

Programming Techniques for Supercomputers:

Distributed memory parallel processing with MPI
Jacobi 3D – a MPI case study

Prof. Dr. G. Wellein^(a,b) , Dr. G. Hager^(a)

^(a)HPC Services – Regionales Rechenzentrum Erlangen

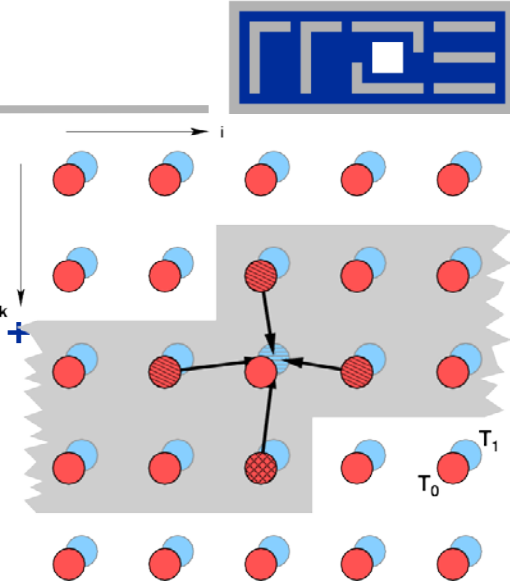
^(b)Department für Informatik

University Erlangen-Nürnberg, Sommersemester 2020

MPI parallelization

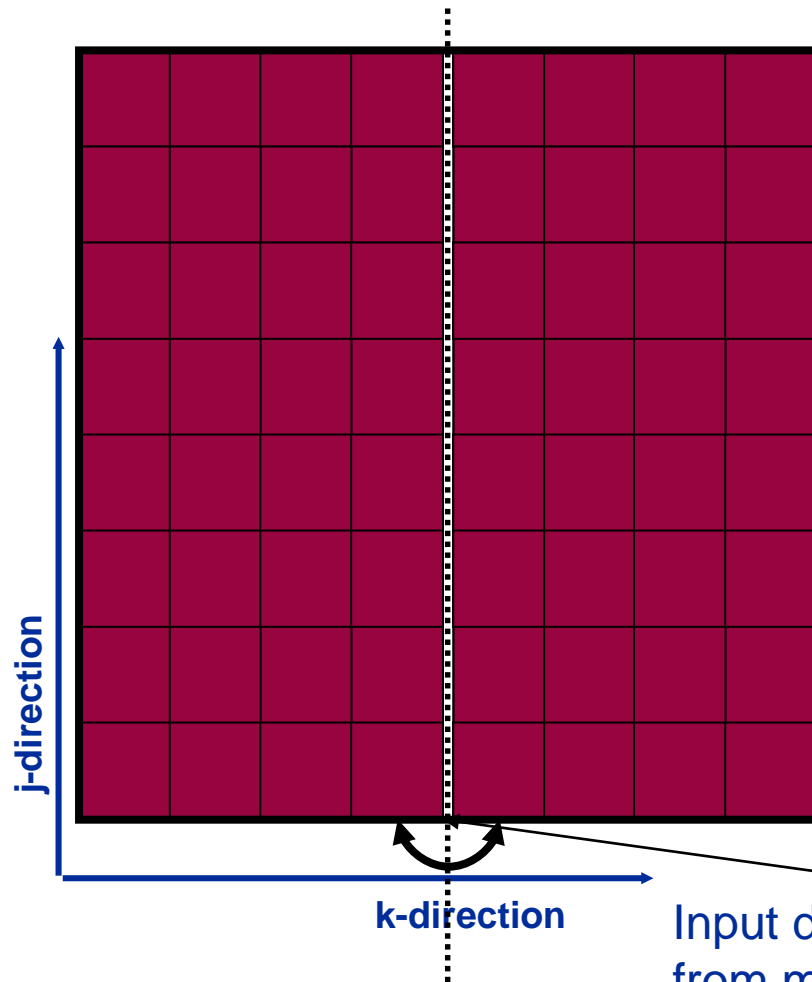
Jacobi solver – again a simple prototype

```
do k = 1 , Nk
  do j = 1 , Nj
    do i = 1 , Ni
      y(i,j,k) = b* (x(i-1,j,k)+x(i+1,j,k)+
                    x(i,j-1,k)+x(i,j+1,k)+
                    x(i,j,k-1)+x(i,j,k+1))
    enddo
  enddo
enddo
```



■ Using two arrays:

- Read input data from $\mathbf{x}(:, :, :)$; write new data to $\mathbf{y}(:, :, :)$
- Each entry of $\mathbf{y}(:, :, :)$ is updated only once in an outer iteration
- → All updates can be done in parallel – provided that input data from neighboring cells are available
- OpenMP/shared memory parallelization is straightforward; all threads can access complete input data in $\mathbf{x}(:, :, :)$



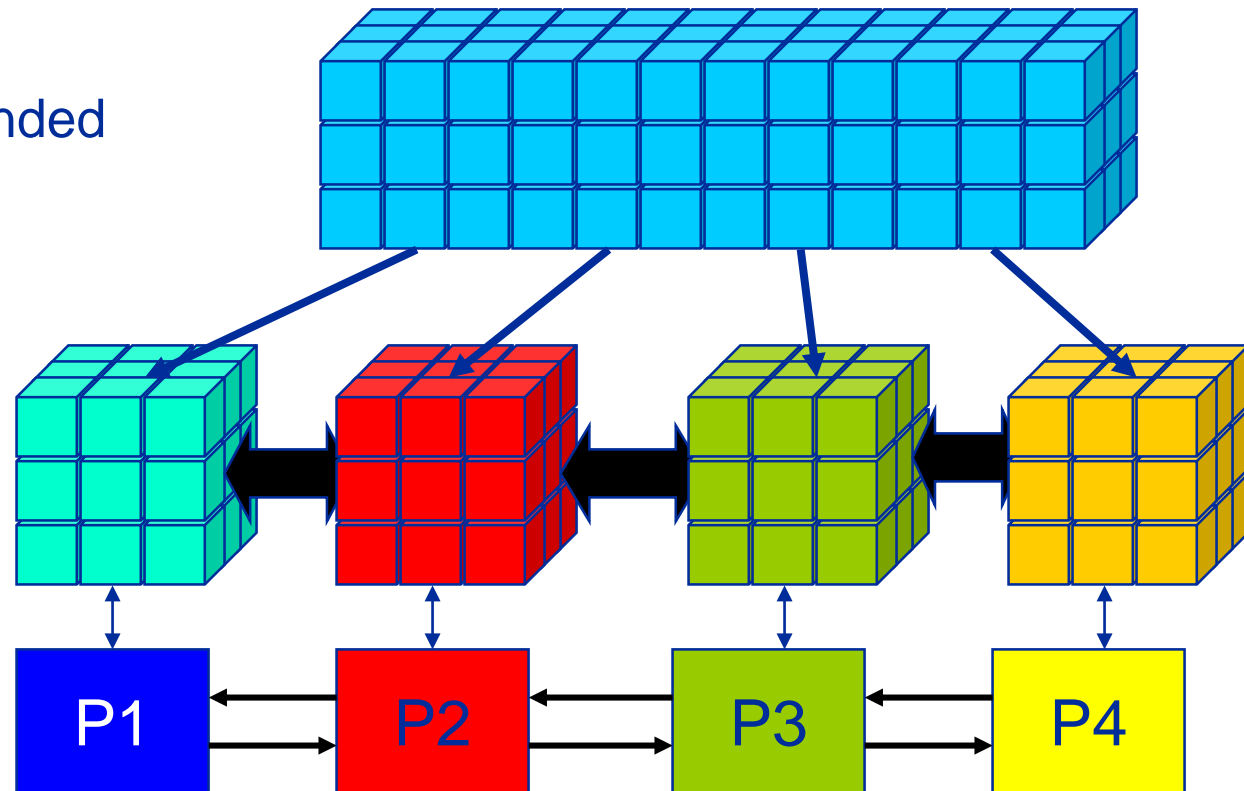
```
do t=1,tMax
!$OMP PARALLEL DO private(...)
  do k=1,Nk
    do j=1,Nj
      do i=1,Ni
        y(i,j,k) = ...
      enddo
    enddo
  enddo (IMPLICIT BARRIER)
  y ↔ x
enddo
```

Input data at domain boundaries are read from main memory – no intervention from programmer required



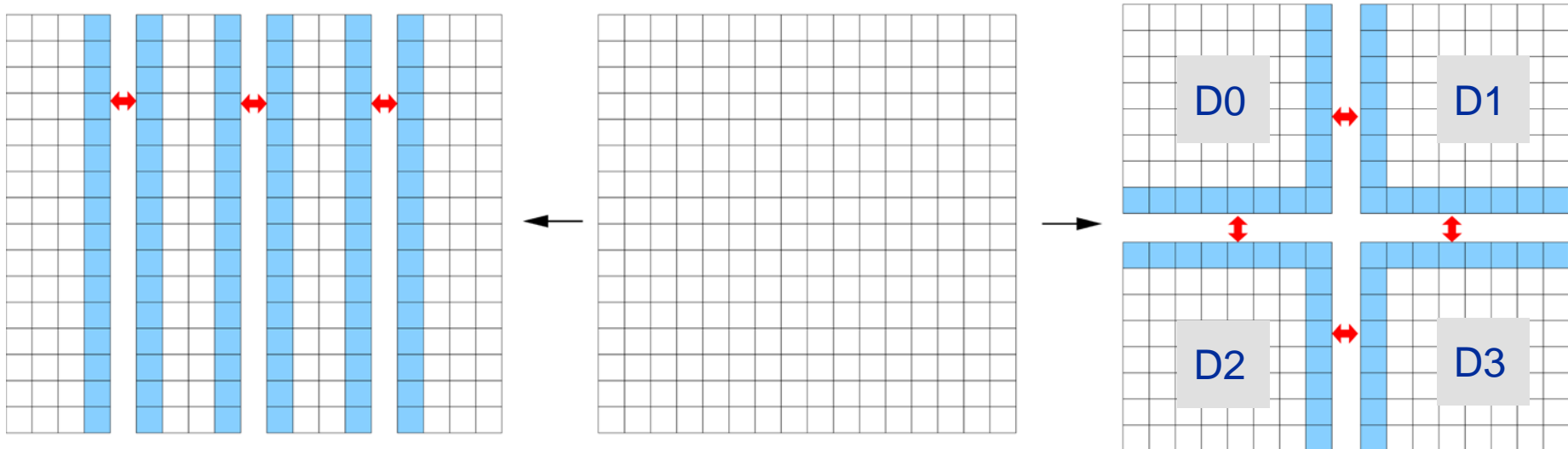
- Split up $12 \times 3 \times 3$ domain into 4 equally sized disjoint subdomains
- Each subdomain can be updated by a single MPI process, provided that the boundary cells from adjacent subdomains are available
- Boundary cells need to be exchanged between adjacent subdomains (i.e. MPI processes) after each outer iteration

- Subdomains are extended by halo-layers which hold boundary data from adjacent subdomains





- Choose a domain decomposition which ensures
 - Perfect load balancing ($O(L^3)$ effect)
 - Then** try to minimize communication overhead ($O(L^2)$ effect)

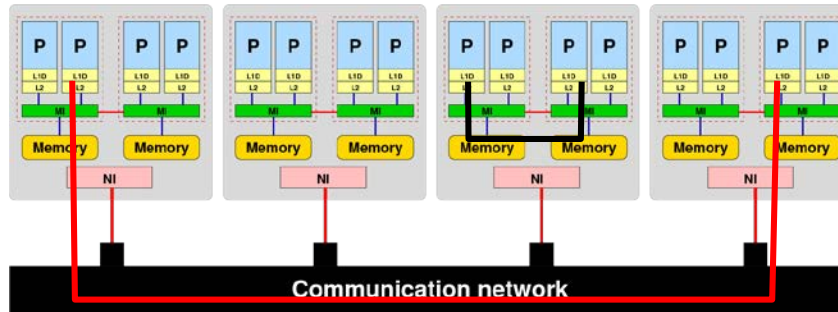


- But how to map domains (D0,...D3) to multicore nodes????**
- E.g. 4 processes (rank=0,...,3) on 2 nodes with 2 cores
 - D0,D1@node=0 & D2,D3@node=1 → 2 internode communications
 - D0,D3@node=0 & D1,D2@node=1 → 4 internode communications

→ **How to control MPI rank mapping to hardware?**



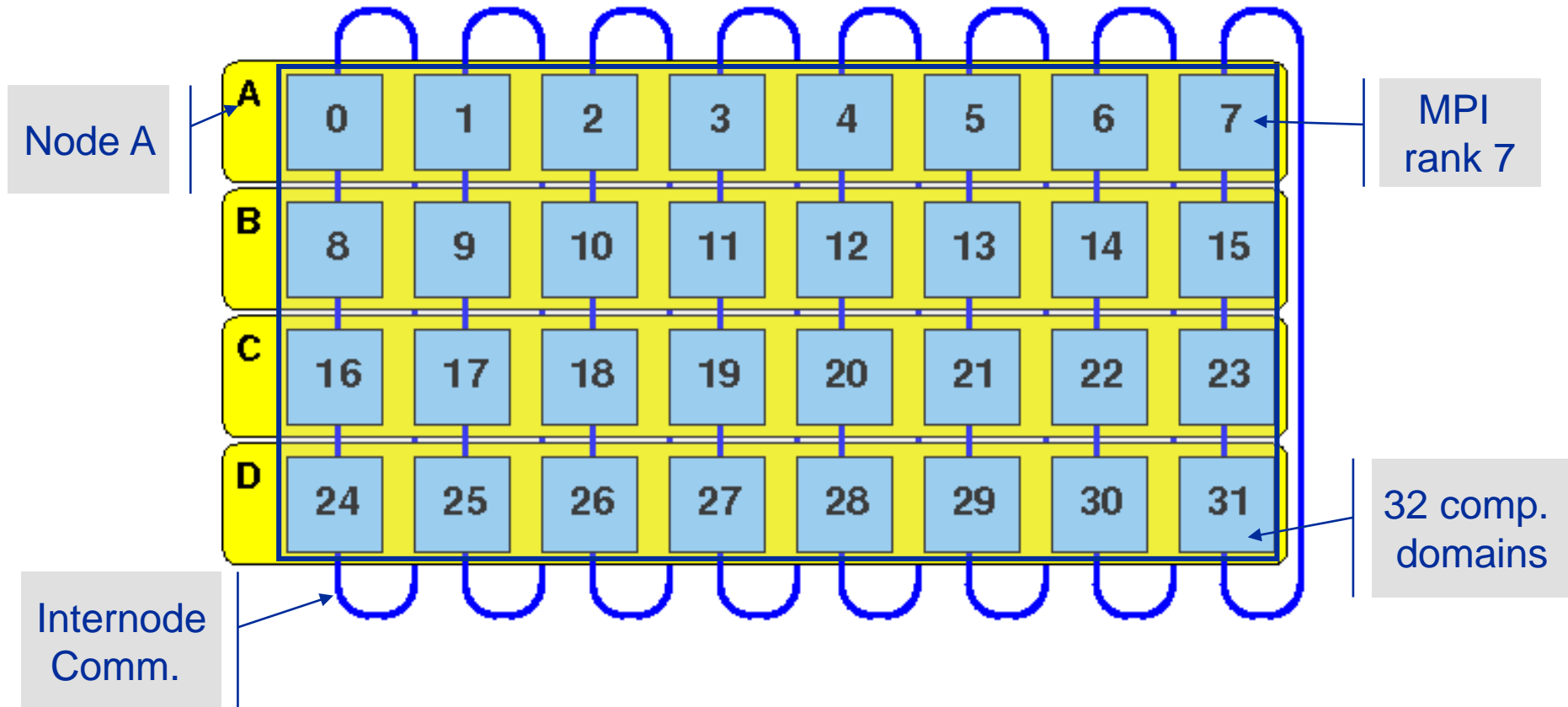
- All modern systems implement a hierarchical architecture with at least two different (in terms of performance) interprocess data paths
 - Communicating within a node is assumed to be (infinitely) fast
 - Communication between nodes through the network (**slow path**)



- `MPI_Cart_create(..., reorder=.true., ...)` should know about topology and optimally reorder process topology but it does not!
- Example (next slides)
 - 4 x 8 cartesian process grid with periodic boundary conditions (PBC)
 - Nearest neighbor data exchange
 - How to best distribute to 4 nodes (A,B,C,D) with 8 cores each?
 - Consider **internode communication** only



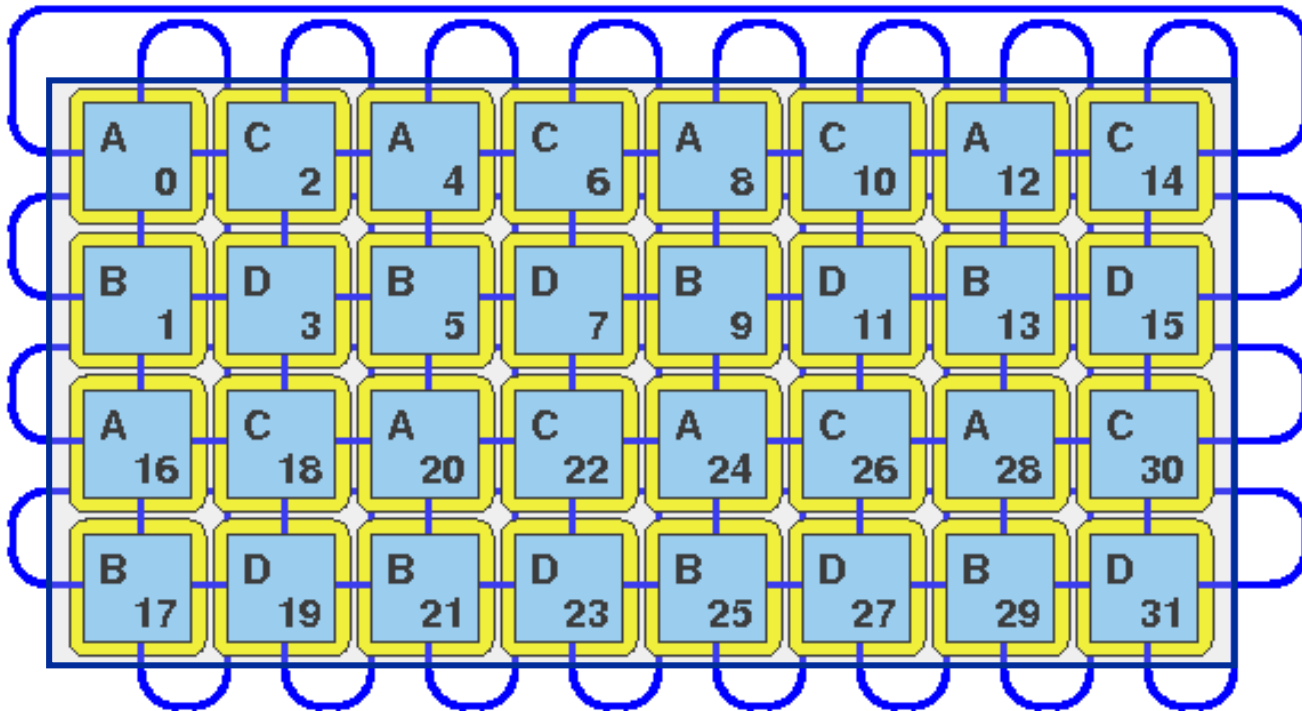
- **Blockwise** mapping of 32 MPI processes to the 4 nodes (A,B,C,D) (default for most `mpirun` implementations)



- **16 internode connections per node** when using this process topology and mapping for domain decomposition



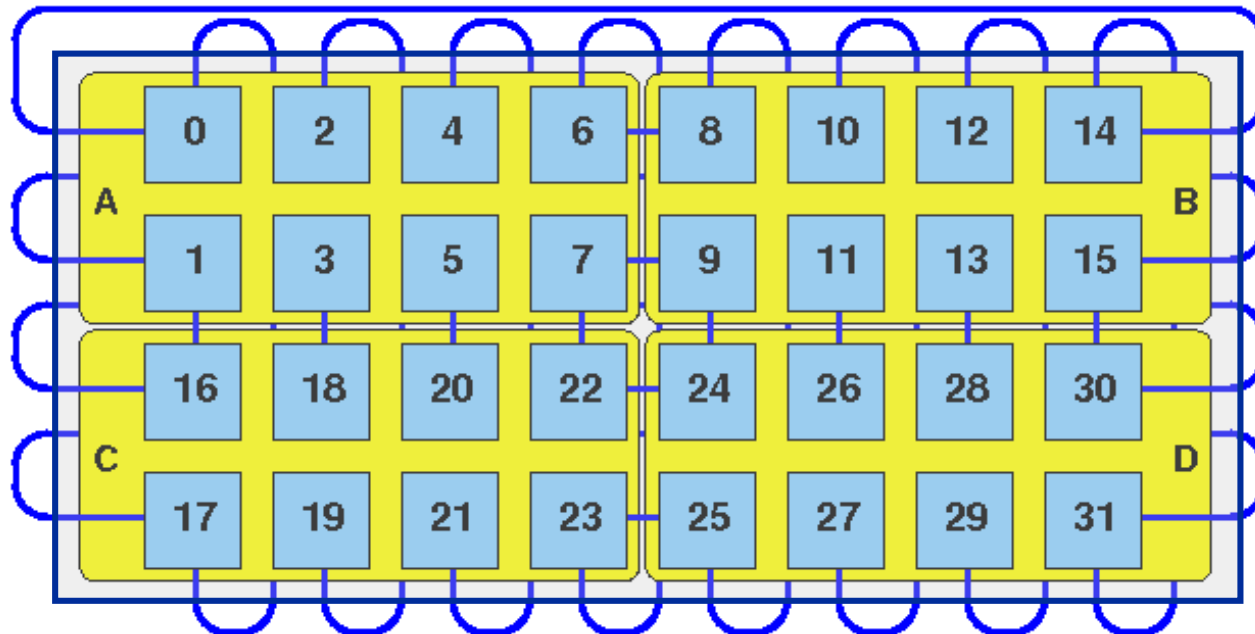
- **Round-robin (cyclic)** mapping of 32 MPI processes to 4 nodes (A,B,C,D) (`mpirun -rr` for most implementations)



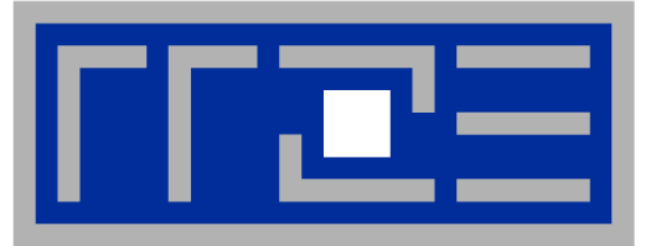
- **32 internode connections per node** when using this process topology and mapping for domain decomposition



- Optimal mapping of 32 MPI processes to 4 nodes (A,B,C,D)



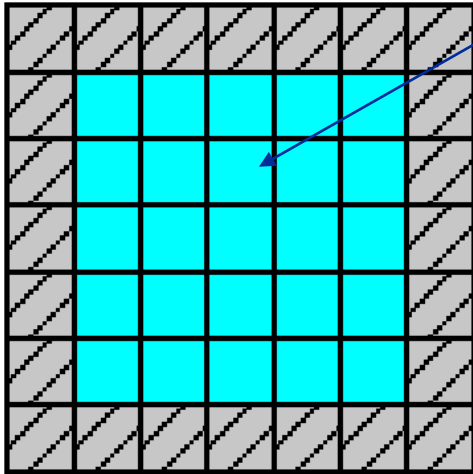
- 12 internode connections** per node when using this process topology and mapping for domain decomposition
- `MPI_Cart_create(..., reorder=.true.,...)` **should do it - does not!**
- No standard way** → Manual remapping of ranks: rank \leftrightarrow newrank (e.g. blockwise $\{4, 5, 6, 7\} \leftrightarrow \{8, 9, 10, 11\}$)



Let's implement 3D Jacobi type solver



- Add halo layers to computational domain on each MPI process

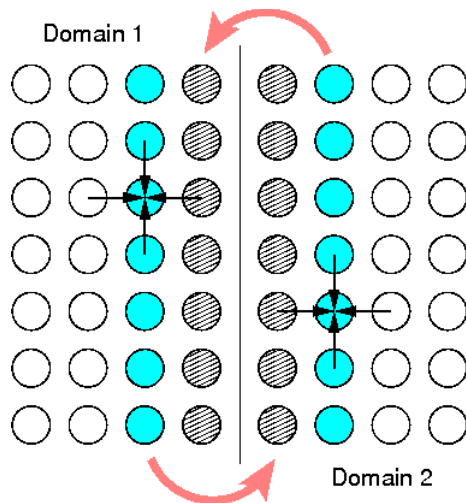


If N_{loc} is extension of cubic subdomain on each process then the overall cells on each process (including halo cells) is $(N_{loc}+2)^3$ for Jacobi3D

→

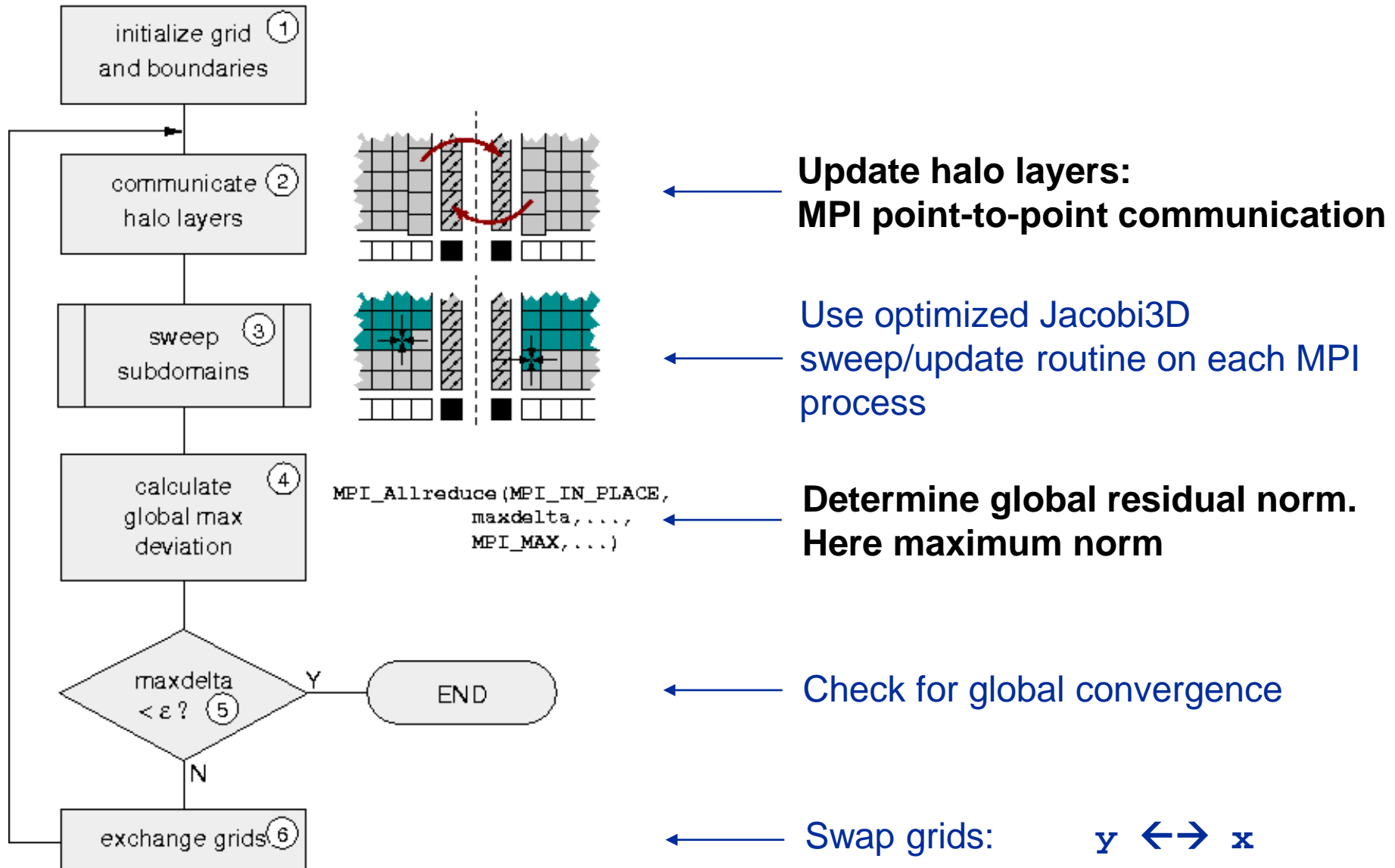
- local computation: N_{loc}^3
- communication overhead (in 3D): $6 * N_{loc}^2$
- Overhead $\sim 1 / N_{loc}$ (holds for 2D as well)

- But N_{loc} is limited by available memory!
- N_{loc} is typically smaller for 3D case
- 10^6 cells per process $\rightarrow N_{loc}=10^2$ (3D) and 10^3 (2D)



MPI parallelization

Jacobi solver – MPI code flowchart



MPI parallelization

Jacobi solver – Initialize (1)



- 3D domain with $N_i \times N_j \times N_k$ cells running on `numprocs` MPI ranks

`spat_dim(d)`: #cells in d-direction (`spat_dim(1)=Nk`, `spat_dim(2)=Nj`, `spat_dim(3)=Ni`)

`pbs_check(d)`: periodicity in d-direction; if `.true.` then PBC are applied in d-direction

```
logical, dimension(1:3) :: pbs_check
```

```
integer, dimension(1:3) :: spat_dim, proc_dim
```

`proc_dim(d)`: User may specify specific domain decomposition or set 0

```
call MPI_Comm_rank(MPI_COMM_WORLD, myid, ierr)
call MPI_Comm_size(MPI_COMM_WORLD, numprocs, ierr)
```

```
if(myid.eq.0) then
  write(*,*) ' spat_dim , proc_dim, PBC ? '
  do i=1,3
    read(*,*) spat_dim(i), proc_dim(i), pbs_check(i)
  enddo
endif
```

Read input on master process and broadcast input

```
call MPI_Bcast(spat_dim , 3, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
call MPI_Bcast(proc_dim , 3, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
call MPI_Bcast(pbs_check, 3, MPI_LOGICAL, 0, MPI_COMM_WORLD, ierr)
```



- **Set up 3D Cartesian process topology**

Distribute `numprocs` in a 3D Cartesian topology, e.g. `numprocs=16` → `proc_dim={4,2,2}`

```
call MPI_Dims_create(numprocs, 3, proc_dim, ierr)

if(myid.eq.0) write(*,'(a,3(i3,x))') 'Grid: ', &
    (proc_dim(i),i=1,3)
```

(1) `l_reorder = .true.`

```
call MPI_Cart_create(MPI_COMM_WORLD, 3, proc_dim, pbc_check, &
    l_reorder, GRID_COMM_WORLD, ierr)
```

Create a 3D process topology with appropriate boundary conditions.

(3) `if(GRID_COMM_WORLD .eq. MPI_COMM_NULL) goto 999`

```
call MPI_Comm_rank(GRID_COMM_WORLD, myid_grid, ierr)
call MPI_Comm_size(GRID_COMM_WORLD, nump_grid, ierr)
```

} Determine ranks & processes in new communicator

(1) Allow MPI reordering of process ids to match communication pattern

(2) Receive a new communicator for process topology → `GRID_COMM_WORLD`

(3) If total size of Cartesian process topology is smaller than `numprocs` → mapped out

MPI parallelization

Jacobi solver – Initialize (3)



- **After topology is set up: Set up rank local subdomains and allocate memory accordingly**

```
integer, dimension(1:3) :: loca_dim, mycoord

call MPI_Cart_coords(GRID_COMM_WORLD, myid_grid, 3,
    mycoord, ierr)

do i=1,3
    loca_dim(i) = spat_dim(i)/proc_dim(i)
    if(mycoord(i) < mod(spat_dim(i),proc_dim(i))) then
        local_dim(i) = loca_dim(i)+1
    endif
enddo

iStart = 0 ; iEnd = loca_dim(3)+1
jStart = 0 ; jEnd = loca_dim(2)+1
kStart = 0 ; kEnd = loca_dim(1)+1

allocate(phi(iStart:iEnd, jStart:jEnd, kStart:kEnd, 0:1))
    Initialize computational domain and boundary
    layers(not shown)
```

Determine local process coordinates

Determine local subdomain extensions

Allocate local computational domain including boundary cells

MPI parallelization

Jacobi solver – Initialize (4)



- **Set up send and receive buffers for halo exchange through MPI**
(tc: Current time step $\in \{0,1\}$; $l_{di} \leftrightarrow \text{local_dim}(i)$)

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

! j-k plane    Exchange in "3-direction"
totmsgsize(3) = loca_dim(1)*loca_dim(2)  phi( 0 , 1:ld2 , 1:ld1 , tc) &
MaxBufLen=max(MaxBufLen,totmsgsize(3))  phi(iEnd, 1:ld2 , 1:ld1 , tc)
! i-k plane
totmsgsize(2) = loca_dim(1)*loca_dim(3)  phi(1:ld3 , 0 , 1:ld1 , tc) &
MaxBufLen=max(MaxBufLen,totmsgsize(2))  phi(1:ld3, jEnd , 1:ld1 , tc)
! i-j plane
totmsgsize(1) = loca_dim(2)*loca_dim(3)  phi(1:ld3, 1:ld2 , 0 , tc) &
MaxBufLen=max(MaxBufLen,totmsgsize(1))  phi(1:ld3, 1:ld2 , kEnd , tc)
```

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

Frequent drawback of combining halo cells and computational domain:
Additional copy operation between halo cells and separate message buffer

MPI parallelization

Jacobi solver – parallel code

Begin: outer iteration

```
t0=0 ; t1=1
tag = 0
do iter = 1, ITERMAX
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
```

```
call Jacobi_sweep(loca_dim(1), loca_dim(2), loca_dim(3), &
phi(iStart, jStart, kStart, 0), t0, t1, &
maxdelta)
```

```
call MPI_Allreduce(MPI_IN_PLACE, maxdelta, 1, &
MPI_DOUBLE_PRECISION, &
MPI_MAX, 0, GRID_COMM_WORLD, ierr)
```

```
if(maxdelta<eps) exit
tmp=t0; t0=t1; t1=tmp
enddo ! iter
```

```
999 continue
```

Check for convergence

End: outer iteration

Determine neighbors in **dir** – direction

Receive from **source** and send to **dest**

Copy local boundary cells to send buffer

Exchange boundaries for a maximum of 6 surfaces → 6 Irecv/Send pairs; pair wise synchronization in each step

Copy receive buffer to halo cells

Do Jacobi iteration on each subgrid independently & determine process local residual (**maxdelta**)

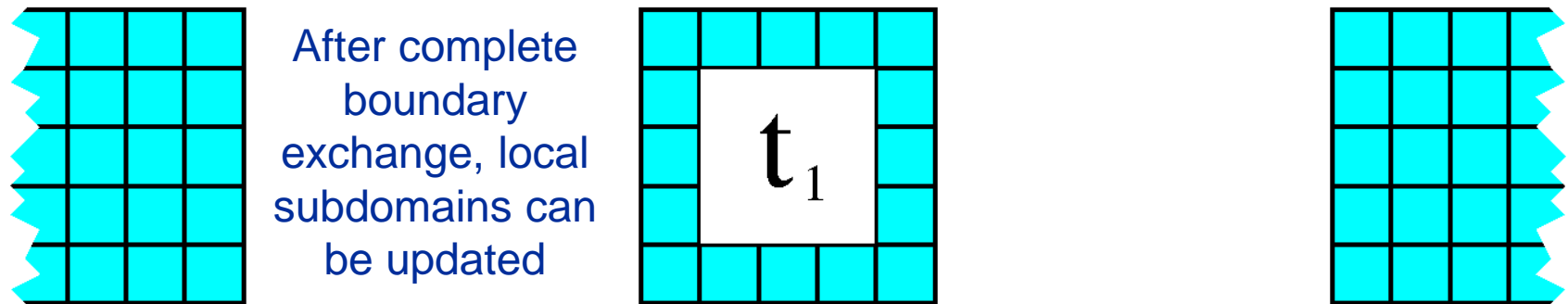
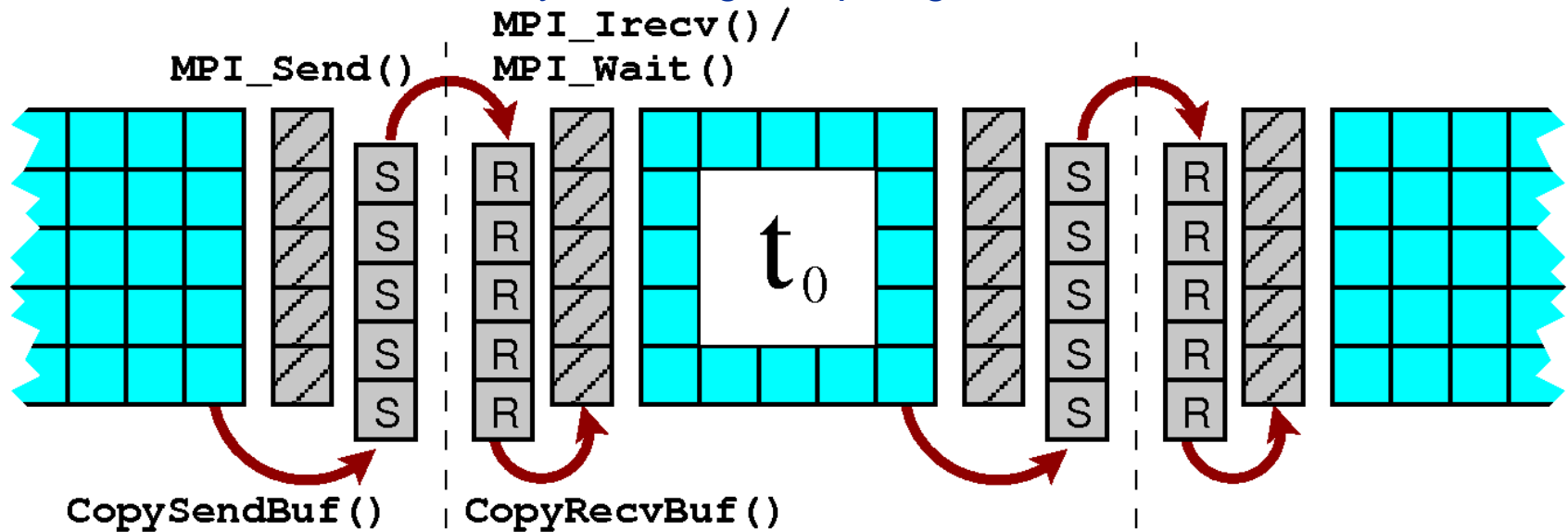
Determine global residual (**maxdelta**)

MPI parallelization

Jacobi solver – data exchange



One boundary exchange step, e.g. `disp=+1, dir =3`



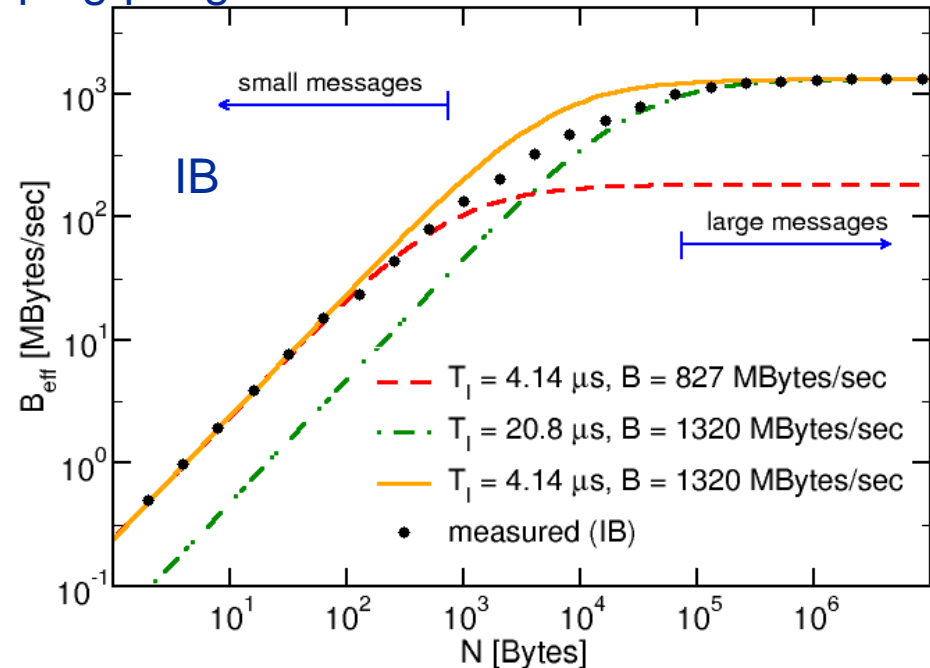
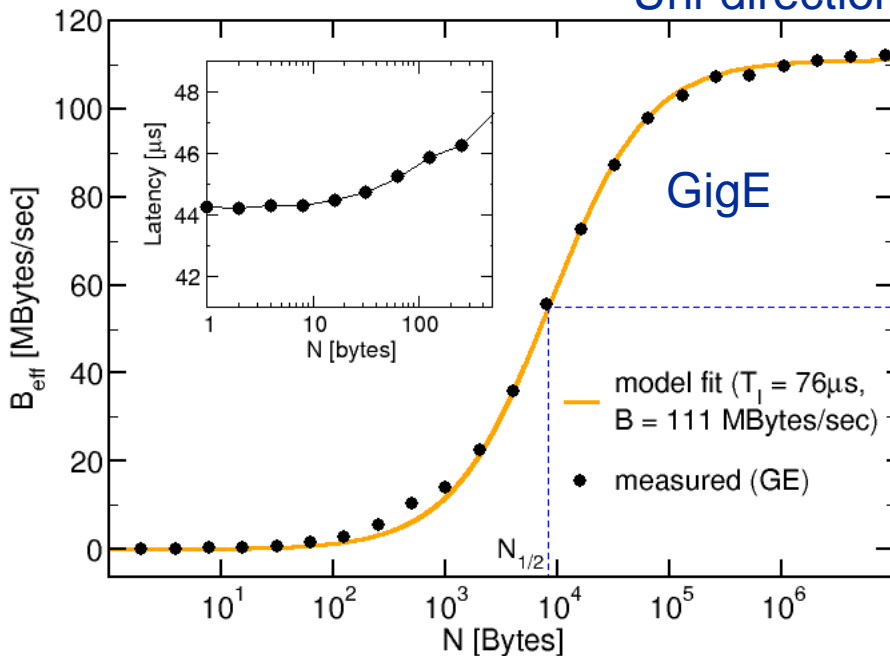
MPI parallelization

Jacobi solver – GBit/IB cluster



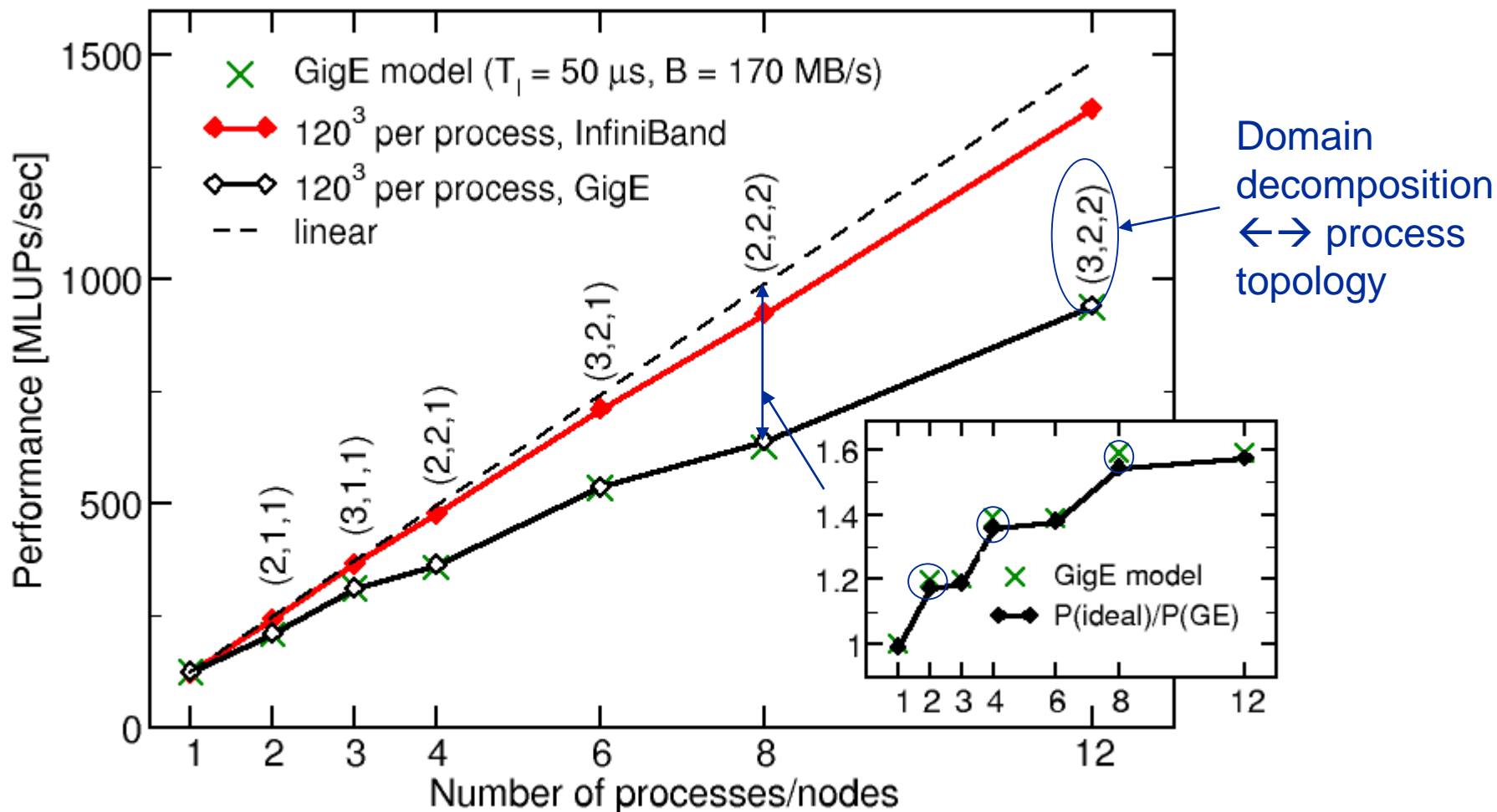
- Run parallel code on single node cluster with
 - GBit Ethernet (GigE) and DDR Infiniband (IB) interconnect
 - Each node is a dual-core Intel Xeon 3070 processor (Core2)
 - Run a single MPI process per node to test for network limitations
 - Single core – serial performance: 130 MLUPs/sec

Uni-directional ping-pong bandwidth





- Domain per MPI process: 120^3





- Maximum data volume to be communicated (L=120; cubic subdomains!) per process

- L^2 cells * 8 Byte / cell * 2 * $k =: c(L, \vec{N})$

Send/receive in each direction **Max. number of neighbors for communication**

- Max. time for communication: $T_c(L, \vec{N}) = \frac{c(L, \vec{N})}{B} + kT_\ell$
(B: Bi-directional bandwidth; T_ℓ : latency)

- $T_s(L)$ is the time required to update a subdomain of L^3 cells

- Performance model:

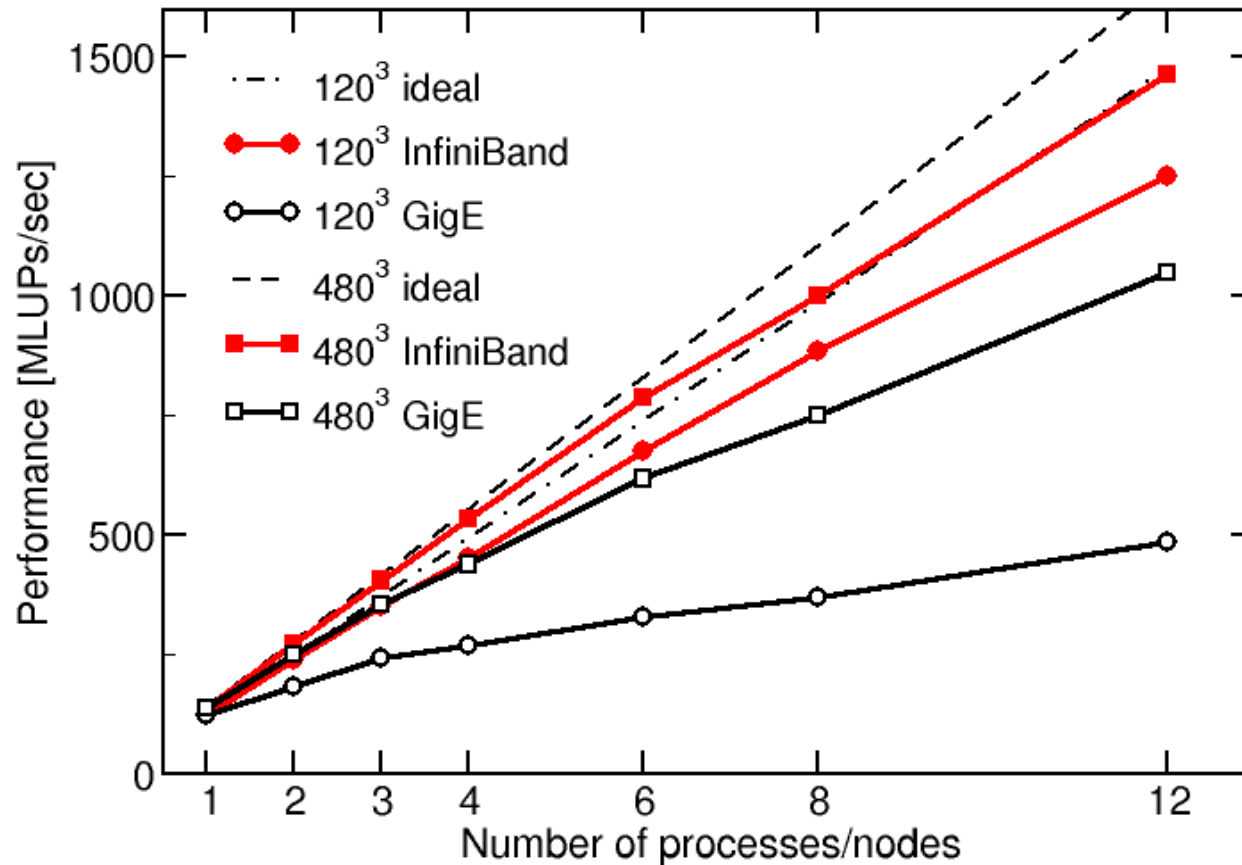
$$P(L, \vec{N}) = \frac{L^3 N}{T_s(L) + T_c(L, \vec{N})}$$

B=170 MByte/s; $T_\ell=50\mu\text{s}$

N	(N_z, N_y, N_x)	k	$c(L, \vec{N})$ [MB]	$P(L, \vec{N})$ [MLUPs/sec]	$\frac{NP_1(L)}{P(L, \vec{N})}$
1	(1,1,1)	0	0.000	124	1.00
2	(2,1,1)	2	0.461	207	1.20
3	(3,1,1)	2	0.461	310	1.20
4	(2,2,1)	4	0.922	356	1.39
6	(3,2,1)	4	0.922	534	1.39
8	(2,2,2)	6	1.382	625	1.59
12	(3,2,2)	6	1.382	938	1.59

MPI parallelization

Jacobi solver – strong scaling



- Scalability is rather limited on GigE
- $B \rightarrow B(L(N))$; cf. ping-pong measurements
- Single core performance depends on L
- Refined performance model required.
- Recover magic processor numbers: 2,4,8