# Foundations of High Performance Computing

Lecture 10: Performance model for jacobi solver

"**Foundation of HPC" course**

DATA SCIENCE & SCIENTIFIC COMPUTING

2021-2022 Stefano Cozzini

# Agenda

- Performance evaluation on Jacobi 3D solver

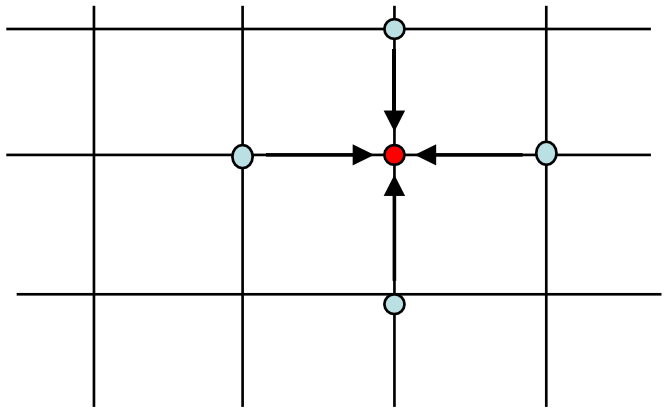
# Jacobi solver..

- prototype for many stencil-based iterative methods in numerical analysis and simulation
- Basic form: solve the diffusion equation for a scalar function: $\Phi(\boldsymbol{r}, t)$

$$\frac{\partial \Phi}{\partial t} = \Delta \Phi \ ,$$

# Straightforward 2D serial implementation: the stencil

```
do k = 1,kmax
 do i = 1,imax
  phi(i,k,t1) = 0.25 * phi(i+1,k,t0) + 0.25 * phi(i-1,k,t0)
              + 0.25 * phi(i,k+1,t0) + 0.25 * phi(i,k-1,t0)
  enddo
 enddo
```



All taken from reference 4

# The full serial algorithm..

- Ensure that the code produces a converged result.

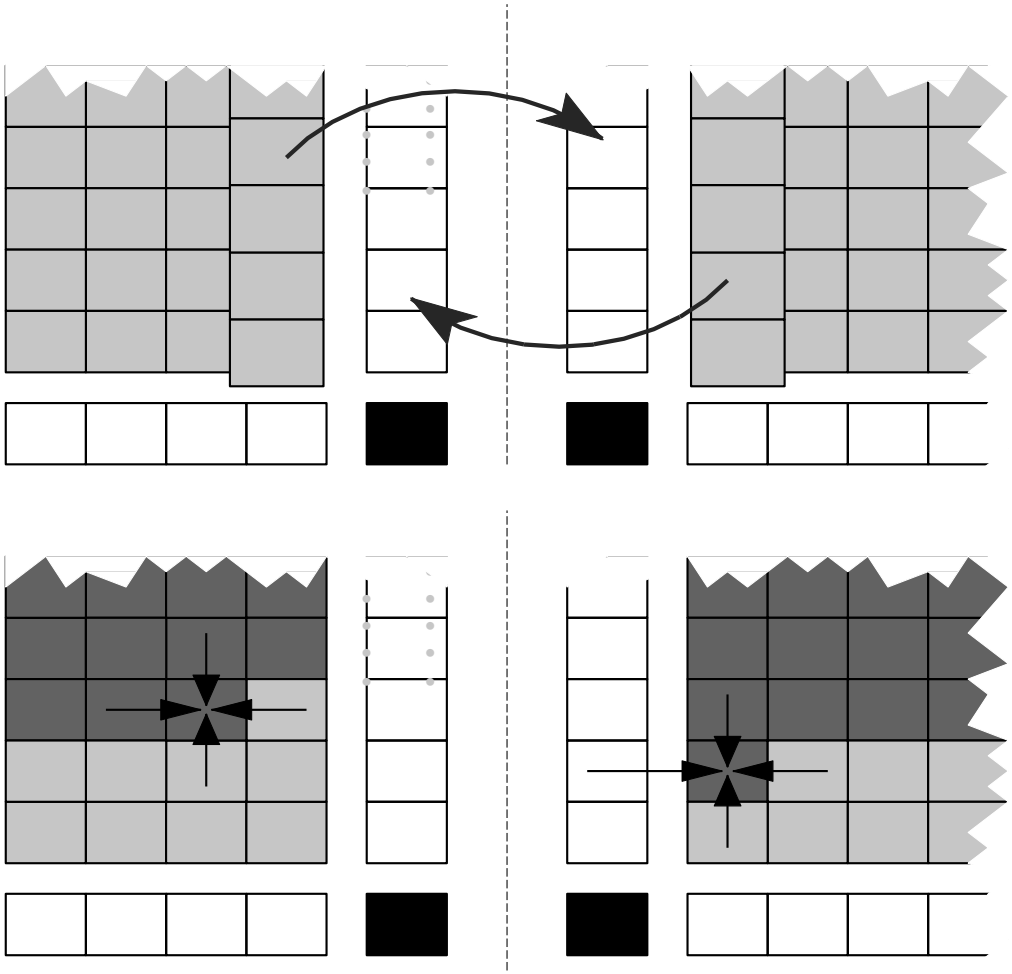
```
double precision, dimension(0:imax+1,0:kmax+1,0:1) :: phi
double precision :: maxdelta,eps
 integer :: t0,t1
 eps = 1.d-14 ! convergence threshold
 t0 = 0 ; t1 = 1
 maxdelta = 2.d0*eps
 do while(maxdelta.gt.eps)
 maxdelta = 0.d0
do k = 1,kmax
   do i = 1,imax
          phi(i,k,t1) = 0.25 * phi(i+1,k,t0) + 0.25 * phi(i-1,k,t0)
                      + 0.25 * phi(i,k+1,t0) + 0.25 * phi(i,k-1,t0)
          maxdelta = max(maxdelta,abs(phi(i,k,t1)-phi(i,k,t0)))

  enddo
      enddo
! swap arrays
i = t0 ; t0=t1 ; t1=i
 enddo
```

# What happens in the parallel implementation ?

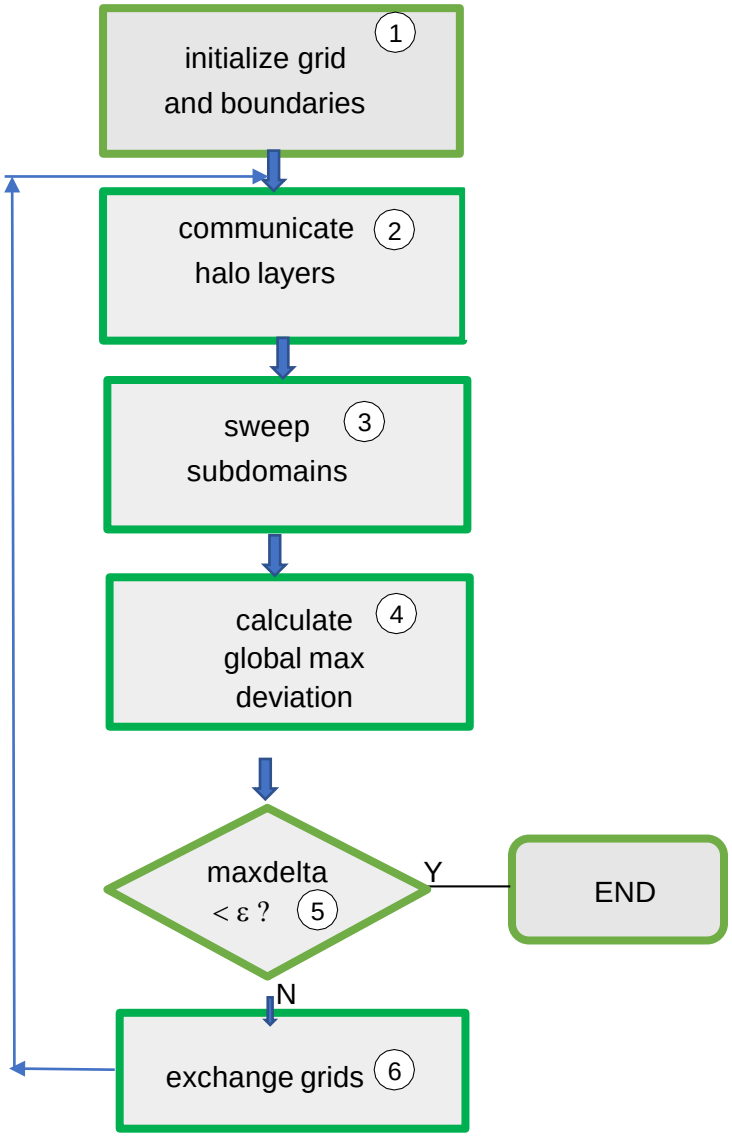
- The computational core works the same but:
  - Convergence criterion is no longer enough on subdomains but need to be computed globally

    ➔ requires a reduce operation among all processors
  - We need to take care of boundary conditions : Cell close to border require special care and require halo layers

# The parallel data distribution..



# The parallel algorithm



# A few remarks

- Initialization is done with virtual topology

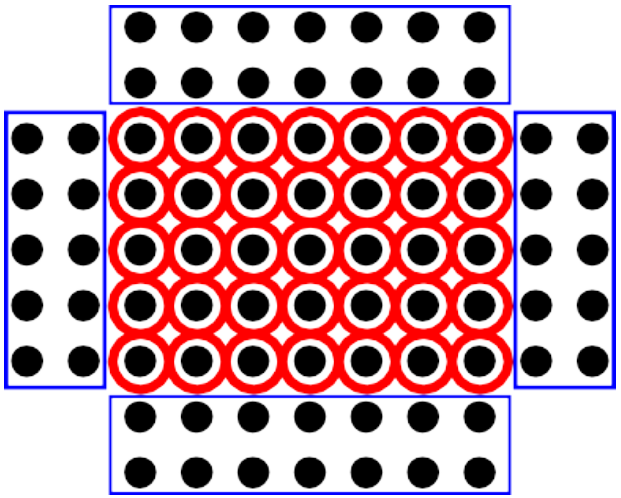
```
1  call MPI_Dims_create(numprocs, 3, proc_dim, ierr)
2
3  if(myid.eq.0) write(*,'(a,3(i3,x))') 'Grid: ', &
4      (proc_dim(i),i=1,3)
5
6  l_reorder = .true.
7  call MPI_Cart_create(MPI_COMM_WORLD, 3, proc_dim, pbc_check, &
8      l_reorder, GRID_COMM_WORLD, ierr)
9
10 if(GRID_COMM_WORLD .eq. MPI_COMM_NULL) goto 999
11
12 call MPI_Comm_rank(GRID_COMM_WORLD, myid_grid, ierr)
13 call MPI_Comm_size(GRID_COMM_WORLD, nump_grid, ierr)
```

# Halo communication

- We use point to point communication to exchange halo layer.
- Point-to-point communication requires consecutive message buffers.
- Do we have contiguous location in memory for the halo ?

# Not at all…

- only those boundary cells that are consecutive in the inner (i) dimension are also consecutive in memory (fortran column-major order..)
- Whole layers in the i-j, i-k, and j-k planes are never consecutive

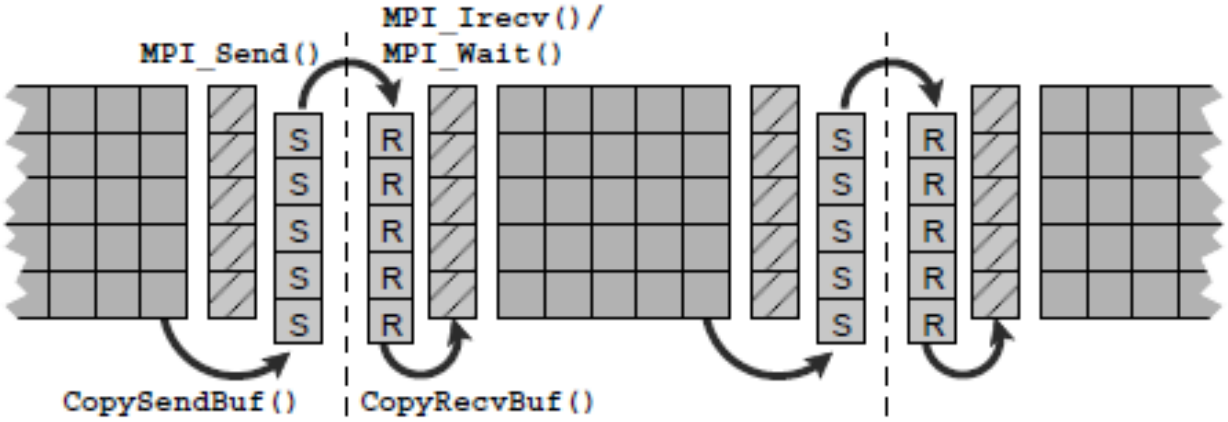


# How to deal with this issue ?

intermediate buffer must be used to gather boundary data to be communicated to a neighbor’s ghost layer.

# Halo exchange in one direction..

- We use two intermediate buffers per process, one for sending and one for receiving.



# Dimension of the buffer…

- halo data can be different along different Cartesian directions

➔ the size of the intermediate buffer must be chosen to accommodate the largest possible halo

```
integer, dimension(1:3) :: totmsgsize

! j-k plane
totmsgsize(3) = loca_dim(1)*loca_dim(2)
MaxBufLen=max(MaxBufLen,totmsgsize(3))
! i-k plane
totmsgsize(2) = loca_dim(1)*loca_dim(3)
MaxBufLen=max(MaxBufLen,totmsgsize(2))
! i-j plane
totmsgsize(1) = loca_dim(2)*loca_dim(3)
MaxBufLen=max(MaxBufLen,totmsgsize(1))

allocate(fieldSend(1:MaxBufLen))
allocate(fieldRecv(1:MaxBufLen))
```

# Halo exchange.

```
  do disp = -1, 1, 2
    do dir = 1, 3

      call MPI_Cart_shift(GRID_COMM_WORLD, (dir-1), &
                          disp, source, dest, ierr)

      if(source /= MPI_PROC_NULL) then
        call MPI_Irecv(fieldRecv(1), totmsgsize(dir), &
             MPI_DOUBLE_PRECISION, source, &
             tag, GRID_COMM_WORLD, req(1), ierr)
      endif    ! source exists

      if(dest /= MPI_PROC_NULL) then
        call CopySendBuf(phi(iStart, jStart, kStart, t0), &
                         iStart, iEnd, jStart, jEnd, kStart, kEnd, &
                         disp, dir, fieldSend, MaxBufLen)

        call MPI_Send(fieldSend(1), totmsgsize(dir), &
                      MPI_DOUBLE_PRECISION, dest, tag, &
                      GRID_COMM_WORLD, ierr)
      endif    ! destination exists

      if(source /= MPI_PROC_NULL) then
        call MPI_Wait(req, status, ierr)

        call CopyRecvBuf(phi(iStart, jStart, kStart, t0), &
                         iStart, iEnd, jStart, jEnd, kStart, kEnd, &
                         disp, dir, fieldRecv, MaxBufLen)
      endif    ! source exists

    enddo    ! dir
  enddo    ! disp
```

# A full 3D implementation

- Input

```
if(myid.eq.0) then

   write(*,*) " spat_dim , proc_dim, PBC ? "
   do i=1,n_dim
      read(*,*) spat_dim(i) , proc_dim(i), pbc_check(i)
      write(*,*) i,"-Dim Input ",spat_dim(i) , proc_dim(i), pbc_check(i)
   enddo

    pbc_check(1) = .false.
    pbc_check(2) = .false.
    pbc_check(3) = .false.

endif
```

# Analysis of the Jacobi Solver…

- Performance depends on L grid point over a $N=N_xN_yN_z$ processors
- where:
  - $T_s(L)$ sweep part: sequential time
  - $T_c(L,N)$: communication time

$$P(L,\vec{N}) = \frac{L^3 N}{T_s(L) + T_c(L,\vec{N})} \ ,$$

# Jacobi: computing time

- Cost of sweep domain:
  - The subdomain size ($L^3$) is the same regardless of the number of processes, so the raw compute time $T_s$ for all cell updates in a Jacobi sweep is also constant

→ Ts= 22.07 seconds (for L=1200) on GPU node on ORFEO

# Jacobi: communication time: halo exchange

- Each process sends  each of the 6 of point to point communication at the same time.
- we need to consider bandwidth number for full-duplex data transfer over a single link.
- Communication time $T_c$ depends on the number and size of domain cuts that lead to internode communication.
- Assumption:
  - copying to/from intermediate buffers and communication of a process with itself come at no cost.

# Communication time:

$$T_c(L,\vec{N}) = \frac{c(L,\vec{N})}{B} + kT_\ell \ .$$

- c(L,N): amount of data volume transferred over a node's network link
- B: bidirectional bandwidth of the network link
- $T_l$ = latency of the network
- k the largest number (over all domains) of coordinate directions in which the number of processes is greater than one.

→ $c(L,N)=L^2 * k * 2 * 8$ (Mb)

# Model prediction for ORFEO

- Take latency and bandwidth and use them into the formulas above.
- Check if what you are measuring is what you are expecting
- Clarify why…

# A possible model prediction for ORFEO infiniband..

N=1200

$T_l$ =1.36

B= 12000 Mb/seconds

| N | Nx | Ny | Nz | k | C(L,N) | Tc(L,N) | P(L,N) | P(1)*N/P(L,N) |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 0 | 0.00 | 78.30 | 1.00 |
| 2 | 2 | 1 | 1 | 2 | 46080 | 1.92 | 144.06 | 1.09 |
| 3 | 3 | 1 | 1 | 2 | 46080 | 1.92 | 216.09 | 1.09 |
| 4 | 2 | 2 | 1 | 4 | 92160 | 3.84 | 266.77 | 1.17 |
| 6 | 3 | 2 | 1 | 4 | 92160 | 3.84 | 400.15 | 1.17 |
| 8 | 2 | 2 | 2 | 6 | 138240 | 5.76 | 496.73 | 1.26 |
| 12 | 3 | 3 | 3 | 6 | 138240 | 5.76 | 745.10 | 1.26 |
| 16 | 4 | 2 | 2 | 6 | 138240 | 5.76 | 993.46 | 1.26 |