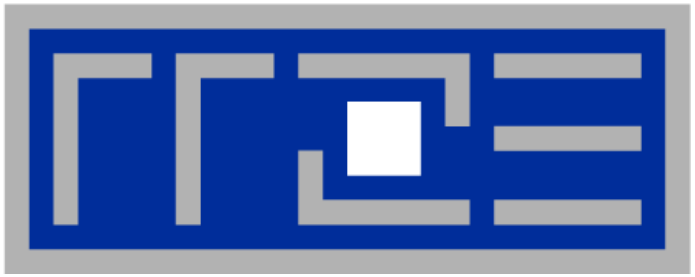


Programming Techniques for Supercomputers:

A few comments on hybrid MPI/OpenMP programming

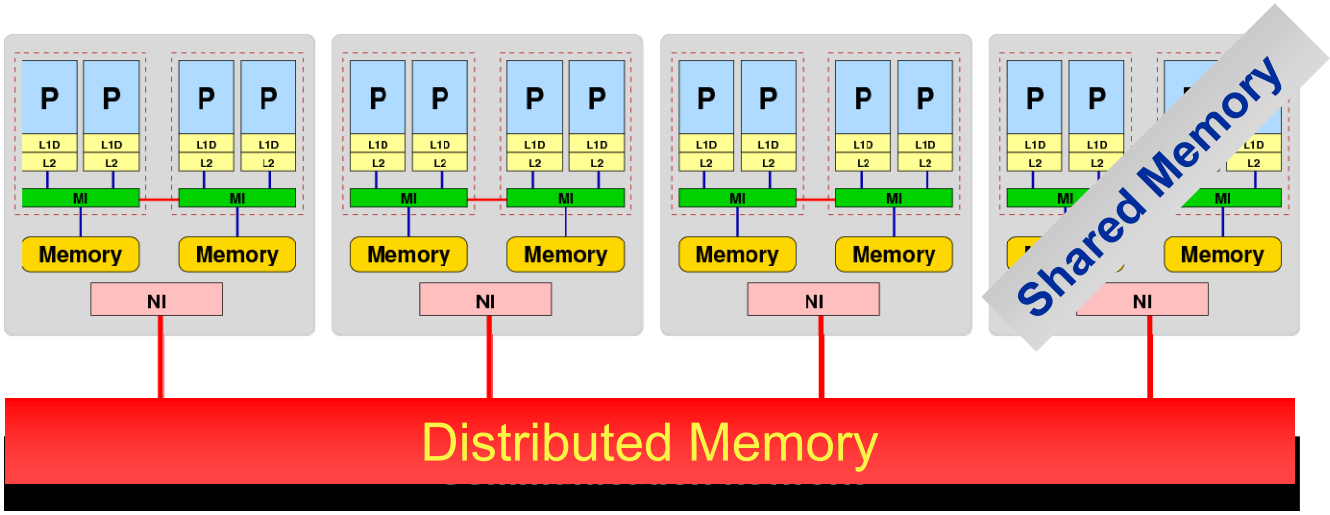
Prof. Dr. G. Wellein[a,b] , Dr. G. Hager[a] , J. Habich[a]

[a]HPC Services – Regionales Rechenzentrum Erlangen
[b]Department für Informatik

University Erlangen-Nürnberg, Sommersemester 2011

Hybrid MPI/OpenMP
Most architectures are of hybrid type

- **Hybrid distributed/shared memory computer architectures**



- **Use hybrid programming model?!**

  MPI for communication between the nodes
  OpenMP within the node

- **But the hybrid model can further be refined:**
  - 1 MPI process per socket with n-threads if running on n-core processor chips
  - Use dedicated OpenMP threads for asynchronous MPI communication
  - ….

Hybrid MPI/OpenMP
Jacobi solver – hybrid code!?

Insert OpenMP directives

Compile with –openmp and
link with MPI library

```
setenv OMP_NUM_THREADS=8
mpirun -pernode ./jacobiHybrid
```

Starts one MPI process per node which launches 8 threads

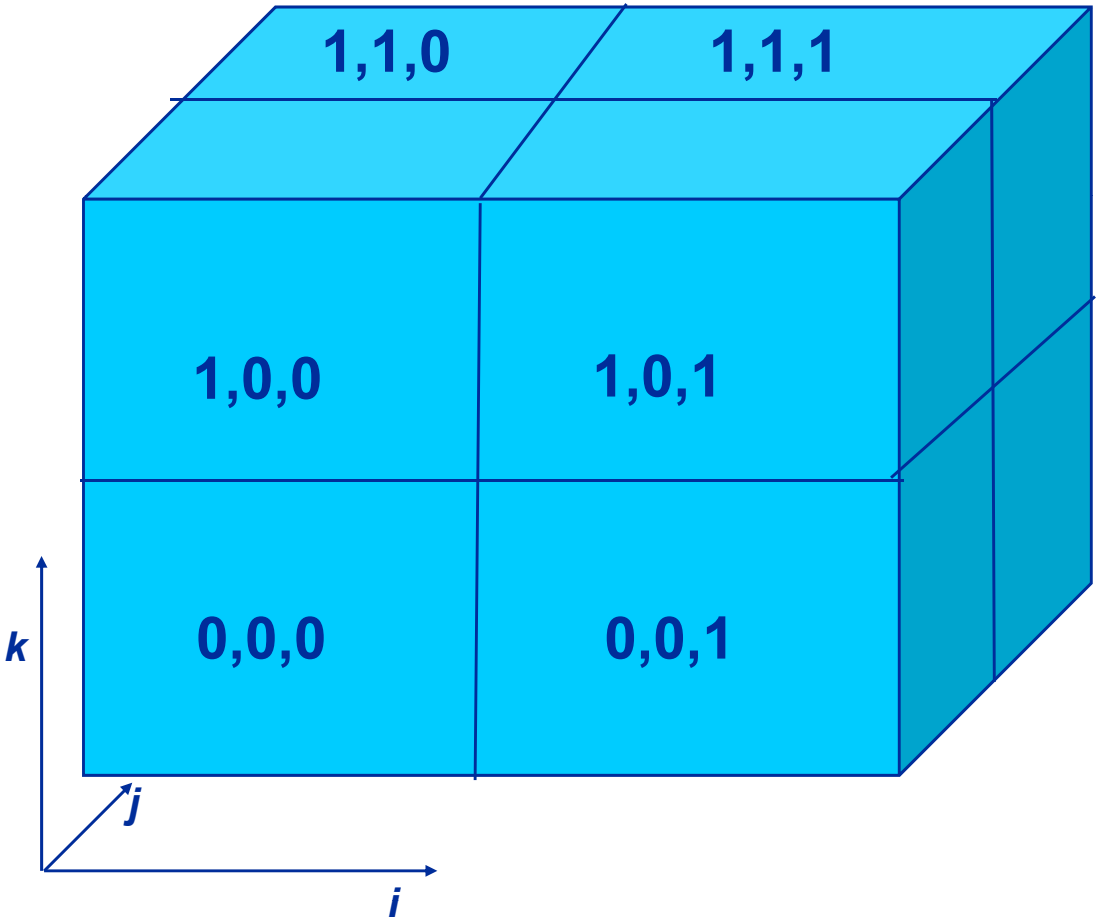
```
!$OMP PARALLEL DO private(…)
  do k=1,Nk
    do j=1,Nj
      do i=1,Ni
        phi(i,j,k,t0) = …
      enddo
    enddo
  enddo
```

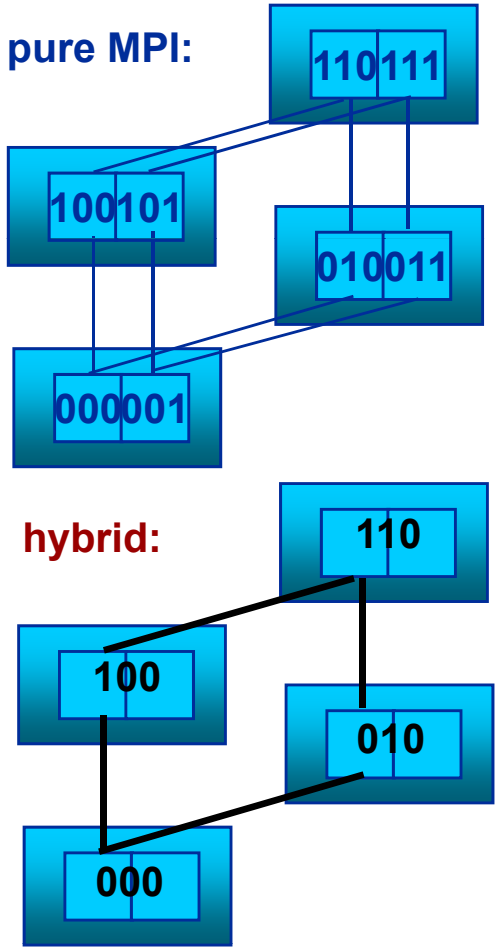
"vector mode"

```
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
```

Hybrid MPI/OpenMP
Jacobi solver – simple hybrid approach ("vector mode")

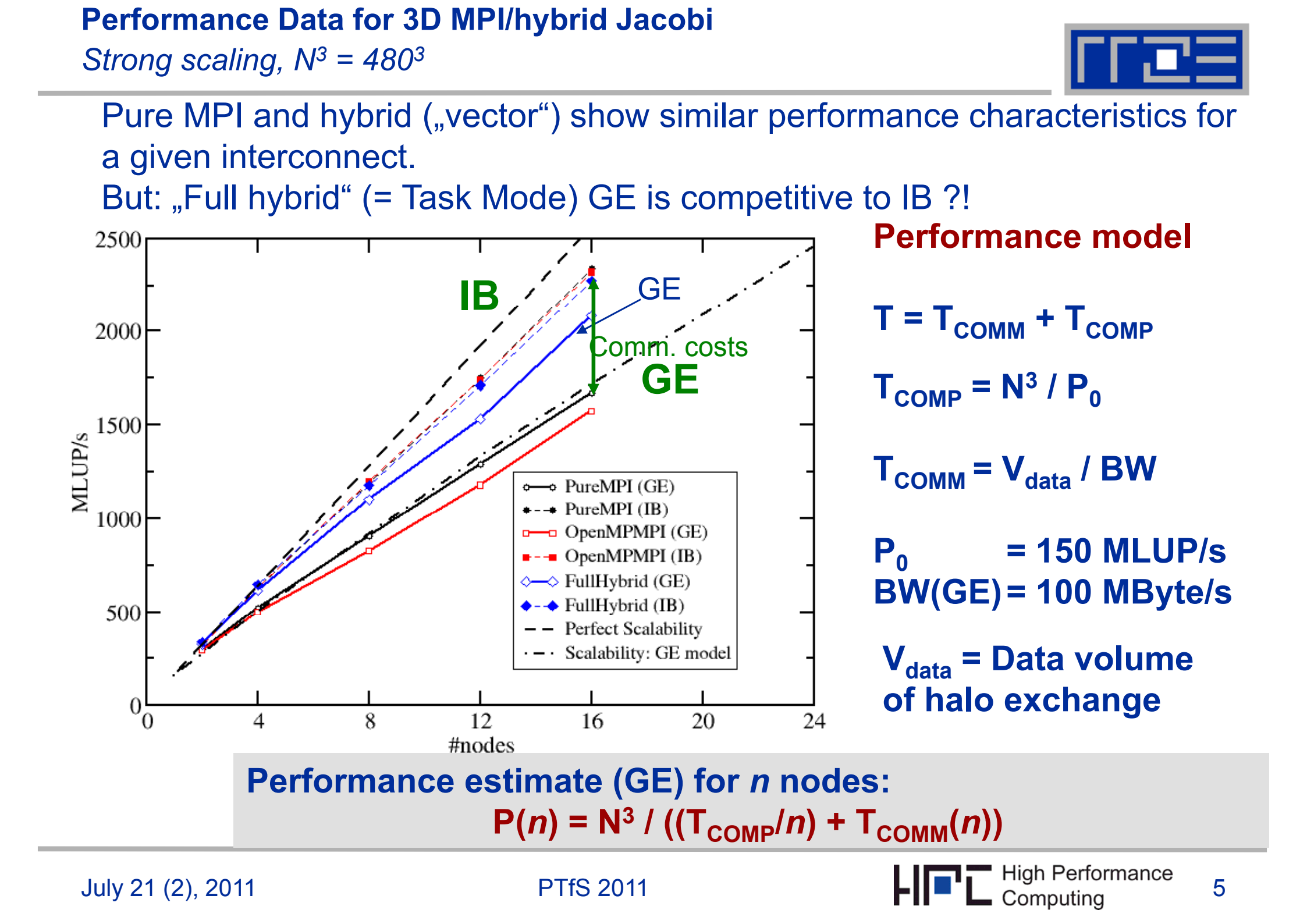
- Cubic 3D computational domain with periodic BCs in all directions
- Use single-node IB/GE cluster with one dual-core chip per node
- Homogeneous distribution of workload, e.g. on 8 procs





Performance Data for 3D MPI/hybrid Jacobi

Strong scaling, $N^3 = 480^3$



Pure MPI and hybrid („vector“) show similar performance characteristics for a given interconnect.

But: „Full hybrid“ (= Task Mode) GE is competitive to IB ?!

Performance model

$T = T_{COMM} + T_{COMP}$

$T_{COMP} = N^3 / P_0$

$T_{COMM} = V_{data} / BW$

P_0 = 150 MLUP/s
BW(GE) = 100 MByte/s

V_{data} = Data volume of halo exchange

Performance estimate (GE) for *n* nodes:

$$P(n) = N^3 / ((T_{COMP}/n) + T_{COMM}(n))$$

Hybrid MPI/OpenMP "Task mode" implementation

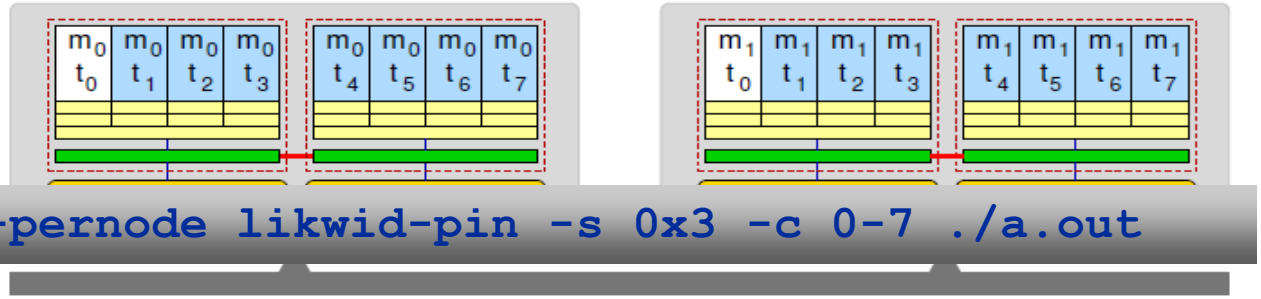
- **Task Mode: Functional decomposition and decoupling of communication and computation**
- **threadID=0: a) update boundary cells b) do halo exchange**
- **threadID>0: update all inner cells**
- **BARRIER after each outer iteration**

```
!$OMP PARALLEL PRIVATE(iteration,threadID,k,j,i,...)
  threadID  = omp_get_thread_num()
  do iteration=1,MAXITER
  ...
  if(threadID .eq. 0) then
  ...
! Standard 3D Jacobi iteration
! updating BOUNDARY cells
  ...
! After updating BOUNDARY cells
! do halo exchange

    do dir=i,j,k
      call MPI_Irecv( halo data from neighbor in -dir direction )
      call MPI_Send( data to neighbor in +dir direction )
      call MPI_Irecv( halo data from neighbor in +dir direction )
      call MPI_Send( data to neighbor in -dir direction )
    enddo

    call MPI_Waitall( )

  else ! not thread ID 0

! Remaining threads perform
! update of INNER cells 2,...,N-1
! Distribute outer loop iterations manually:

    chunksize = (N-2) / (omp_get_num_threads()-1) + 1
    my_k_start = 2 + (threadID-1)*chunksize
    my_k_end = 2 + (threadID-1+1)*chunksize-1
    my_k_end = min(my_k_end, (N-2))

! INNER cell updates
    do k = my_k_start , my_k_end
      do j = 2, (N-1)
        do i = 2, (N-1)
          ...
        enddo
      enddo
    enddo
  endif ! thread ID
!$OMP BARRIER
  enddo
!$OMP END PARALLEL
```

Manual workshare distribution…

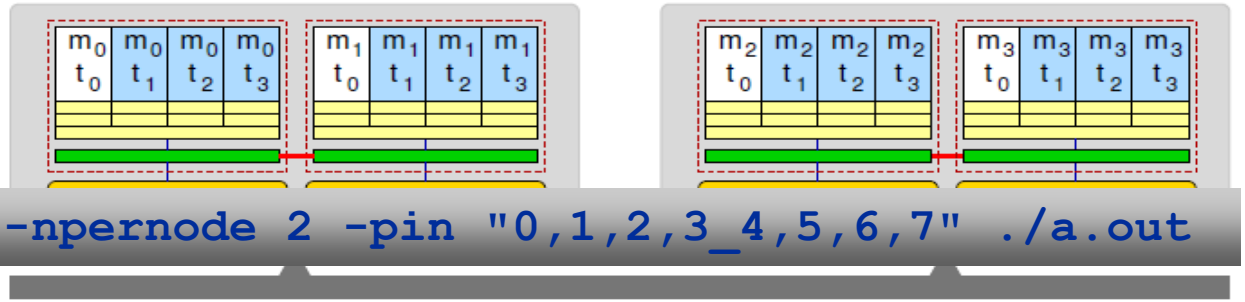
Topology ("mapping") choices with MPI+OpenMP Dual-Socket Quad-core nodes

One MPI process per node



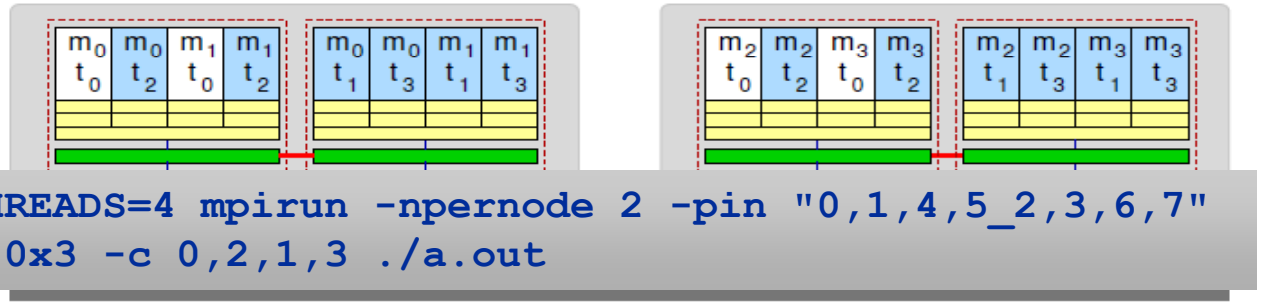

```
env OMP_NUM_THREADS=8 mpirun -pernode likwid-pin -s 0x3 -c 0-7 ./a.out
```

One MPI process per socket



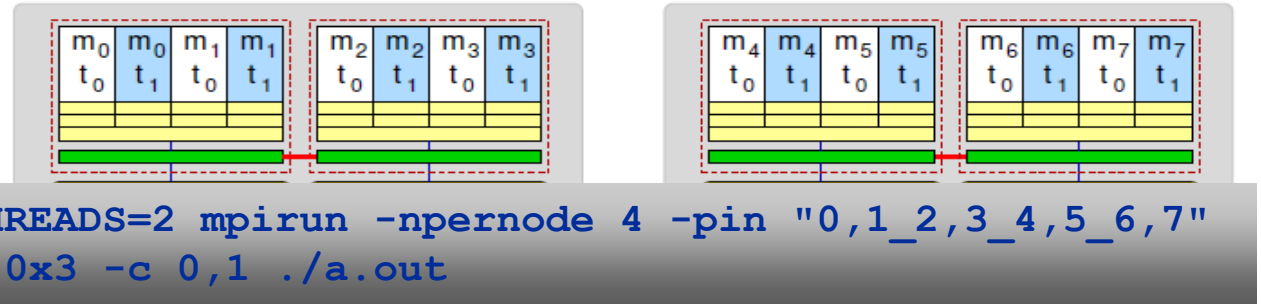

```
env OMP_NUM_THREADS=4 mpirun -npernode 2 -pin "0,1,2,3_4,5,6,7" ./a.out
```

OpenMP threads pinned "round robin" across cores in node




```
env OMP_NUM_THREADS=4 mpirun -npernode 2 -pin "0,1,4,5_2,3,6,7"
likwid-pin -s 0x3 -c 0,2,1,3 ./a.out
```

Two MPI processes per node



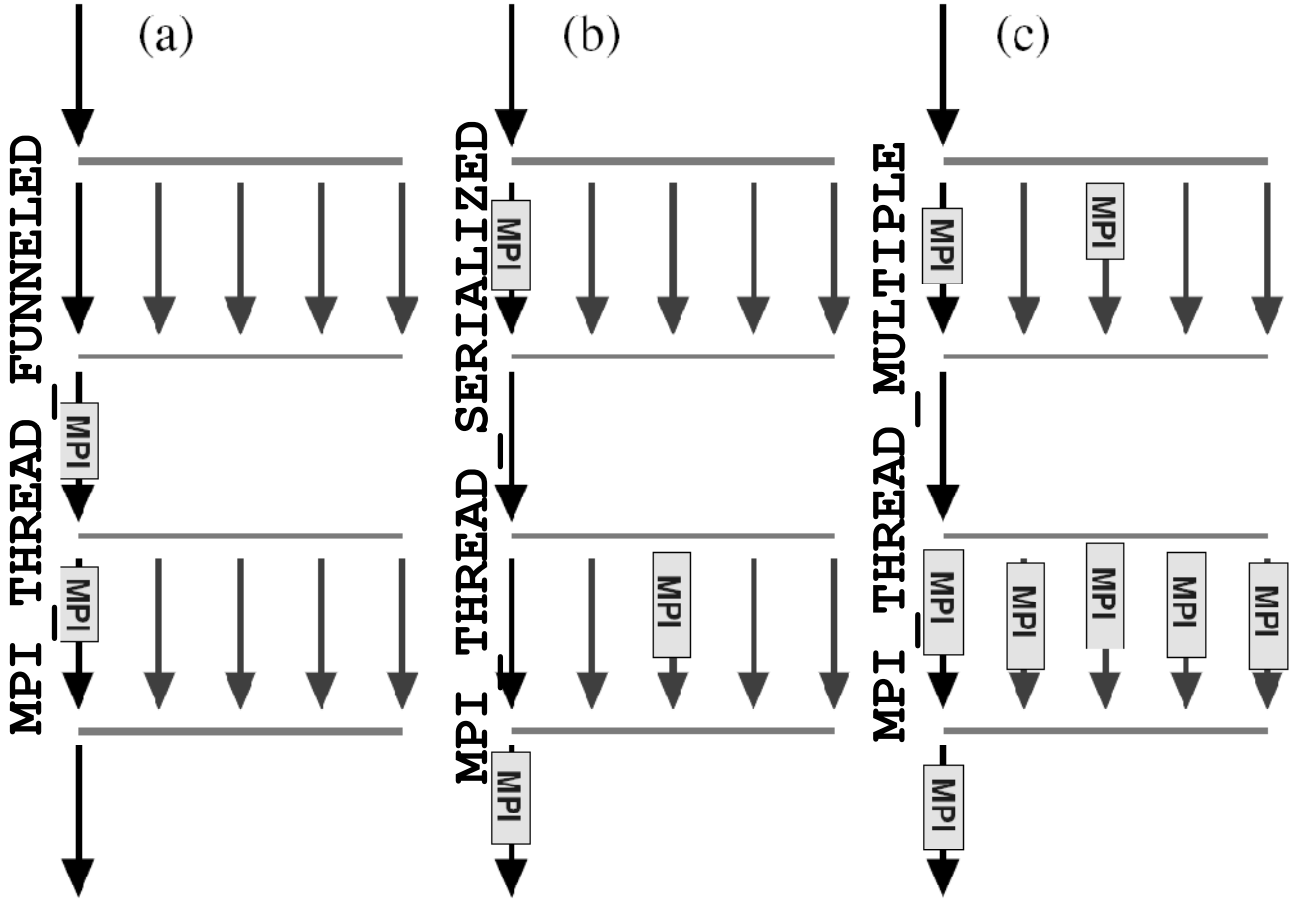

```
env OMP_NUM_THREADS=2 mpirun -npernode 4 -pin "0,1_2,3_4,5_6,7"
likwid-pin -s 0x3 -c 0,1 ./a.out
```

Hybrid MPI/OpenMP
MPI taxonomy of thread interoperability

- MPI implementation must support threaded execution
- MPI standard defines 4 levels of support:

`MPI_THREAD_SINGLE`

Only one thread executes

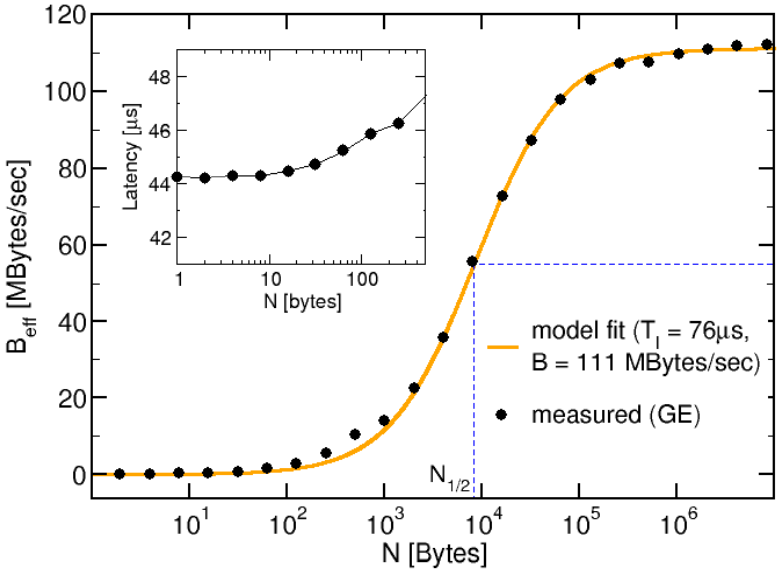


- At least one must be available. Determine which one is available:

```
MPI_INIT_THREAD( required, provided, ierror )
INTEGER required, provided, ierror
```

Hybrid MPI/OpenMP
Potential benefits and drawbacks

- **If your MPI code scales very well – do not try to be better with hybrid:** Additional overhead, ccNUMA,…
- **There must be a good reason for going hybrid, e.g.**
  - Explicitly overlapping communication with computation (→ cf. task mode)
  - Improve MPI performance through larger messages, e.g. domain decomposition with 1 instead of 8 processes per node may increase message size by 4x! ("ride the ping-pong curve")



  - Exploiting additional levels of parallelism (vector processor codes, parallel structures where MPI overhead is too large for parallelization)
  - Reuse of data in shared caches → wavefront
  - Improve convergence if you parallelize loop carried dependences, e.g. domain decomposition with a implicit solver