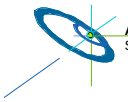
A Heat-Transfer Example with MPI

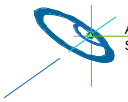
Rolf Rabenseifner

University of Stuttgart
High-Performance Computing-Center Stuttgart (HLRS)
www.hlrs.de



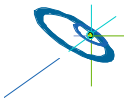
Goals

- first complex MPI example
- exercise with basic MPI features
- base for your own applications



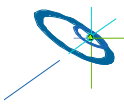
Heat: MPI features

- block data decomposition
- communication: filling the halo with
  - non-blocking point-to-point
  - blocking MPI_SENDRECV
  - MPI_ALLTOALLV
- and for the abort-criterion
  - MPI_ALLREDUCE
  - usage of message tags in point-to-point communication
  - MPI_BCAST
- derived datatypes: MPI_TYPE_VECTOR
- Minor features: MPI_ADDRESS(), MPI_BOTTOM, topologies, MPI_PROC_NULL, MPI_Wtime()
- Performance



Heat Conduction Program - an Example

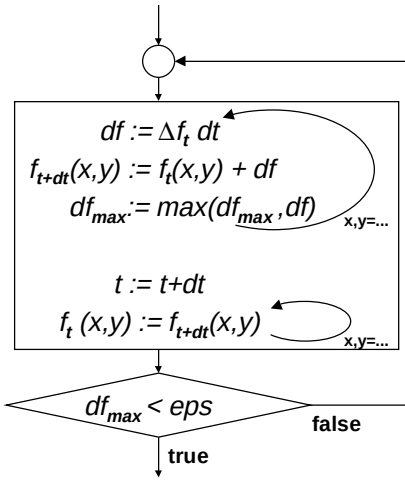
- solves the partial differential equation for unsteady heat conduction $df/dt = \Delta f$
- uses an explicit scheme: forward-time, centered-space
- solves the equation over a square domain
- initial conditions: *f=0* everywhere inside the square
- boundary conditions: *f=x* on all edges
- number of grid points: 80x80
- based on Parallel CFD Test Case *www.hlrs.de/people/resch/PROJECTS/PARACFD.html*
- Aims of this session:
  - how to parallelize the example
  - to recapitulate most of the MPI methods



Fortran Introduction for C Programmers

- Array declaration:
  - *type array_name (first_index_dim1 : last_index_dim1, first_index_dim2 : last_index_dim2, ...)*
  - *aaa(17)* means *aaa(1:17)*
- Memory layout: the first index is contiguous
- Using the array:
  - one element: *array_name(i1,i2,...)*
  - sub-array: *array_name (first_sub_index1 : last_sub_index1, first_sub_index2 : last_sub_index2, ...)*
  - referencing:
    - **the whole array by the *array_name***
    - **a contiguous subarray by the fist element *array_name(i1,i2,...)***
- Logical operations: *.gt.* / *.lt.* means *greater / less than*
  .ge. / *.le.* means *greater / less than or equal to*
  .eq. / *.ne.* means *equal / not equal to*

Heat: Sequential Program — Scheme



Heat: Sequential Program — heat-big.f

```
 1          program heat
 2          parameter (imax=80,kmax=80)
 3          parameter (itmax=20000)
 4          double precision eps
 5          parameter (eps=1.d-08)
 6          double precision
              phi(0:imax,0:kmax),
              phin(1:imax-1,1:kmax-1)
 7          double precision
              dx,dy,dx2,dy2,dx2i,dy2i,dt,
              dphi,dphimax
 8          real tarray(2)
 9  c
10          dx=1.d0/kmax
11          dy=1.d0/imax
12          dx2=dx*dx
13          dy2=dy*dy
14          dx2i=1.d0/dx2
15          dy2i=1.d0/dy2
16          dt=min(dx2,dy2)/4.d0
17  c start values 0.d0
18          do k=0,kmax-1
19           do i=1,imax-1
20             phi(i,k)=0.d0
21           enddo
22          enddo
23  c start values 1.d0
24          do i=0,imax
25           phi(i,kmax)=1.d0
26          enddo
27  c start values dx
28          phi(0,0)=0.d0
29          phi(imax,0)=0.d0
30          do k=1,kmax-1
31           phi(0,k)=phi(0,k-1)+dx
32           phi(imax,k)=phi(imax,k-1)+dx
33          enddo
```

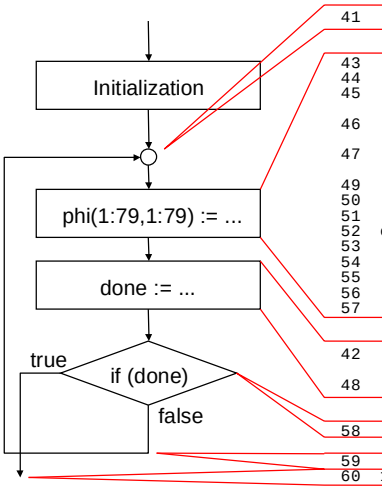
```
34  c print array
35          write (*,'(/,a)')
36       f  'Heat Conduction 2d'
37          write (*,'(/,4(a,1pg12.4))')
38       f  'dx =',dx,', dy =',dy,', dt =',dt,',
                                 eps =',eps
39          t0=dtime(tarray)
40  c iteration
41          do it=1,itmax
42           dphimax=0.
43           do k=1,kmax-1
44            do i=1,imax-1
45              dphi=(phi(i+1,k)+phi(i-1,k)
                      -2.*phi(i,k))*dy2i
46       f             +(phi(i,k+1)+phi(i,k-1)
                        -2.*phi(i,k))*dx2i
47              dphi=dphi*dt
48              dphimax=max(dphimax,dphi)
49              phin(i,k)=phi(i,k)+dphi
50            enddo
51           enddo
52  c save values
53           do k=1,kmax-1
54            do i=1,imax-1
55             phi(i,k)=phin(i,k)
56            enddo
57           enddo
58           if(dphimax.lt.eps) goto 10
59          enddo
60  10      continue
61          t1=dtime(tarray)
62  c print array
63          write (*,'(/,a,i6,a)')
64       f  'phi after',it,' iterations'
65          write (*,'(/,2(a,1pg12.4,/))')
66       f  'user time   = ', tarray(1),
67       f  'system time = ', tarray(2)
68          stop
69          end
```

Main loop

Numerical Iteration

Abort criterion

Heat: Sequential Program Scheme




```
41          do it=1,itmax
43           do k=1,kmax-1
44            do i=1,imax-1
45              dphi=(phi(i+1,k)+phi(i-1,k)
                      -2.*phi(i,k))*dy2i
46       f             +(phi(i,k+1)+phi(i,k-1)
                        -2.*phi(i,k))*dx2i
47              dphi=dphi*dt
                 [...]
49              phin(i,k)=phi(i,k)+dphi
50            enddo
51           enddo
52  c save values
53           do k=1,kmax-1
54            do i=1,imax-1
55             phi(i,k)=phin(i,k)
56            enddo
57           enddo

42          dphimax=0.
             [...]
48            dphimax=max(dphimax,dphi)

58          if(dphimax.lt.eps) goto 10

59          enddo
60  10      continue
```

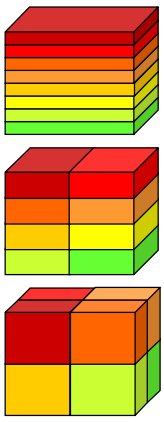
computing inner area

Using outer area

Heat: How to parallelize with MPI

- Block data decomposition of arrays phi, phin
  - array sizes is variable, depends on processor number
- Halo (shadow, ghost cells, ...)
  - to store values of the neighbors
  - width of the halo := needs of the numerical formula (heat: 1)
- Communication:
  - fill the halo with values of the neighbors
  - after each iteration (*dt*)
- global execution of the abort criterion
  - collective operation

Block data decomposition — in which dimensions?



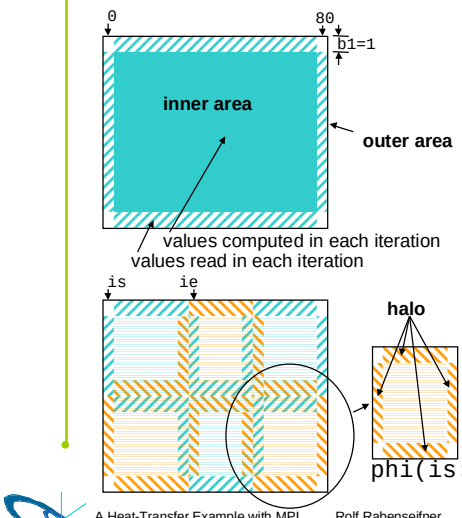
Splitting in

- **one** dimension:
  communication
  $= n^2 * 2 * w * 1$
- **two** dimensions:
  communication
  $= n^2 * 2 * w * 2 / p^{1/2}$
- **three** dimensions:
  communication
  $= n^2 * 2 * w * 3 / p^{2/3}$

w=width of halo
n^3 =size of matrix
p = number of processors
cyclic boundary
—> two neighbors in each direction

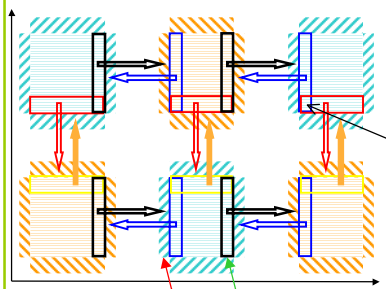
optimal for p>11

Heat: How to parallelize with MPI - Block Data Decomposit.



- Block data decomposition
- Cartesian process topology
  - two-dimensional
    `size = idim * kdim`
  - process coordinates
    `icoord = 0 .. idim-1`
    `kccord = 0 .. kdim-1`
  - see heat-mpi-big.f, lines 18-42
- Static array declaration
  –> dynamic array allocation
  - `phi(0:80,0:80)` –> `phi(is:ie,ks:ke)`
  - additional subroutine "`algorithm`"
  - see heat-mpi-big.f, lines 186+192
- Computation of `is`, `ie`, `ks`, `ke`
  - see heat-mpi-big.f, lines 44-98

Heat: Communication Method - Non-blocking



from **left** to **right**
from **right** to **left**
from **upper** to **lower**
from **lower** to **upper**

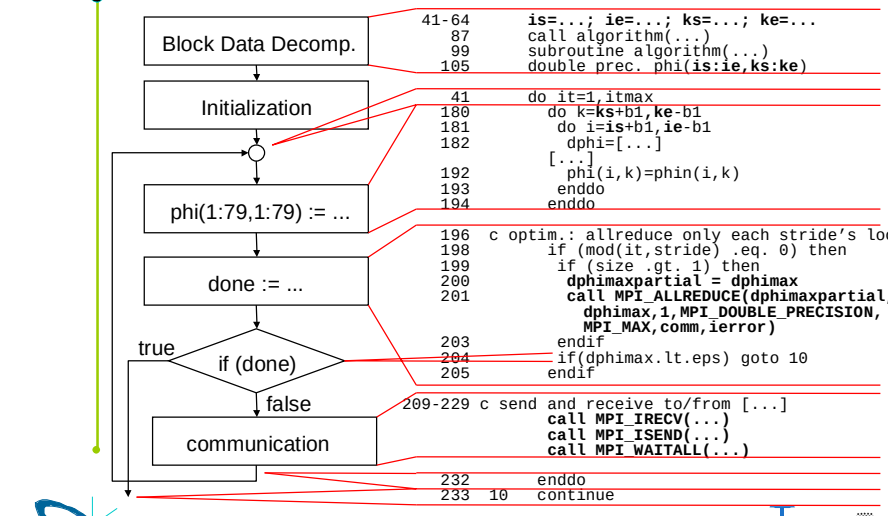
- Communicate halo in each iteration
- Ranks via MPI_CART_SHIFT

```
114   call MPI_CART_SHIFT(comm,
        0, 1, left, right, ierror)
115   call MPI_CART_SHIFT(comm,
        1, 1, lower,upper, ierror)
```

- non-blocking point-to-point
  - **Caution: corners must not be sent simultaneously in two directions**
- vertical boundaries:
  - non-contiguous data
  - MPI derived datatypes used

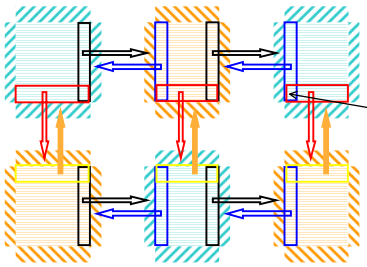
```
118 call MPI_TYPE_VECTOR(
      kinner,b1,iouter,MPI_DOUBLE_PRECISION,
      vertical_border, ierror)
120 call MPI_TYPE_COMMIT(
      vertical_border, ierror)

221 call MPI_IRECV(phi(is,ks+b1),
      1,vertical_border,
      left,MPI_ANY_TAG,comm, req(1),ierror)
225 call MPI_ISEND(phi(ie-b1,ks+b1),
      1,vertical_border,
      right,0,comm, req(3),ierror)
```

Heat: Parallel Program Scheme - heat-mpi0-big.f




```
41-64       is=...; ie=...; ks=...; ke=...
  87        call algorithm(...)
  99        subroutine algorithm(...)
 105        double prec. phi(is:ie,ks:ke)

  41        do it=1,itmax
 180          do k=ks+b1,ke-b1
 181           do i=is+b1,ie-b1
 182             dphi=[...]
               [...]
 192             phi(i,k)=phin(i,k)
 193           enddo
 194          enddo

 196  c optim.: allreduce only each stride's loop:
 198          if (mod(it,stride) .eq. 0) then
 199           if (size .gt. 1) then
 200            dphimaxpartial = dphimax
 201            call MPI_ALLREDUCE(dphimaxpartial,
                  dphimax,1,MPI_DOUBLE_PRECISION,
                  MPI_MAX,comm,ierror)
 203           endif
 204           if(dphimax.lt.eps) goto 10
 205          endif

209-229 c send and receive to/from [...]
               call MPI_IRECV(...)
               call MPI_ISEND(...)
               call MPI_WAITALL(...)

 232        enddo
 233  10    continue
```

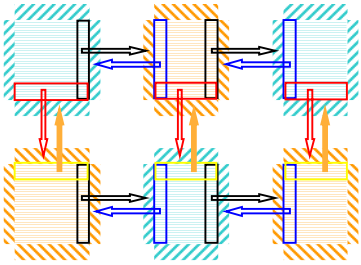

Heat: Different Communication Methods - heat-mpi1-big.f



from **left** to **right**
from **right** to **left**
from **upper** to **lower**
from **lower** to **upper**

- three different communication methods
  - non-blocking point-to-point
    - **examples** `communication1-4` heat-mpi1-big.f, lines 435-442, lines 523-705
    - **Caution: corners must not be sent simultaneously in two directions**
  - MPI_SENDRECV
    - **examples** `communication5+6` heat-mpi1-big.f, lines 435-442, lines 523-705
    - **only for cartesian problems**
  - MPI_ALLTOALLV
    - **label 207, lines 447-468, 224-265**
    - **no MPI tag, see next slide**
    - **Fortran pitfalls with MPI_BOTTOM**

Heat: sending non-contiguous data - heat-mpi1-big.f



from `left` to `right`
from `right` to `left`
from `upper` to `lower`
from `lower` to `upper`

two methods for vertical boundaries:

- MPI derived datatypes (`comm.1-5`)
  - **some MPI implementations are slow**
- local copying (`comm.2-7`)
  - non-contiguous array is copied to contigous scratch array
  - scratch array is sent with MPI
  - **may be executed fast**
    - **optimized by the compiler**
    - **vectorized**
    - **automatically multi-threaded parallelized**
    - **multi-threaded parallelized with OpenMP**

Heat: Non-blocking point-to-point with derived datatypes (1)

```
214  c create a MPI_Vector
215        call MPI_TYPE_VECTOR(kinner,b1,iouter, MPI_DOUBLE_PRECISION,
216       f                     vertical_border,ierror)
217        call MPI_TYPE_COMMIT(vertical_border, ierror)
218
219        call MPI_TYPE_VECTOR(b1,iinner,iouter, MPI_DOUBLE_PRECISION,
220       f                     horizontal_border,ierror)
221        call MPI_TYPE_COMMIT(horizontal_border, ierror)
```

```
535          call MPI_IRECV(phi(is+b1,ks),1,horizontal_border,
536       f                 lower,MPI_ANY_TAG,comm, req(2),ierror)
537          call MPI_IRECV(phi(is+b1,ke),1,horizontal_border,
538       f                 upper,MPI_ANY_TAG,comm, req(1),ierror)
539          call MPI_ISEND(phi(is+b1,ke-b1),1,horizontal_border,
540       f                 upper,upper_right_tag,comm, req(3),ierror)
541          call MPI_ISEND(phi(is+b1,ks+b1),1,horizontal_border,
542       f                 lower,lower_left_tag,comm, req(4),ierror)
543          call MPI_WAITALL(4, req, sts_upper, ierror)

550          call MPI_IRECV(phi(is,ks+b1),1,vertical_border,
551       f                 left,MPI_ANY_TAG,comm, req(2),ierror)
552          call MPI_IRECV(phi(ie,ks+b1),1,vertical_border,
553       f                 right,MPI_ANY_TAG,comm, req(1),ierror)
554          call MPI_ISEND(phi(ie-b1,ks+b1),1,vertical_border,
555       f                 right,upper_right_tag,comm, req(3),ierror)
556          call MPI_ISEND(phi(is+b1,ks+b1),1,vertical_border,
557       f                 left,lower_left_tag,comm, req(4),ierror)
558          call MPI_WAITALL(4, req, sts_right, ierror)
```

Heat: MPI_SENDRECV with local copying (communica.6)

```
765          double precision phiright(ks+1:ke-1), phirecvright(ks+1:ke-1)
766          double precision phileft(ks+1:ke-1),  phirecvleft(ks+1:ke-1)

774              call MPI_SENDRECV(phi(is+1,ke-1),iinner,
775           f                    MPI_DOUBLE_PRECISION,upper,upper_right_tag,
776           f                    phi(is+1,ks),iinner,
777           f                    MPI_DOUBLE_PRECISION,lower,MPI_ANY_TAG,
778           f                    comm,sts_lower,ierror)
779              call MPI_SENDRECV(phi(is+1,ks+1),iinner,
780           f                    MPI_DOUBLE_PRECISION,lower,lower_left_tag,
781           f                    phi(is+1,ke),iinner,
782           f                    MPI_DOUBLE_PRECISION,upper,MPI_ANY_TAG,
783           f                    comm,sts_upper,ierror)

790              phileft(ks+1:ke-1)=phi(is+1,ks+1:ke-1)
791              phiright(ks+1:ke-1)=phi(ie-1,ks+1:ke-1)
792              call MPI_SENDRECV(phiright(ks+1),kinner,
793           f                    MPI_DOUBLE_PRECISION,right,upper_right_tag,
794           f                    phirecvleft(ks+1),kinner,
795           f                    MPI_DOUBLE_PRECISION,left,MPI_ANY_TAG,
796           f                    comm,sts_left,ierror)
797              call MPI_SENDRECV(phileft(ks+1),kinner,
798           f                    MPI_DOUBLE_PRECISION,left,lower_left_tag,
799           f                    phirecvright(ks+1),kinner,
800           f                    MPI_DOUBLE_PRECISION,right,MPI_ANY_TAG,
801           f                    comm,sts_right,ierror)
802              if (left.ne.MPI_PROC_NULL)
803           f    phi(is,ks+1:ke-1)=phirecvleft(ks+1:ke-1)
804              if (right.ne.MPI_PROC_NULL)
805           f    phi(ie,ks+1:ke-1)=phirecvright(ks+1:ke-1)
```


Heat: MPI_ALLTOALLV with local copying (label 207)

```
448          phileft(ks+1:ke-1)=phi(is+1,ks+1:ke-1)
449          phiright(ks+1:ke-1)=phi(ie-1,ks+1:ke-1)

450  c       ... to prevent compiler optimizations because the compiler
451  c           does not see the input arguments of ALLTOALLV:
452          call MPI_ADDRESS(phi,iaddr_dummy,ierror)       <- horizontal inner borders
453          call MPI_ADDRESS(phileft,iaddr_dummy,ierror)   <- left vertical inner border
454          call MPI_ADDRESS(phiright,iaddr_dummy,ierror)  <- right vertical inner border

455          call MPI_ALLTOALLV(
456           f       MPI_BOTTOM, sendcounts, sdispls, MPI_DOUBLE_PRECISION,
457           f       MPI_BOTTOM, recvcounts, rdispls, MPI_DOUBLE_PRECISION,
458           f       comm, ierror)

459  c       ... to prevent compiler optimizations because the compiler
460  c           does not see the output arguments of ALLTOALLV:
461          call MPI_ADDRESS(phi,iaddr_dummy,ierror)          <- horizontal halos
462          call MPI_ADDRESS(phirecvleft,iaddr_dummy,ierror)  <- left vertical halo
463          call MPI_ADDRESS(phirecvright,iaddr_dummy,ierror)<- right vertical halo

464          if (left.ne.MPI_PROC_NULL)
465           f  phi(is,ks+1:ke-1)=phirecvleft(ks+1:ke-1)
466          if (right.ne.MPI_PROC_NULL)
467           f  phi(ie,ks+1:ke-1)=phirecvright(ks+1:ke-1)
```

arrays: entries for each direction

Heat: MPI_ALLTOALLV with local copying (initialization)

```
224          call MPI_TYPE_EXTENT(MPI_DOUBLE_PRECISION, dp_size, ierror)
225          do i=0, size-1
226            sendcounts(i) = 0
227            sdispls(i) = 0
228            recvcounts(i) = 0
229            rdispls(i) = 0
233          enddo

234          if (upper.ne.MPI_PROC_NULL) then
235            sendcounts(upper) = iinner
236            call MPI_ADDRESS(phi(is+1,ke-1), sdispls(upper), ierror)
237            sdispls(upper)  = sdispls(upper) / dp_size
238            recvcounts(upper) = iinner
239            call MPI_ADDRESS(phi(is+1,ke)  , rdispls(upper), ierror)
240            rdispls(upper)  = rdispls(upper) / dp_size
241          endif

242-249: same with "lower",  ks+1 / ks instead of ke-1 / ke

250          if (right.ne.MPI_PROC_NULL) then
251            sendcounts(right) = kinner
252            call MPI_ADDRESS(phiright(ks+1), sdispls(right), ierror)
253            sdispls(right)  = sdispls(right) / dp_size
254            recvcounts(right) = kinner
255            call MPI_ADDRESS(phirecvright(ks+1), rdispls(right), ierror)
256            rdispls(right)  = rdispls(right) / dp_size
257          endif

258-265: same with "left"
```


Heat: Communication Efficiency

Communication-time (number of PEs)	T3E (16)	Hitachi (16)	HP-V (8)	
MPI non-blocking	3.4	**5.3**	**2**	[sec]
MPI_SENDRECV	1.9	5.8	**2**	[sec]
MPI_ALLTOALLV	**0.8 *)**	15.4	**2**	[sec]
Computation-Time	0.65	1.9	2	[sec]

MPI targets portable and efficient message-passing programming
but
efficiency of MPI application-programming is **not portable**!

***) up 128 PEs: ALLTOALLV is better than SENDRECV**

(measured April 29, 1999, with heat-mpi1-big.f and stride 179,
CRAY T3E: sn6715 hwwt3e.hww.de 2.0.4.48 unicosmk CRAY T3E mpt.1.3.0.0.6,
Hitachi: HI-UX/MPP hitachi.rus.uni-stuttgart.de 02-03 0 SR2201,
HP: HP-UX hp-v.hww.de B.11.00 A 9000/800 75859)

Heat: Parallel Abort Criterion

- Three methods:
  - MPI_ALLREDUCE
    - exact implementation of the sequential code
    - very expensive
    - less expensive, if not computed in each iteration: using variable `stride`
    - computing efficient `stride` needs knowledge about number of iterations, execution time of MPI_ALLREDUCE, and execution time of one iteration without allreduce: $stride_{optimal}$ = `sqrt(iterations*time_allreduce/time_exec)`
  - Computing **local** abort-criterion
    - using the MPI message tag to send the criterion to the neighbors
    - problems if abort-criterion is locally fulfilled on some PEs in the first iterations, see (1) on next slide
  - Computing **global** abort-criterion by communicating the local abort-criterion via message tags from lowest to highest node and backward

Heat: Parallel Abort Criterion - local & tags

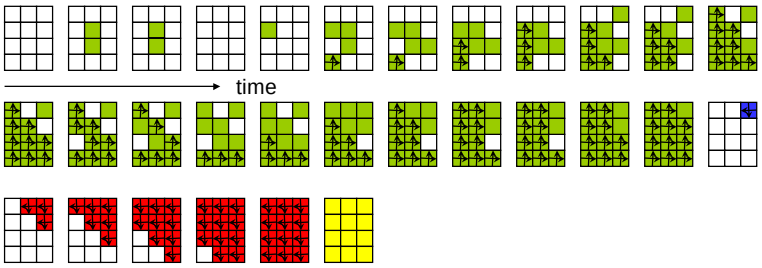
(3)(4) done after, (1)(2) i.e. in the next iteration

```
331            if (done_local.eq.0) then
...
348            endif

364  102        continue
365  c          ... end_method == 2
366              if (done_local.eq.1) then
367  c             ... After sending "done_local" in the last iteration:
368  c                 if a neighbor is also in the state 'done_local' then
369  c                 the neighbor now knows that this node is also in
370  c                 'done_local' and both can stop to communicate:
371                if (sts_upper(MPI_TAG).eq.1) upper = MPI_PROC_NULL
372                if (sts_lower(MPI_TAG).eq.1) lower = MPI_PROC_NULL
373                if (sts_right(MPI_TAG).eq.1) right = MPI_PROC_NULL
374                if (sts_left (MPI_TAG).eq.1) left  = MPI_PROC_NULL
375  c             ... if there is no more any local computation
376  c                 nor communication then the process can be stopped:
377                if ((upper.eq.MPI_PROC_NULL).and.(lower.eq.MPI_PROC_NULL)
378           f    .and.(right.eq.MPI_PROC_NULL).and.(left.eq.MPI_PROC_NULL))
379           f      goto 110
380              endif
381  c           ... "reached_local" means that the heat wave has reached
382  c               the local area of phi.
383  c               To test "reached_local" is necessary to prohibit that
384  c               a node stops before it has computed anything due
385  c               to its starting values that are all zero.
386              if (dphimax.gt.eps) reached_local = 1
387              if ((reached_local.eq.1).and.(dphimax.lt.eps)) then
388                done_local = 1
389                lower_left_tag = 1
390                upper_right_tag = 1
391              endif
```

Heat: Parallel Abort Criterion - global & tags

```
394  103        continue
395  c          ... end_method == 3
396              upper_right_tag = 0
397  c          ... outgoing from node (0,0) the local_done criterion
398  c              is propageded with AND operation to the upper-right node
399              if ((dphimax.lt.eps) .and.
400         f (1)  ((lower.eq.MPI_PROC_NULL).or.(sts_lower(MPI_TAG).eq.1))
401         f      .and.((left.eq.MPI_PROC_NULL).or.(sts_left(MPI_TAG).eq.1)))
402         f        upper_right_tag = 1
403  c          ... after the local_done criterion has reached at the
404  c              uppermost-rightmost node in the interval of the
405  c              last (idim+kdim) iterations, it is sent back to
406  c              the node (0,0) to inform all nodes about the global stop
407              if ((icoord.eq.idim-1) .and. (kcoord.eq.kdim-1)) then
408               if ((dphimax.lt.eps) .and.
409         f (2)  ((lower.eq.MPI_PROC_NULL).or.(sts_lower(MPI_TAG).eq.1))
410         f      .and.((left.eq.MPI_PROC_NULL).or.(sts_left(MPI_TAG).eq.1)))
411         f        lower_left_tag = 1
412              else
413               if (((upper.eq.MPI_PROC_NULL).or.(sts_upper(MPI_TAG).eq.1))
414         f      .and.
415         f (3)  ((right.eq.MPI_PROC_NULL).or.(sts_right(MPI_TAG).eq.1)))
416         f        lower_left_tag = 1
417              endif
418              it_stopping = it_stopping + lower_left_tag
419  c          ... then the global stop is done synchronously after all
420  c       (4)    nodes are informed:
421              if (it_stopping .eq. (icoord+kcoord+1)) goto 110
422              goto 199
```


Heat: Parallel Abort Criterion - global & tags (cont.)

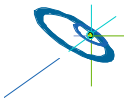


- Computing,
- and abort criterion fulfilled locally,
- and received from left and lower node: Now sending to right and upper node, (1)
- Top-node: Criterion fulfilled locally and received from left and lower node: Now sending stopping-tag to lower and left node. (2)
- Stopping-tag received and now sending to lower and left node. (3)
- Synchronous stopping after all nodes have received the stopping-tag. (4)

—> At maximum, $2*(dim_1+dim_2-2)$ additional iterations!

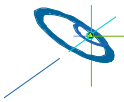
Heat: Summary (1)

- block data decomposition
- communication: filling the halo with
  - non-blocking point-to-point
  - blocking MPI_SENDRECV
  - MPI_ALLTOALLV
- and for the abort-criterion and interactive input
  - MPI_ALLREDUCE
  - usage of message tags in point-to-point communication
  - MPI_BCAST
- derived datatypes: MPI_TYPE_VECTOR



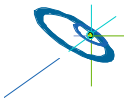
Heat: Summary (2)

- absolute addressing with MPI_ADDRESS and MPI_BOTTOM and problems with Fortran language binding and MPI (see MPI-2, pages 289f)
- MPI topologies: MPI_CART_CREATE / _COORDS / _SHIFT
- Null processes for not existing neighbors: MPI_PROC_NULL see MPI 1.1, Chapter 3.11, pages 60f)
- timer: MPI_WTIME
- performance of different communication methods



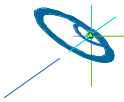
Practical (a)

- cp ~/MPI/course/F/Ch7/ring.f .
  C ring.c .
- make two-dimensional topology
  - splitting “size” with MPI_DIMS_CREATE()
  - cyclic in the first dimension
  - linear in the second dimension
- compute and print the sum of the original Cartesian ranks separately in each ring
- please, discuss and implement the best choice for large scale systems, i.e., expect that the computation is repeated very often and hundreds of processors are used
- your trainer will come to look at your decisions



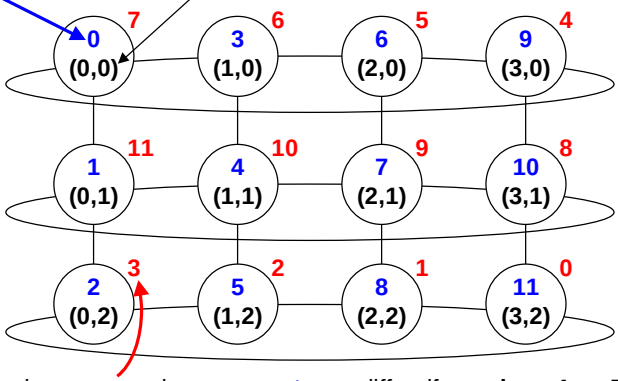
Practical (a) — Background, I.

- MPI_Dims_create:
  - int MPI_Dims_create(int nnodes, int ndims, int *dims);
  - SUBR. MPI_DIMS_CREATE(nnodes, ndims, dims, ierror)
    INTEGER nnodes, ndims, dims(ndims), ierror
  - ndims := number of dimensions in dims, e.g., := 2
  - dims(...) must be initialized with zero, e.g., (0,0)
  - nnodes := size of MPI_COMM_WORLD, e.g., := 12
  - result: dims contains a balanced distribution, e.g., (4,3)
- MPI_Cart_create:
  - „dims“ and „periods“ are now arrays!
- Expected results:
  - size=4 => dims=(2,2) => sums = (2,4)
  - size=6 => dims=(3,2) => sums = (6,9)
  - size=12 => dims=(4,3) => sums = (18,22,26)



Practical (a) — Background, II.

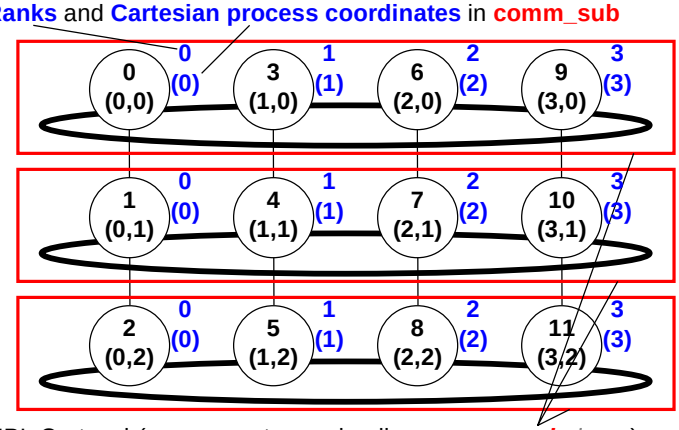
- **Ranks** and **Cartesian process coordinates** in **comm_cart**



- Ranks in **comm** and **comm_cart** may differ, if **reorder = 1** or **.TRUE.**
- This reordering can allow MPI to optimize communications

Practical (a) — Background, III.

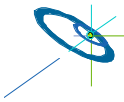
- **Ranks** and **Cartesian process coordinates** in **comm_sub**



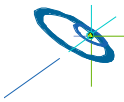
- MPI_Cart_sub(comm_cart, remain_dims, ***comm_sub***, *ierror*)
  (true,false)

Heat: Practical (b)

- cp ~/MPI/course/F/heat/* .
- Compile heat-mpi0-big.f and run it on 1, 4, 8 and 9 PEs and notify the execution time
- Compute the speedup and efficiency (e.g. with xcalc)
- Which speedup and efficiency do you expect for 16 PEs?
- Switch on the printing of the result (line 87)
- Compile heat-mpi0-big.f and run it on 4 PEs and verify the output



Heat: Files for Practical (b)

All examples with 80x80 array

- ~/MPI/course/F/heat/heat-big.f
  heat-mpi0-big.f
  heat-mpi1-big.f
- ~/MPI/course/C/heat/heat-big.c
  heat-mpi-slow-big.c
  heat-mpi-fast-big.c