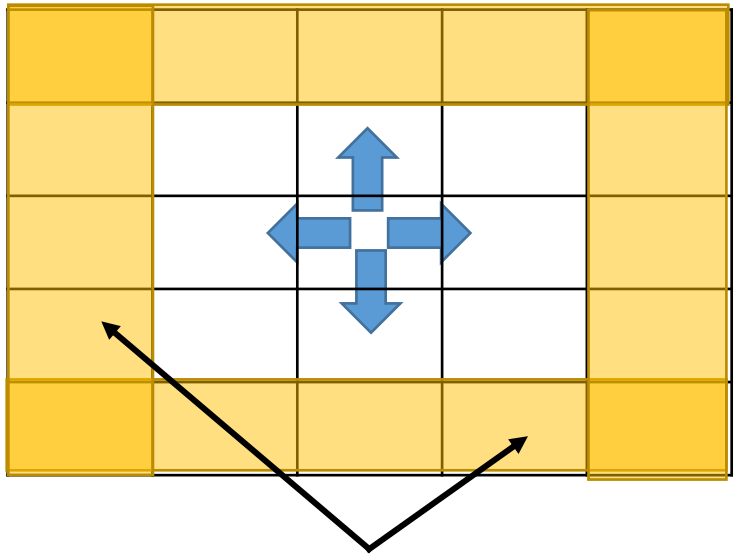
Jacobi MPI –example

Also known as 2D stencil or Laplace 2D operator

Jacobi Solver

- 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).



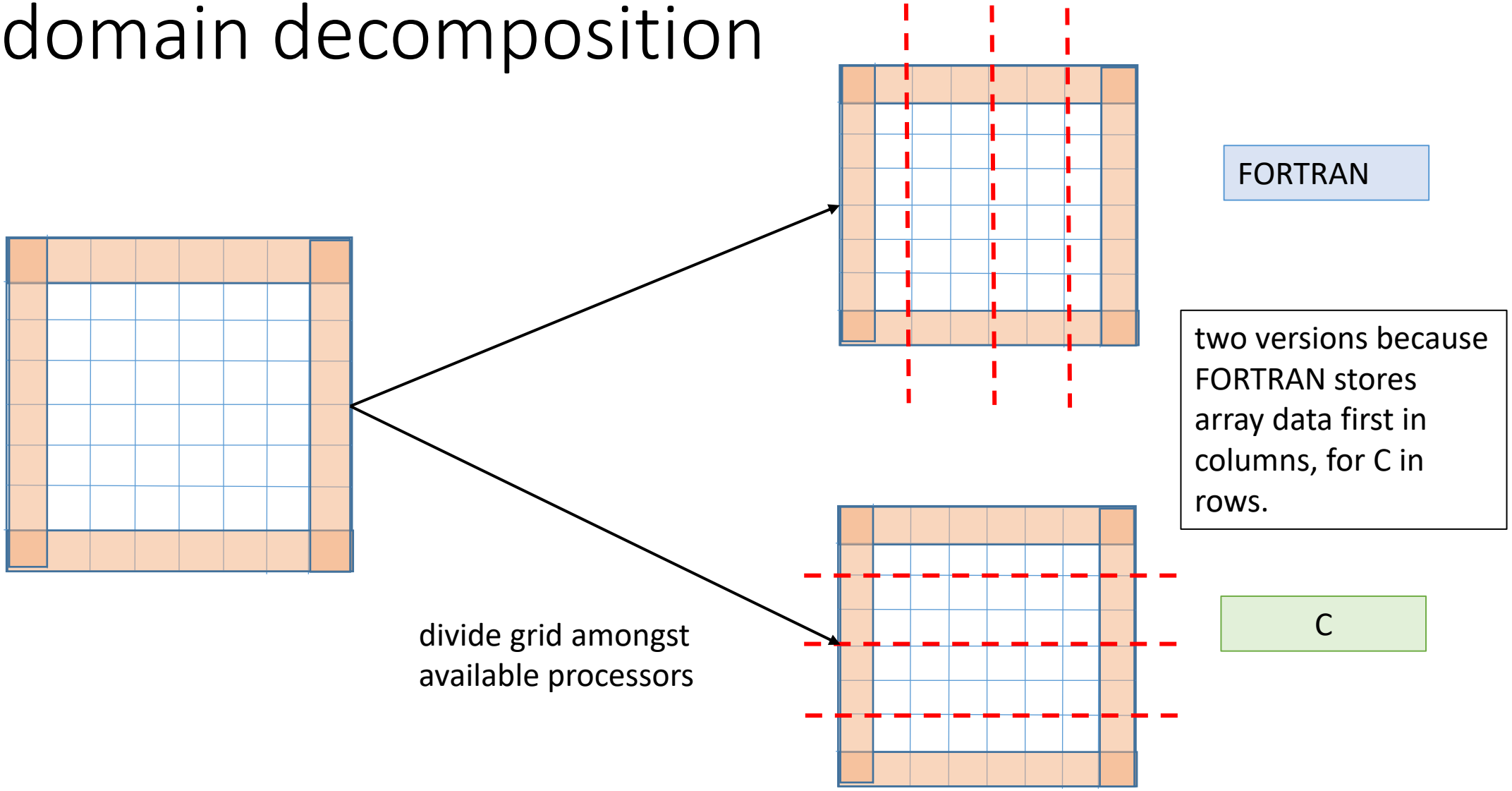

```
REAL A(0:n+1,0:n+1), B(1:n,1:n)
...
!Main Loop
DO WHILE(.NOT.converged)
   ! perform 4 point stencil
   DO j=1, n
      DO i=1, n
         B(i,j)=0.25*(A(i-1,j)+A(i+1,j)+A(i,j-
1)+A(i,j+1))
      END DO
   END DO

   ! copy result back into array A
   DO j=1, n
      DO i=1, n
         A(i,j) = B(i,j)
      END DO
   END DO
...
! convergence test
END DO
```

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.
- For efficiency, exchange columns with Fortran and rows with C.
- As a first version, can use MPI_Send and MPI_Recv to exchange the data.

1D-domain decomposition



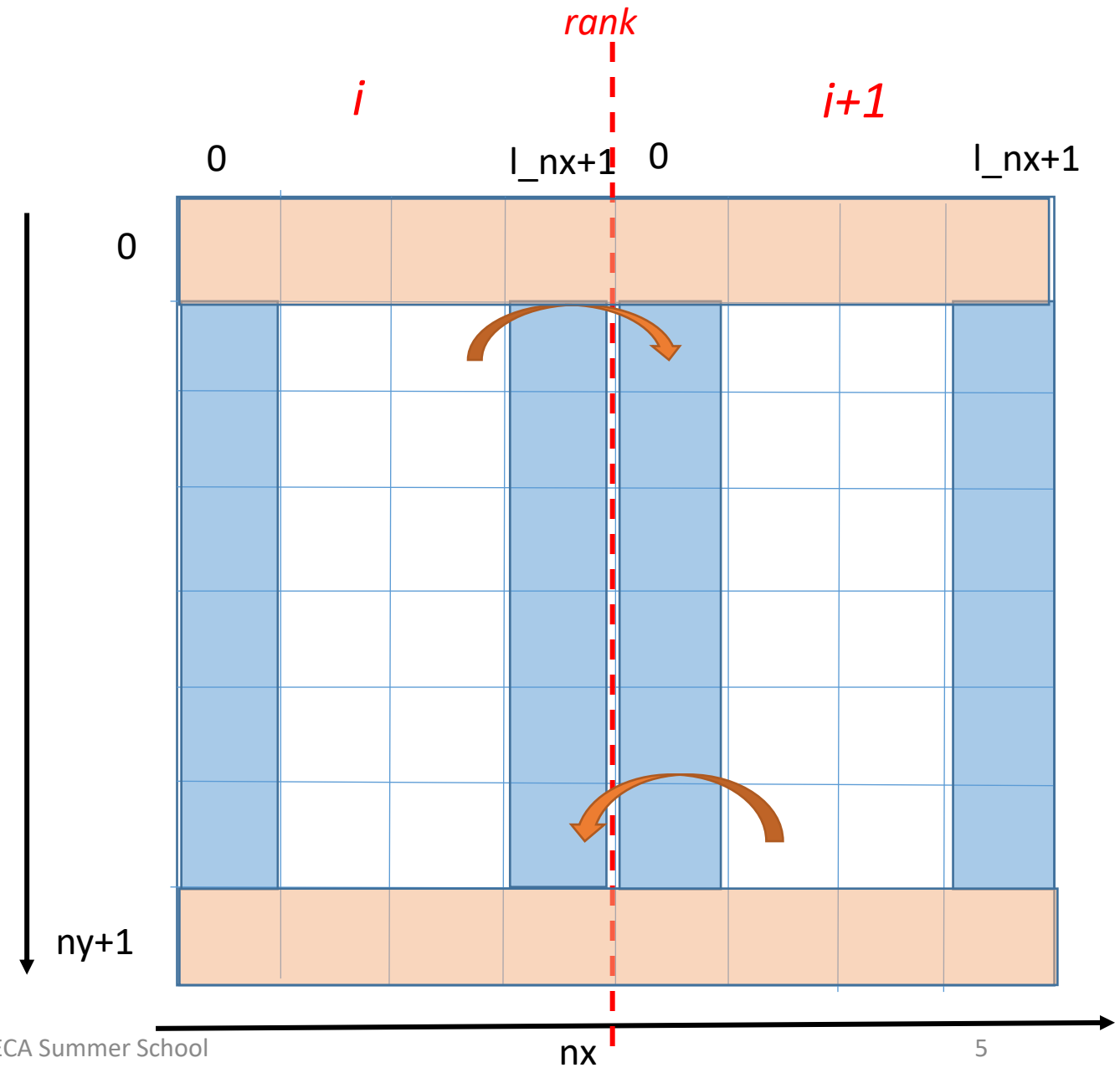
Halo exchange -Fortran Version

We start with a data grid(ny,nx) which becomes grid(0:ny+1,0:nx+1) when we include the boundaries.

Each local domain is (1..ny,1..l_{nx}) but with a halo region + boundaries we have

(0..ny+1,0..l_{nx}+1).

Ranks 0 and size-1 need only 1 halo region, since they must store the left and right border conditions.



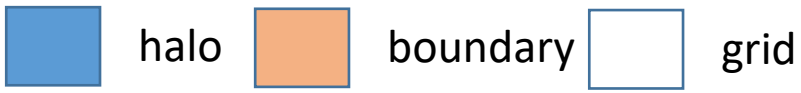
halo boundary grid

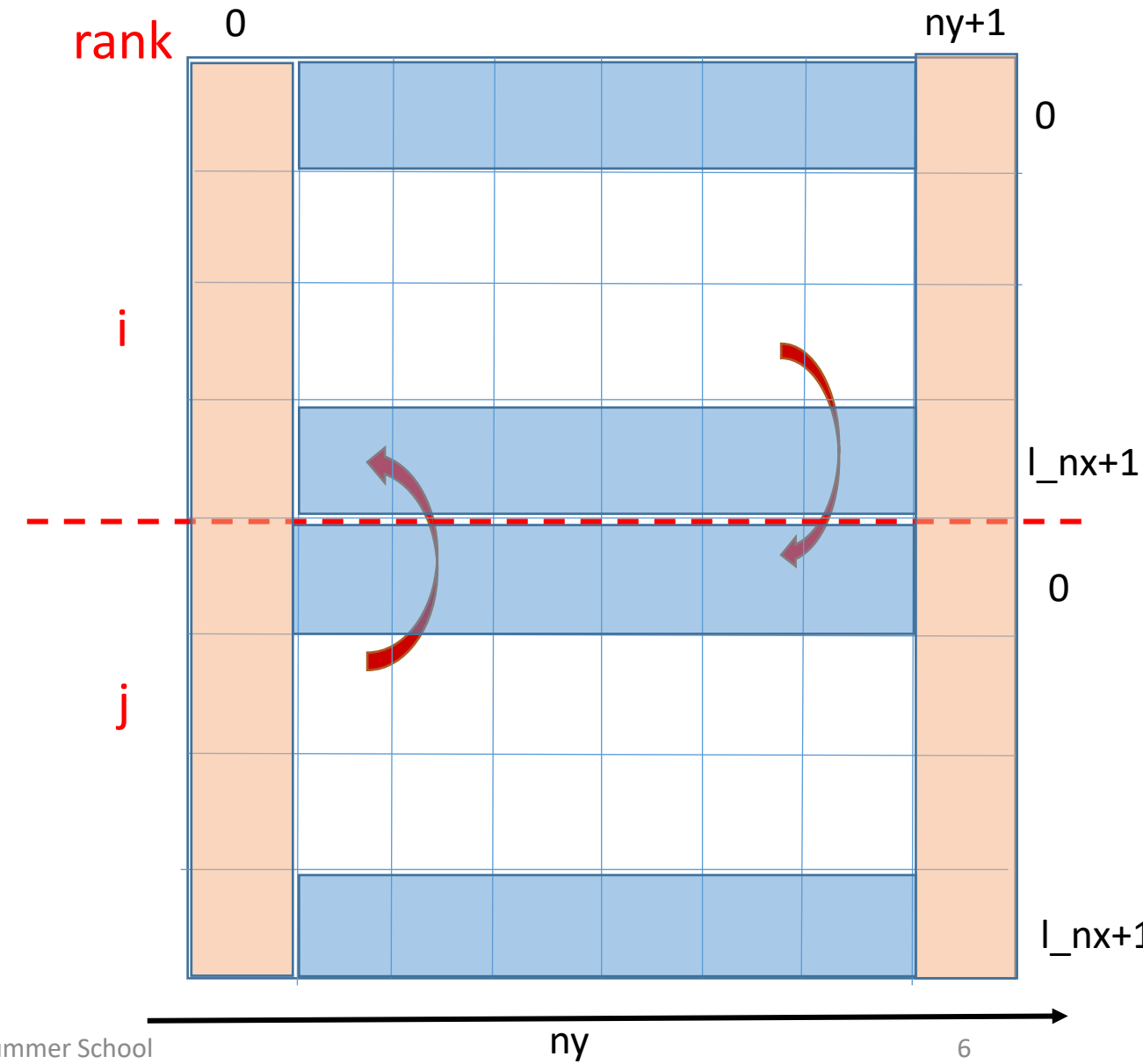
Halo exchange C Version

For C we exchange rows instead of columns, such that the rows are divided among the MPI ranks.

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

Note we have inverted the nx and ny axes with respect to the FORTRAN version.





Practical

1. Download the MPI C or Fortran source code:
   - git clone https://gitlab.hpc.cineca.it/training/mpi-openmp.git
2. Try first the serial code, for various grid sizes.

   ./jacobi_serial 100 100
3. Look at the MPI code - this is missing the MPI commands needed to transfer the data to the halo regions. Try first using blocking mpi sends or mpi recvs to perform the transfer.
4. Replace the blocking send/recv with non-blocking send/recv and re-run. (optional: mpi_sendrecv). Check the results.
5. Alternatively use the solution to test the parallel scaling as a function of MPI tasks and grid sizes.

Inspired by the example at http://www.archer.ac.uk/training/course-material/2016/09/160929_AdvMPI_EPCC/index.php

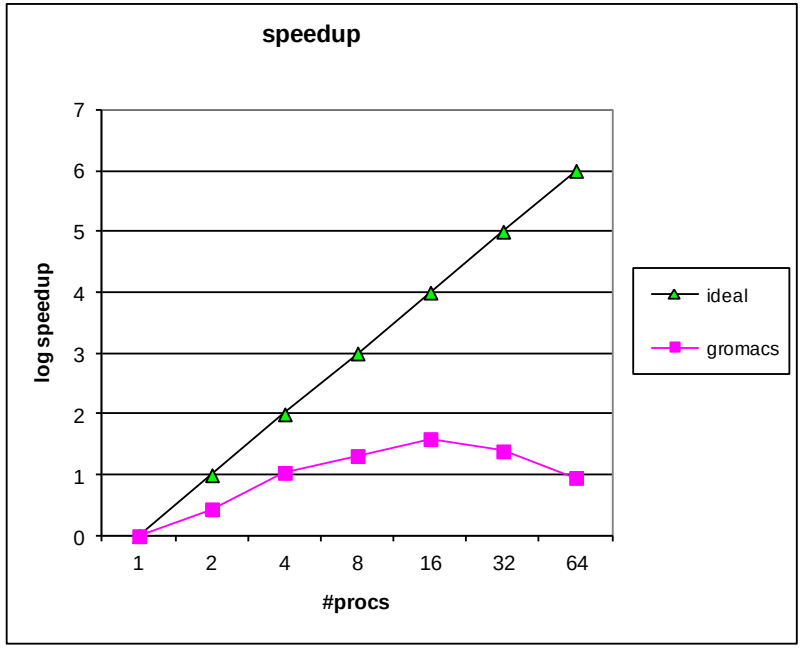
Hints

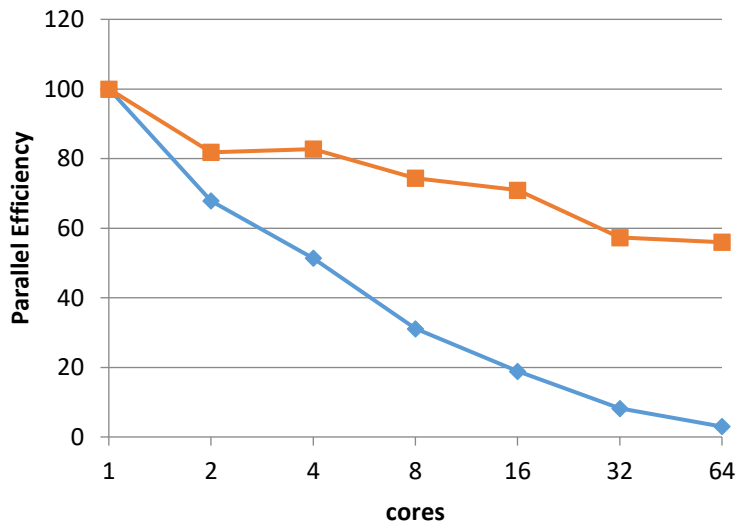
- For C :
  - mpi_send(&grid[1][1],..) to send the first data row
  - mpi_send(&grid[lx][1],) to send the last data row
  - mpi_recv(&grid[0][1],..) to receive into the upper halo region
  - mpi_recv(&grid[lx+1][1],..) to receive into the lower halo region
- For Fortran
  - mpi_send(grid(1,1),..) to send the first data column
  - mpi_send(grid(1,lx),..) to send the last data column
  - mpi_recv(grid(1,0),..) to receive into the left halo column
  - mpi_recv(grid(1,lx+1),..) to receive into the right halo column

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:

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





Scaling Exercises

1. With a working program(!) choose a grid size (200x200) and run with 1,2,4,8, .. MPI processes.
2. Plot the scaling curve and also the parallel efficiency.
3. At what stage does the parallel efficiency drop below 50%? For the jacobi solver what causes the scaling behaviour?