- groups and communicators
- virtual topologies
- derived datatypes

Introduction to the Message Passing Interface (MPI)

Rolf Rabenseifner
rabenseifner@hlrs.de

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

(for MPI-2.1, MPI-2.2, MPI-3.0, MPI-3.1, and MPI-4.0)

H L R S

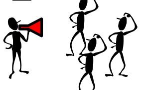
© 2000-2022 HLRS, Rolf Rabenseifner

● REC → online

Groups, Comm., Environment

Chap.8 Groups & Communicators, Environment managem.

1. MPI Overview
2. Process model and language bindings
3. Messages and point-to-point communication
4. Nonblocking communication
5. The New Fortran Module mpi_f08
6. Collective communication
7. Error Handling



MPI_Init()
MPI_Comm_rank()

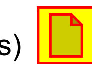
8. Groups & communicators, environment management

- **(1) MPI_Comm_split, intra- & inter-communicators**
- **(2) Re-numbering on a cluster, collectives on inter-communicators, info object, naming & attribute caching, implementation information, Sessions Model**

9. Virtual topologies
10. One-sided communication
11. Shared memory one-sided communication
12. Derived datatypes
13. Parallel file I/O
14. MPI and threads
15. Probe, Persistent Requests, Cancel
16. Process creation and management
17. Other MPI features
18. Best Practice

put
get

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

Goals

Support for libraries or application sub-spaces

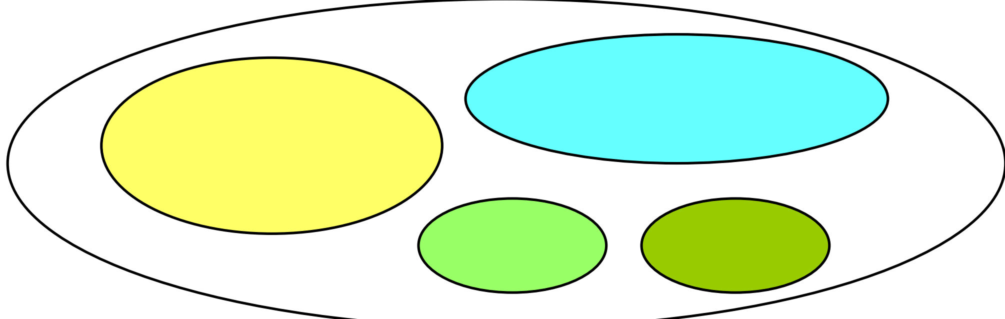
- Safe communication context spaces
  - **e.g., for subsets of processes,**
  - **or duplicated communicators for independent software layers (middle-ware)**

A library should always use a duplicate of MPI_COMM_WORLD, and never MPI_COMM_WORLD itself.

- Collective operations (→course Chapter 6) on a subset of processes
- Re-numbering of the ranks of communicators
- Inter-communicators
- Info handles
- Naming of context spaces
- Add additional user-defined attributes to a communication context
- Inquiry methods
- *World Model* and *Sessions Model*

New in MPI-4.0

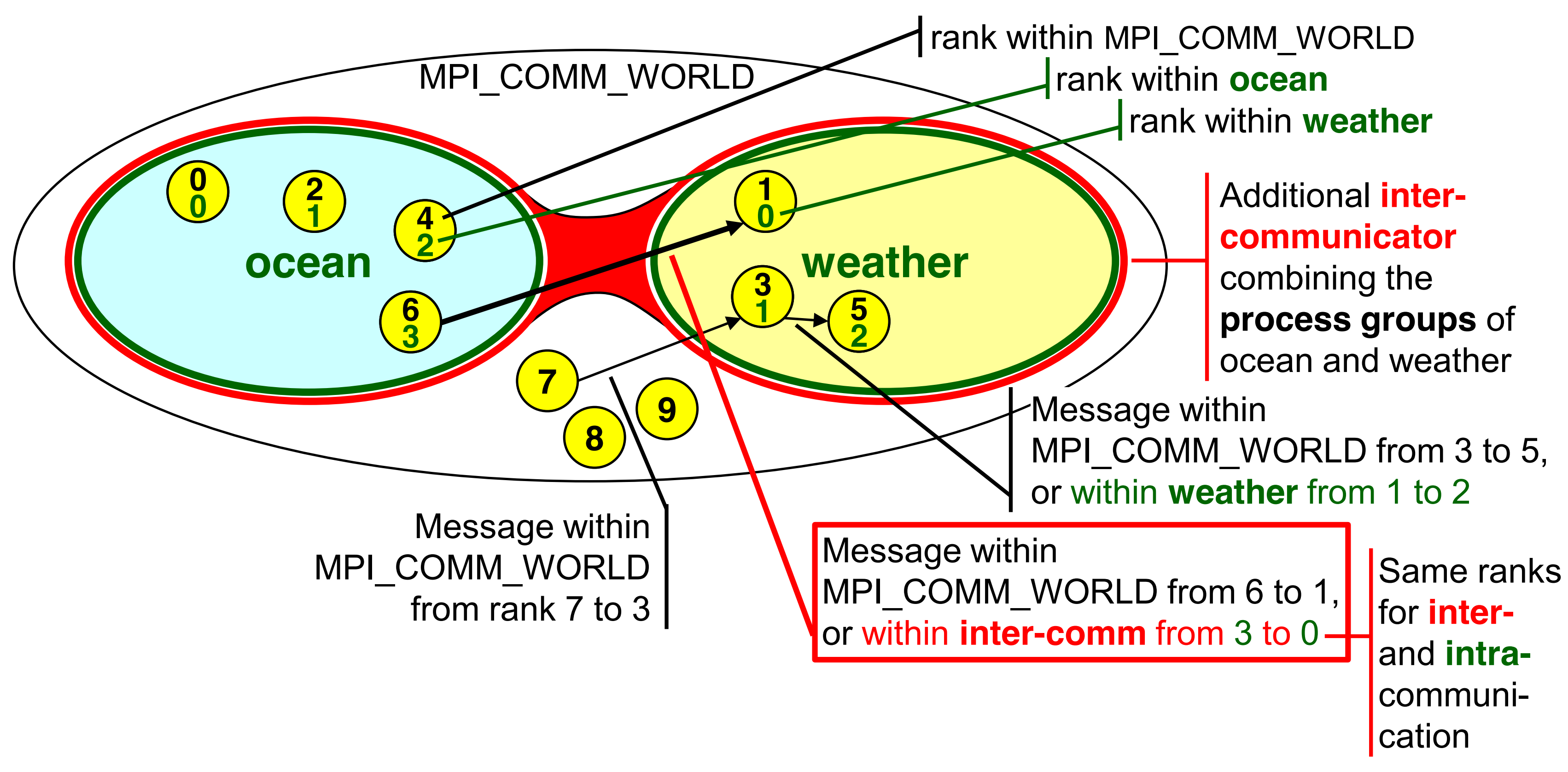
→ Section (2) of this course chapter



New in MPI-4.0

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

Methods – e.g., for coupled applications

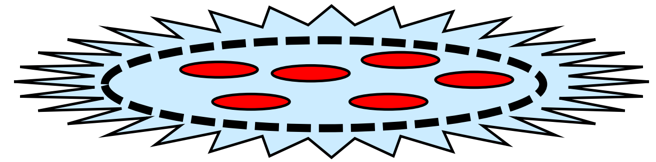


- **Sub-communicators:** Collectively defined communication sub-spaces
- **Intra-** and **inter-communicators**

Perfect for any communication between processes of the two groups (ocean and weather)

© 2000-2022 HLRS, Rolf Rabenseifner

Sub-groups and sub-communicators (1)



Two levels:

- Group of processes
  - **Without the ability to communicate**
  - **Local routines to build group & sub-sets**
  - **Same ranks as in related communicator**
- Communicators
  - **Group of processes with additional ability to communicate**

Scalability problems when handling many processes in each process

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

Sub-groups and sub-communicators (2)

- New sub-communicators via sub-groups

The old (original) communicator

MPI_Comm_group

Many **MPI_Group_…** routines to establish sub-groups

Two methods to establish the new sub-communicator(s):
- **MPI_Comm_create** (can establish **many** disjunctive sub-comms, collective over the original communicator)
- **MPI_Comm_create_group** (collective over the sub-group)

Several sub-comms in one call since MPI-2.2

New in MPI-3.0

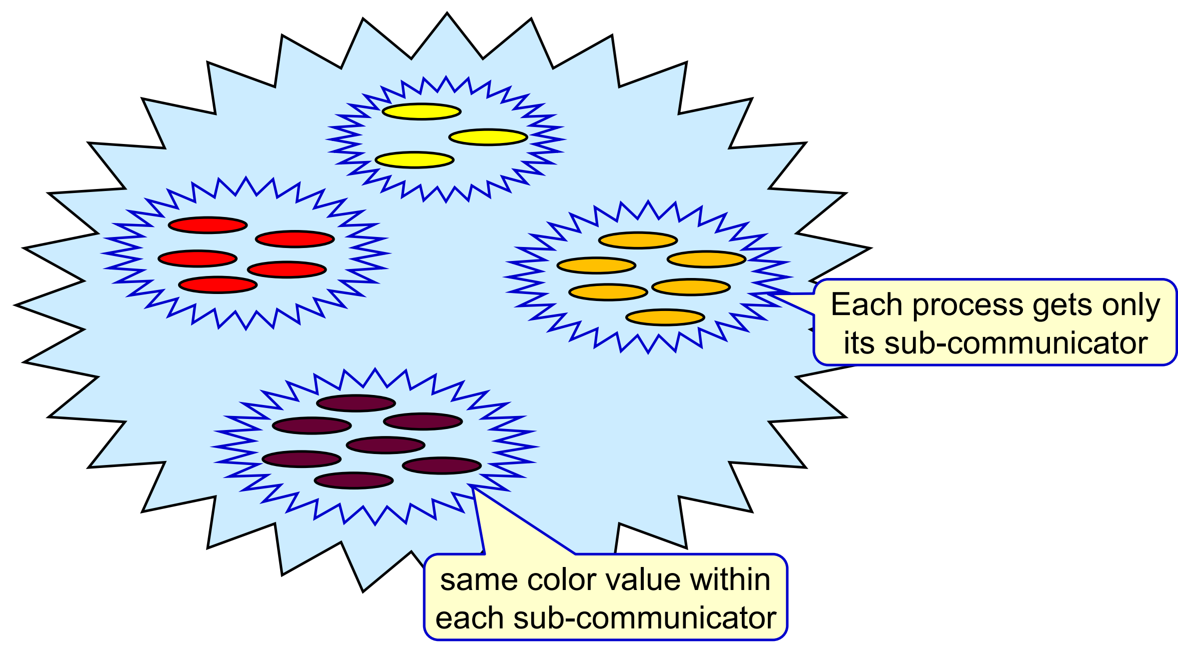
© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

Sub-groups and sub-communicators (3)

- New sub-communicators via MPI_Comm_split

& MPI_Comm_split_type ⟵ New in MPI-3.0

→ course Chapter 11

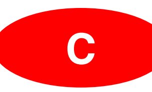


© 2000-2022 HLRS, Rolf Rabenseifner REC → online

Example: MPI_Comm_split()

Creation is **collective** in the **old** communicator.

All processes with same color are grouped into separate sub-communicators



- C/C++: int MPI_Comm_split (MPI_Comm comm, int color, int key, MPI_Comm **newcomm*)

Fortran

- Fortran: MPI_COMM_SPLIT (comm, color, key, *newcomm*, ierror)

  mpi_f08: TYPE(MPI_Comm) :: comm, newcomm
  INTEGER :: color, key;
  INTEGER, OPTIONAL :: ierror

  mpi & mpif.h: INTEGER comm, color, key, newcomm, ierror

Python

- Python: *newcomm* = comm.Split(color=0, key=0)

Each process gets only its own sub-communicator

Example:

```
int my_rank, mycolor, key, my_newrank;
MPI_Comm newcomm;
MPI_Comm_rank (MPI_COMM_WORLD, &my_rank);
mycolor = my_rank/4;
key = 0;
MPI_Comm_split(MPI_COMM_WORLD, mycolor, key, &newcomm);
MPI_Comm_rank (newcomm, &my_newrank);
```

Always 4 process get same color → grouped in an own newcomm

key==0 → ranking in newcomm is sorted as in old comm
key ≠ 0 → ranking in newcomm is sorted according key values

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 … MPI_COMM_WORLD

0 1 2 3	0 1 2 3	0 1 2 3	0 1 2 3	0 1 2 3
newcomm mycolor == 0	newcomm mycolor == 1	newcomm mycolor == 2	newcomm mycolor == 3	newcomm mycolor == 4

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

Example: MPI_Group_range_incl() + MPI_Comm_create()

```
int my_rank, mycolor, my_newrank, ranges[1][3];
MPI_Group world_group, sub_group;
MPI_Comm newcomm;

MPI_Comm_rank (MPI_COMM_WORLD, &my_rank);
MPI_Comm_group(MPI_COMM_WORLD, &world_group)

mycolor = my_rank/4;
/* first rank of my range:*/ ranges[0][0] = mycolor*4;
/* last  rank of my range:*/ ranges[0][1] = mycolor*4 + (4-1);
/* stride of ranks:       */ ranges[0][2] = 1;
MPI_Group_range_incl ( world_group, 1, ranges, &sub_group);

MPI_Comm_create(MPI_COMM_WORLD, sub_group, &newcomm);
MPI_Comm_rank (newcomm, &my_newrank);
```

Only one range

Three values per range:
[0]: first rank
[1]: last rank
[2]: stride

Group of the processes in MPI_COMM_WORLD.
Group and sub-group creation is **local** (non-collective).

Always 4 process get same color
→ **grouped in an own sub_group**
→ grouped in an own newcomm

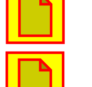
Must be restricted to **< num_procs**

(Sub-)communicator creation is **collective**.

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 ... MPI_COMM_WORLD

newcomm (0 1 2 3)	newcomm (0 1 2 3)	newcomm (0 1 2 3)	newcomm (0 1 2 3)	newcomm (0 1 2 3)
mycolor == 0	mycolor == 1	mycolor == 2	mycolor == 3	mycolor == 4

Move to next course chapter, i.e., skip practical and (2)=advanced topics (13 slides)



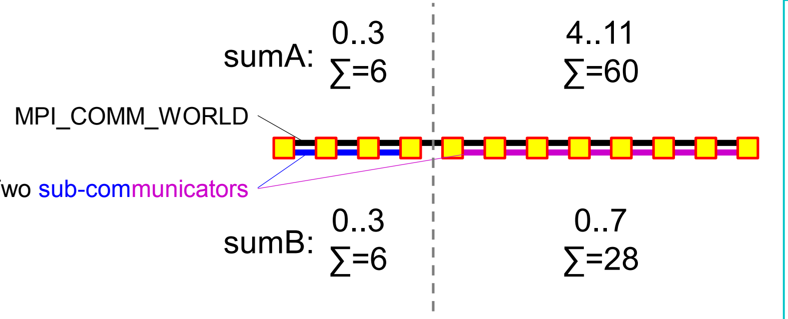
Skip practical, move to (2)=advanced topics

© 2000-2022 HLRS, Rolf Rabenseifner

Exercise 1 — Two independent sub-communicators

In MPI/tasks/…

- Use C C/Ch8/comm-split-skel.c or Fortran F_30/Ch8/comm-split-skel_30.f90 or Python PY/Ch8/comm-split-skel.py
- Modify the *allreduce* program:
  - Split the communicator into 1/3 and 2/3, e.g., with color = (rank > $\left\lfloor\frac{\text{size}-1}{3}\right\rfloor$) as input for **MPI_Comm_split**
  - Calculate **sumA** and **sumB** over all processes within each **sub-communicator**
  - sumA: **ranks in MPI_COMM_WORLD** (but summed up only within each sub-communicator)
    - E.g., with 12 processes → split into 4 & 8 with world ranks 0..3 & 4..11 and sums 6 & 60 → sumA
  - sumB: **ranks in new sub-communicators** (and summed up only within each sub-comm.)
    - E.g., with 12 processes → split into 4 & 8 with sub-comm ranks 0..3 & 0..7 and sums 6 & 28 → sumB
  - Use mpirun ….. | sort +2n -3



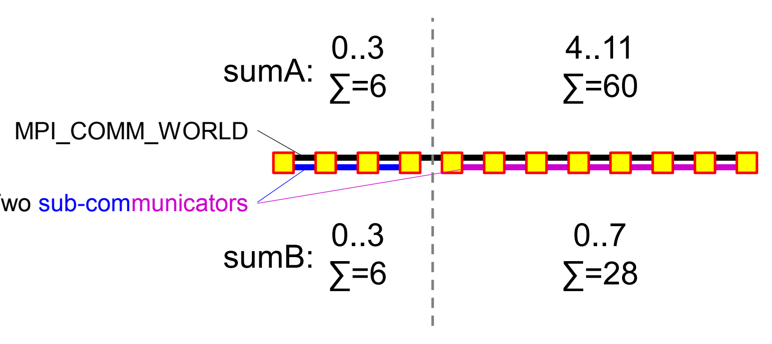

```
Expected results with 12 processes:
PE world:  0, color=0 sub:  0, SumA=  6, SumB=  6 in sub_comm
PE world:  1, color=0 sub:  1, SumA=  6, SumB=  6 in sub_comm
PE world:  2, color=0 sub:  2, SumA=  6, SumB=  6 in sub_comm
PE world:  3, color=0 sub:  3, SumA=  6, SumB=  6 in sub_comm
PE world:  4, color=1 sub:  0, SumA= 60, SumB= 28 in sub_comm
PE world:  5, color=1 sub:  1, SumA= 60, SumB= 28 in sub_comm
PE world:  6, color=1 sub:  2, SumA= 60, SumB= 28 in sub_comm
PE world:  7, color=1 sub:  3, SumA= 60, SumB= 28 in sub_comm
PE world:  8, color=1 sub:  4, SumA= 60, SumB= 28 in sub_comm
PE world:  9, color=1 sub:  5, SumA= 60, SumB= 28 in sub_comm
PE world: 10, color=1 sub:  6, SumA= 60, SumB= 28 in sub_comm
PE world: 11, color=1 sub:  7, SumA= 60, SumB= 28 in sub_comm
```

Exercises 1+2

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

Exercise 2 (advanced) — MPI_Comm_create

- Use C C/Ch8/comm-create-skel.c or Fortran F_30/Ch8/comm-create-skel_30.f90 or Python PY/Ch8/comm-create-skel.py
- Same as Exercise 1, but with **MPI_Comm_group()**, **MPI_Group_range_incl()**, and **MPI_Comm_create()**
  - instead of MPI_Comm_split()
  - Two different ranges for color 0 and 1 !!!
  - Same results in sumA/B as in Exercise 1
- Same details as in Exercise 1:
  - Split the communicator into 1/3 and 2/3, e.g., with color = (rank > $\left\lfloor \frac{\text{size}-1}{3} \right\rfloor$)
  - Calculate **sumA** and **sumB** over all processes within each **sub-communicator**
  - sumA: **ranks in MPI_COMM_WORLD** (but summed up only within each sub-communicator)
  - sumB: **ranks in new sub-communicators** (and summed up only within each sub-comm.)
  - Use mpirun ….. | sort +2n -3




```
Expected results with 12 processes:
PE world:  0, color=0 sub:  0, SumA=  6, SumB=  6 in sub_comm
PE world:  1, color=0 sub:  1, SumA=  6, SumB=  6 in sub_comm
PE world:  2, color=0 sub:  2, SumA=  6, SumB=  6 in sub_comm
PE world:  3, color=0 sub:  3, SumA=  6, SumB=  6 in sub_comm
PE world:  4, color=1 sub:  0, SumA= 60, SumB= 28 in sub_comm
PE world:  5, color=1 sub:  1, SumA= 60, SumB= 28 in sub_comm
PE world:  6, color=1 sub:  2, SumA= 60, SumB= 28 in sub_comm
PE world:  7, color=1 sub:  3, SumA= 60, SumB= 28 in sub_comm
PE world:  8, color=1 sub:  4, SumA= 60, SumB= 28 in sub_comm
PE world:  9, color=1 sub:  5, SumA= 60, SumB= 28 in sub_comm
PE world: 10, color=1 sub:  6, SumA= 60, SumB= 28 in sub_comm
PE world: 11, color=1 sub:  7, SumA= 60, SumB= 28 in sub_comm
```

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

Chapter 8-(1), Exercise1: Two independent sub-communicators

C — MPI/tasks/C/Ch8/solutions/comm-split.c

```
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
MPI_Comm_rank(MPI_COMM_WORLD, &my_world_rank);
mycolor = (my_world_rank > (world_size-1)/3);
/* This definition of mycolor implies that the first color is 0 */
MPI_Comm_split(MPI_COMM_WORLD, mycolor, 0, &sub_comm);
MPI_Comm_size(sub_comm, &sub_size);
MPI_Comm_rank(sub_comm, &my_sub_rank);
MPI_Allreduce (&my_world_rank, &sumA, 1, MPI_INT, MPI_SUM, sub_comm);
MPI_Allreduce (&my_sub_rank,   &sumB, 1, MPI_INT, MPI_SUM, sub_comm);
printf ("PE world:%3i, color=%i sub:%3i, SumA=%3i, SumB=%3i in sub_comm\n",
        my_world_rank, mycolor, my_sub_rank, sumA, sumB);
```

Fortran — MPI/tasks/F_30/Ch8/solutions/comm-split_30.f90

```
CALL MPI_Comm_size(MPI_COMM_WORLD, world_size)
CALL MPI_Comm_rank(MPI_COMM_WORLD, my_world_rank)
IF (my_world_rank>(world_size-1)/3) THEN; mycolor=1; ELSE; mycolor=0; ENDIF
! This definition of mycolor implies that the first color is 0
CALL MPI_Comm_split(MPI_COMM_WORLD, mycolor, 0, sub_comm)
CALL MPI_Comm_size(sub_comm, sub_size)
CALL MPI_Comm_rank(sub_comm, my_sub_rank)
CALL MPI_Allreduce (my_world_rank,sumA, 1, MPI_INTEGER, MPI_SUM, sub_comm);
CALL MPI_Allreduce (my_sub_rank,  sumB, 1, MPI_INTEGER, MPI_SUM, sub_comm);
write(*,'(''PE world:'',I3,'', color='',I3,'' sub:'',I3,'' SumA='',I5,'' SumB='',I5,'' in subcomm'')')&
  &      my_world_rank,     mycolor,   my_sub_rank,   sumA,       sumB
```

Python — MPI/tasks/PY/Ch8/solutions/comm-split.py

```
sub_comm = comm_world.Split(mycolor, 0)
```

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

Chap.9 Virtual Topologies

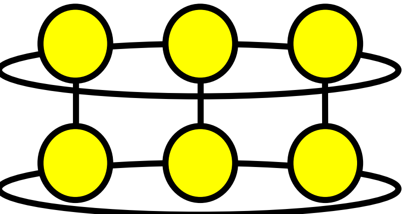
Virtual Topologies

1. MPI Overview
2. Process model and language bindings
3. Messages and point-to-point communication
4. Nonblocking communication
5. The New Fortran Module mpi_f08
6. Collective communication
7. Error Handling
8. Groups & communicators, environment management

9. Virtual topologies

- **(1) A multi-dimensional process naming scheme**
- **(2) Neighborhood communication + MPI_BOTTOM**
- **(3) Optimization through reordering**

10. One-sided communication
11. Shared memory one-sided communication
12. Derived datatypes
13. Parallel file I/O
14. MPI and threads
15. Probe, Persistent Requests, Cancel
16. Process creation and management
17. Other MPI features
18. Best Practice



© 2000-2022 HLRS, Rolf Rabenseifner

Example

- Global data array A(1:3000, 1:4000, 1:500) = 6•10^9 words
- on **3 x 4 x 5 = 60 processes**
- process coordinates **0..2, 0..3, 0..4**

- example:
  on process **ic_0=2, ic_1=0, ic_2=3 (rank=43)**
  decomposition, e.g., A(2001:3000, 1:1000, 301:400) = 0.1•10^9 words

- **process coordinates:** handled with **virtual Cartesian topologies**
- Array decomposition: handled by the application program directly

© 2000-2022 HLRS, Rolf Rabenseifner

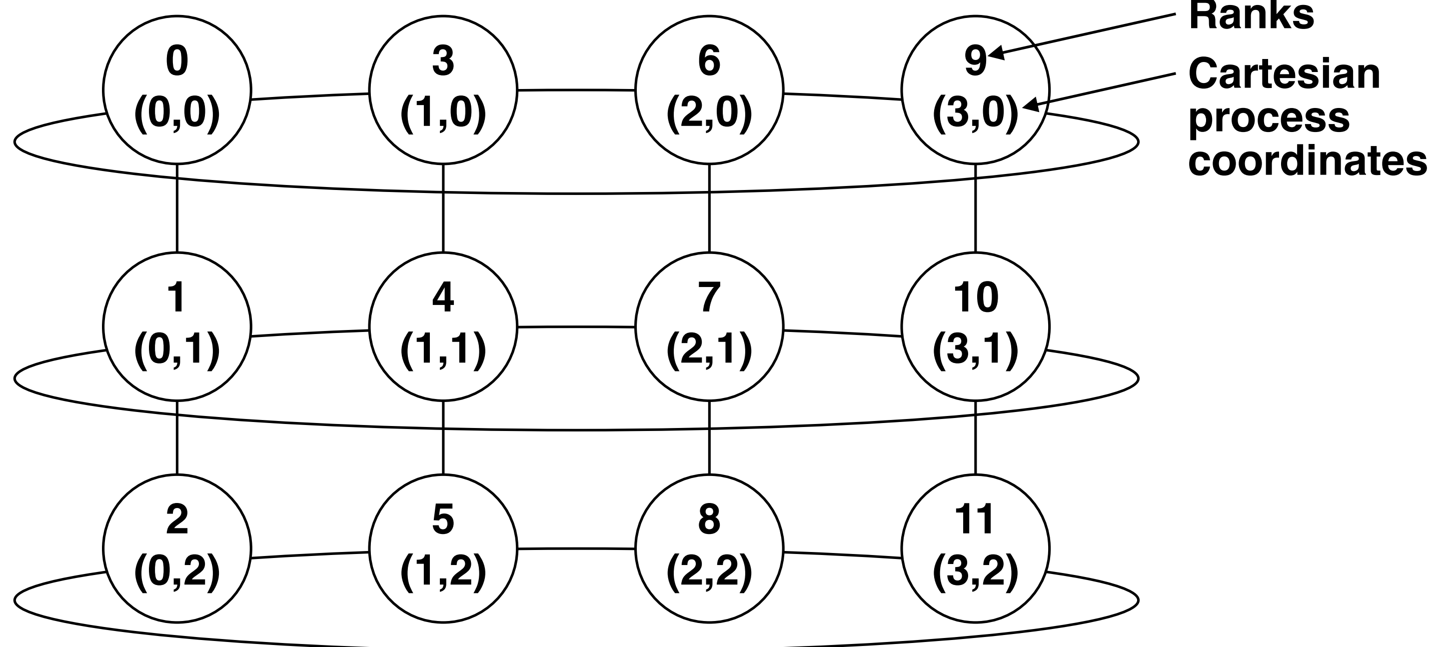
Virtual Topologies

- Convenient process naming.
- Naming scheme to fit the communication pattern.
- Simplifies writing of code.
- Can allow MPI to optimize communications → see course Chapter 9-(3)

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

How to use a Virtual Topology

- Creating a topology produces a new communicator.
- MPI provides mapping functions:
  - to compute process ranks, based on the topology naming scheme,
  - and vice versa.
- Example: 2-dimensional cylinder



© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

Topology Types

- Cartesian Topologies
  - each process is *connected* to its neighbor in a virtual process grid,
  - boundaries can be cyclic, or not,
  - processes are identified by Cartesian coordinates,
  - of course,
    communication between any two processes is still allowed.

- Graph Topologies
  - general graphs,
  - two interfaces:
    - **MPI_Graph_create** (since MPI-1)
    - **MPI_Dist_graph_create_adjacent &**
      MPI_Dist_graph_create (new scalable interface since MPI-2.2)
  - not covered here.
  - See also slides on "**Unstructured Grids**" at the end of course Chapter 9-(3)
  - See also T. Hoefler and M. Snir. 2011. Generic Topology Mapping Strategies for Large-scale Parallel Architectures. In *Proceedings of the 2011 ACM International Conference on Supercomputing (ICS'11)*. ACM, 75–85.

© 2000-2022 HLRS, Rolf Rabenseifner

Creating a Cartesian Virtual Topology

C

- C/C++: int MPI_Cart_create(MPI_Comm comm_old, int ndims, int *dims, int *periods, int reorder, MPI_Comm *comm_cart*)

Fortran

- Fortran: MPI_CART_CREATE(comm_old, ndims, dims, periods, reorder, *comm_cart*, *ierror*)

mpi_f08:

```
TYPE(MPI_Comm)          :: comm_old, comm_cart
INTEGER                 :: ndims, dims(*)
LOGICAL                 :: periods(*), reorder
INTEGER, OPTIONAL       :: ierror
```

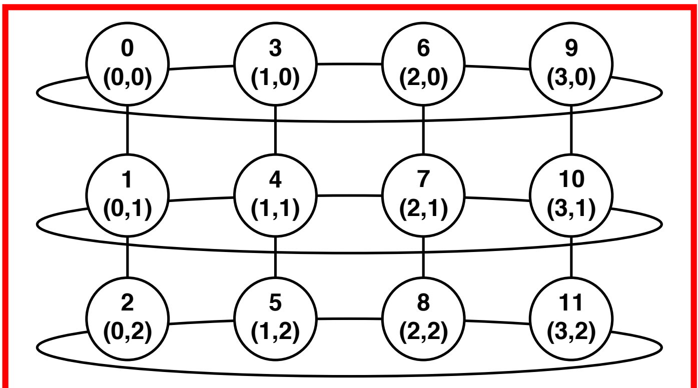
mpi & mpif.h: INTEGER comm_old, ndims, dims(*), comm_cart, ierror ; LOGICAL periods(*), reorder

Python

- Python: *comm_cart* = comm_old.Create_cart(dims, periods, reorder)

see mpi4py.MPI.Intracomm — MPI for Python 3.1.1 documentation

comm_old = MPI_COMM_WORLD
ndims = 2
dims = (4, 3)
periods = (1, 0) (in C)
periods = (.true., .false.) (in Fortran)
reorder = *see next slide*

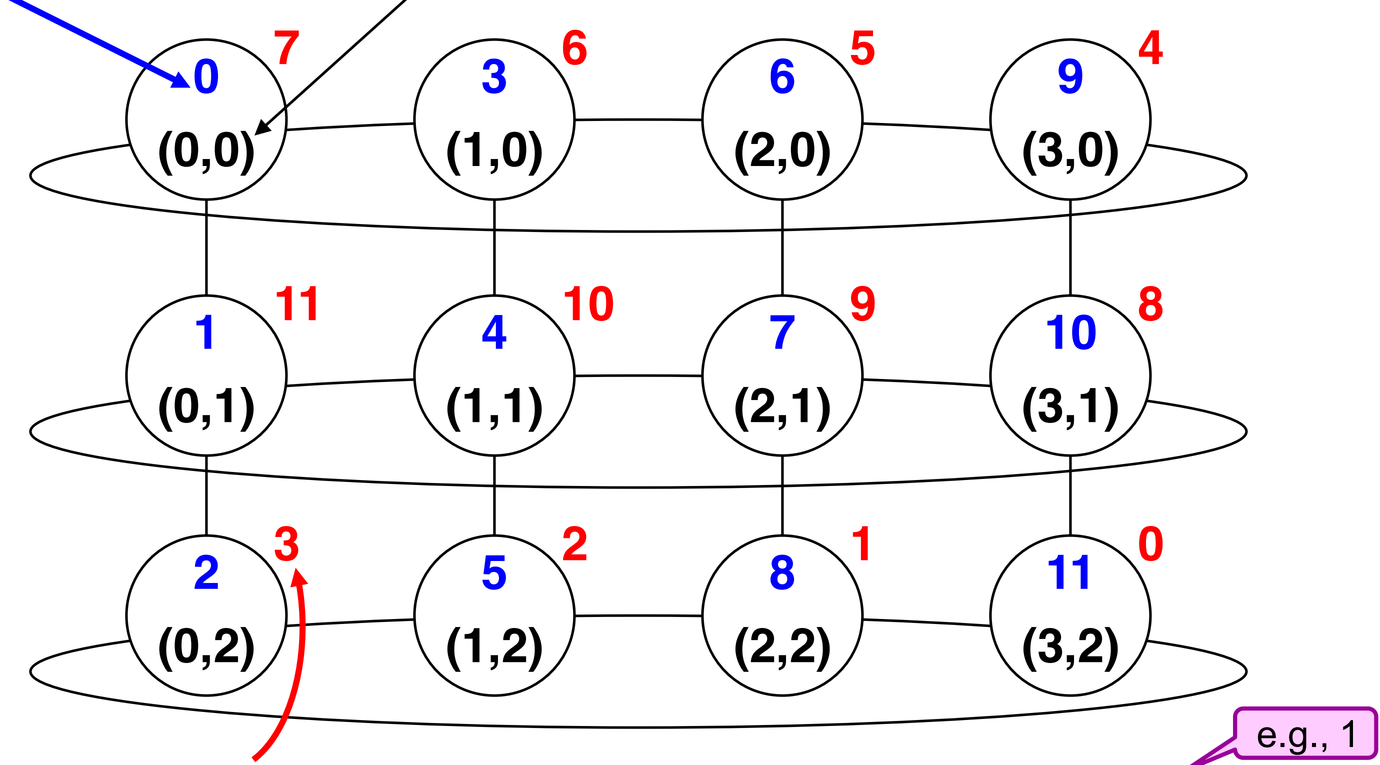


e.g., size==12 factorized with MPI_Dims_create(), see later the slide „**Typical usage of MPI_Cart_create & MPI_Dims_create**" and the advanced exercise 1b

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

Example – A 2-dimensional Cylinder

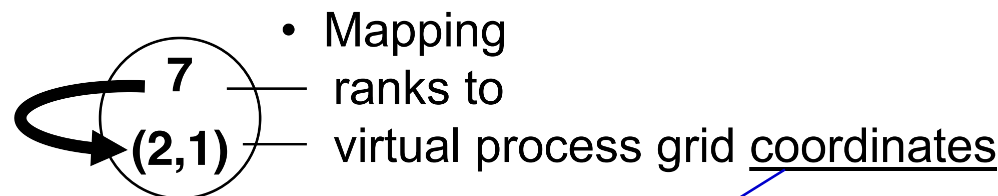
- **Ranks** and **Cartesian process coordinates** in **comm_cart**



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

© 2000-2022 HLRS, Rolf Rabenseifner

Cartesian Mapping Functions



- Mapping ranks to virtual process grid coordinates

C

- C/C++: int MPI_Cart_coords(MPI_Comm comm_cart, int rank, int maxdims, int **coords*)

Fortran

- Fortran: MPI_CART_COORDS(comm_cart, rank, maxdims, *coords*, *ierror)*

```
mpi_f08:      TYPE(MPI_Comm)          :: comm_cart
              INTEGER                 :: rank, maxdims, coords(*)
              INTEGER, OPTIONAL       :: ierror
mpi & mpif.h: INTEGER  comm_cart, rank, maxdims, coords(*), ierror
```

Python

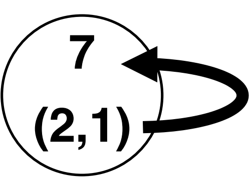
- Python: *coords* = comm_cart.Get_coords(rank)

see mpi4py.MPI.Cartcomm — MPI for Python 3.1.1 documentation

© 2000-2022 HLRS, Rolf Rabenseifner

Cartesian Mapping Functions

- Mapping virtual process grid coordinates to ranks



C

Fortran

Python

- C/C++: int MPI_Cart_rank(MPI_Comm comm_cart, int *coords, int **rank*)

- Fortran: MPI_CART_RANK(comm_cart, coords, *rank*, *ierror*)

  mpi_f08: TYPE(MPI_Comm) :: comm_cart
  INTEGER :: coords(*), rank
  INTEGER, OPTIONAL :: ierror

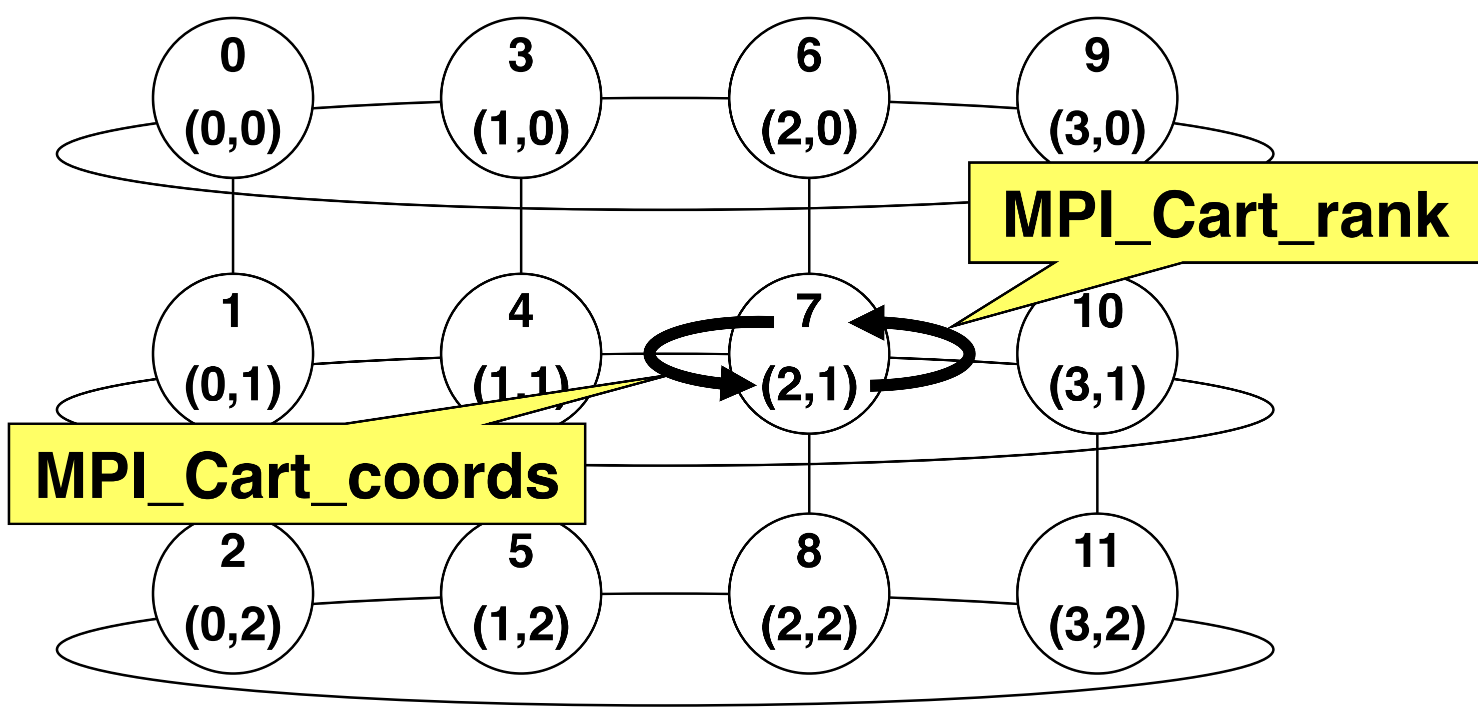
  mpi & mpif.h: INTEGER comm_cart, coords(*), rank, ierror

- Python: *rank* = comm_cart.Get_cart_rank(coords)

  see mpi4py.MPI.Cartcomm — MPI for Python 3.1.1 documentation

© 2000-2022 HLRS, Rolf Rabenseifner

Own coordinates



- Each process gets its own coordinates with (example in Fortran)

  CALL MPI_Comm_rank(comm_cart, ***my_rank***, *ierror*)

  CALL MPI_Cart_coords(comm_cart, **my_rank**, maxdims, ***my_coords***, *ierror*)

Typical usage of MPI_Cart_create & MPI_Dims_create

```
#define ndims 3
int i, nnodes, world_myrank, cart_myrank, dims[ndims], periods[ndims], my_coords[ndims];  MPI_Comm  comm_cart;
MPI_Init(NULL,NULL);
MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &world_myrank);
for (i=0; i<ndims; i++) { dims[i]=0; periods[i]=…; }
MPI_Dims_create(numprocs, ndims, dims); // computes factorization of numprocs
MPI_Cart_create(MPI_COMM_WORLD, ndims, dims, periods,1, &comm_cart);
MPI_Comm_rank(comm_cart, &cart_myrank);
MPI_Cart_coords(comm_cart, cart_myrank, ndims, my_coords, ierror)
```

From now, all communication should be based on
comm_cart & cart_myrank & my_coords

C

Fortran

Python

- C/C++: int MPI_Dims_create(int nnodes, int ndims, int *dims*)
- Fortran: MPI_DIMS_CREATE(nnodes, ndims, *dims, IERROR*)

mpi_f08:	INTEGER	:: nnodes, ndims, dims(*)
	INTEGER, OPTIONAL	:: ierror
mpi & mpif.h:	INTEGER nnodes, ndims, dims(*), ierror	

Array *dims* must be **initialized** with **zeros**
(other possibilities, see MPI standard)

- Python: *dims_out* = MPI.Compute_dims(nnodes, dims)

See mpi4py.MPI.Compute_dims — MPI for Python 3.1.1 documentation

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

Cartesian Mapping Functions

- Computing ranks of neighboring processes

- C/C++: int MPI_Cart_shift(MPI_Comm comm_cart, int direction, int disp, int **rank_source*, int **rank_dest*)

Fortran

- Fortran: MPI_CART_SHIFT(comm_cart, direction, disp, *rank_source*, *rank_dest*, *ierror*)

```
mpi_f08:       TYPE(MPI_Comm)         :: comm_cart
               INTEGER                :: direction, disp, rank_source, rank_dest
               INTEGER, OPTIONAL      :: ierror
mpi & mpif.h:  INTEGER  comm_cart, direction, disp, rank_source, rank_dest, ierror
```

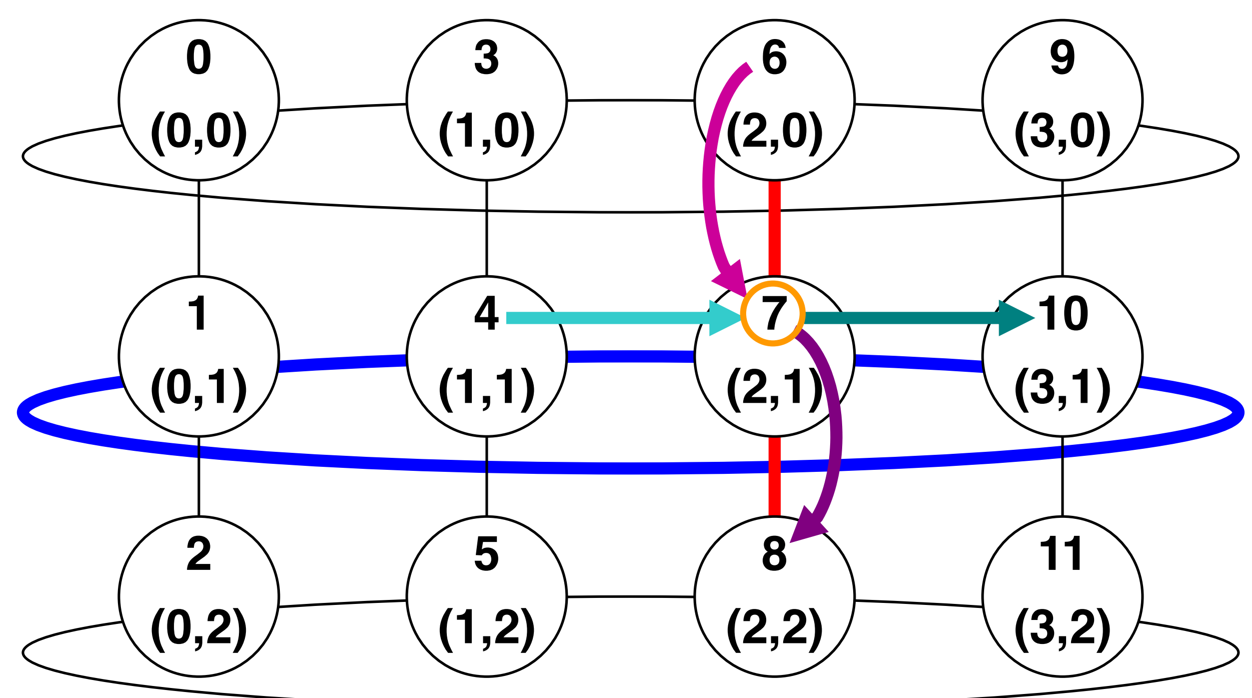
Python

- Python: (*rank_source*, *rank_dest*) = comm_cart.Shift(direction, disp)

- Returns MPI_PROC_NULL if there is no neighbor.
- MPI_PROC_NULL can be used as source or destination rank in each communication ➔ Then, this communication will be a no-operation!

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

MPI_Cart_shift – Example



(...) invisible input argument: **my_rank** in comm_cart

CALL MPI_Cart_shift (comm_cart, direction, disp, *rank_source*, *rank_dest*, *ierror*)

	direction	disp	*rank_source*	*rank_dest*
example on	0 or	**+1**	**4**	**10**
process rank=(7)	**1**	**+1**	**6**	**8**

Cartesian Partitioning

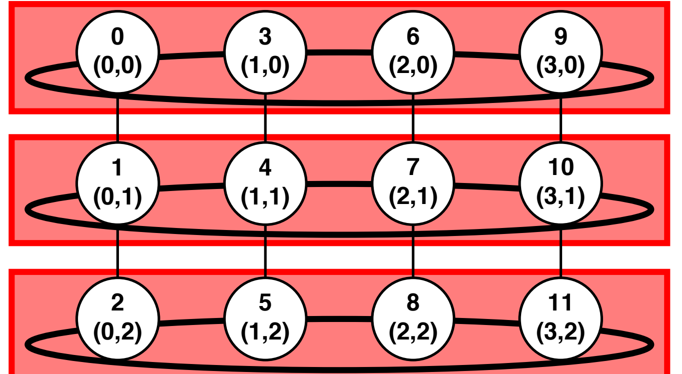
- Cut a virtual process grid up into *slices*.
- A new communicator is produced for each slice.
- Each slice can then perform its own collective communications.

C

- C/C++: int MPI_Cart_sub(MPI_Comm comm_cart, int *remain_dims, MPI_Comm **comm_slice*)

Fortran

- Fortran: MPI_CART_SUB(comm_cart, remain_dims, *comm_slice*, *ierror*)




```
mpi_f08:         TYPE(MPI_Comm)          :: comm_cart
                 LOGICAL                 :: remain_dims(*)
                 TYPE(MPI_Comm)          :: comm_slice
                 INTEGER, OPTIONAL       :: ierror
mpi & mpif.h:    INTEGER comm_cart, comm_slice, ierror
                 LOGICAL remain_dims(*)
```

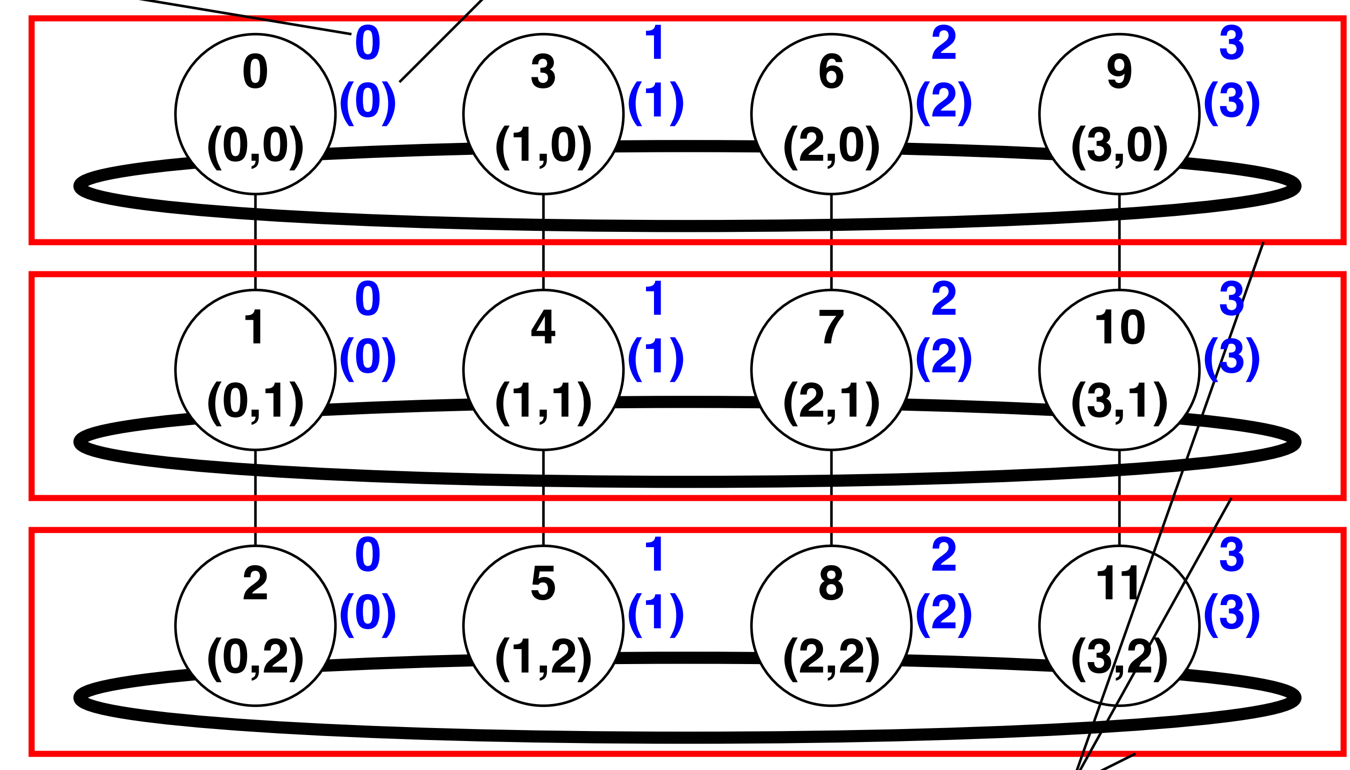
Python

- Python: *comm_slice* = comm_cart.Sub(remain_dims)

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

MPI_Cart_sub – Example

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



- CALL MPI_Cart_sub(comm_cart, remain_dims, ***comm_slice***, *ierror*)

(true, false)

Each process gets only its own sub-communicator

Four slides with **general remarks** before next exercise

Multidimensional domain decomposition

- Applications with 3 dimensions
  - each sub-domain (computed by one CPU) should
  - have the same size → optimal load balance
  - minimal surface → minimal communication
  - Usually optimum with **3-dim. domain decomposition** & **cubic** sub-domains
- Same rule for 2 dimensional application → 2-D domain decomposition & quadratic sub-domains

Exception: The total domain has extremely different dimensions, e.g., weather/climate: **40,000 km x 40,000 km x 15 km** (→ **only 2-dim** domain decomp.)

Splitting in

- one dimension:
  communication $= n^2 * 2 * w * 1 / p^0$
- two dimensions:
  communication $= n^2 * 2 * w * 2 / p^{1/2}$
- three dimensions:
  communication $= n^2 * 2 * w * 3 / p^{2/3}$

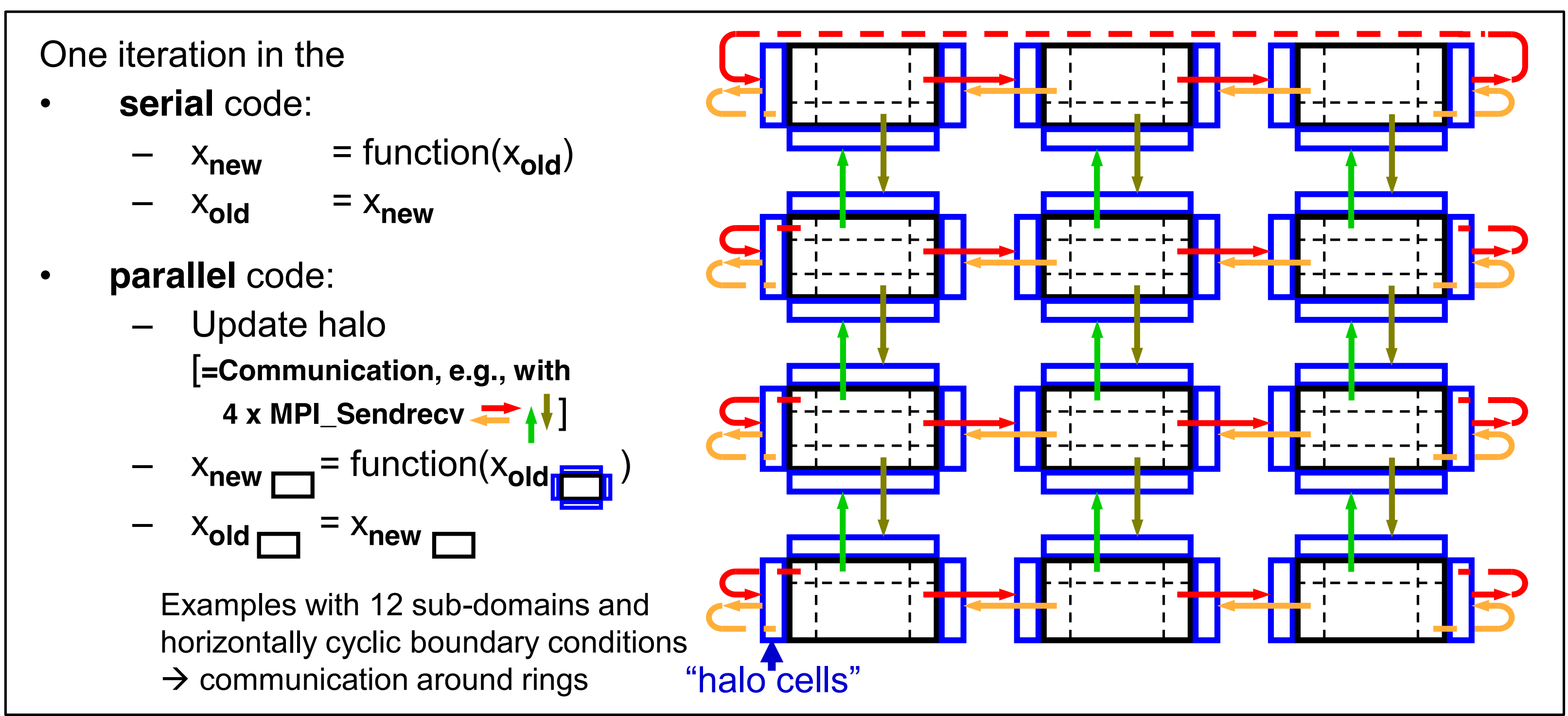
optimal for p≥12

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

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

General rule:
Symmetric vs. asymmetric manager/worker parallelization

- We know this example already from course chapter 1



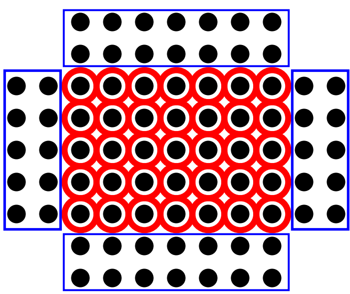
- General rule:
  - Always try to implement such a **symmetric parallelization design**
  - **Avoid** (asymmetric) **manager-worker**[1)]-paradigm
    → the manager always tend to **limit the scaling** to a larger number of processes

1) The outdated wording "*master/slave*" should be avoided

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

Halo

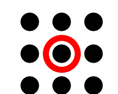
- Stencil:
  - To calculate a new data mesh point (○),
    old data from the stencil mesh points (•) are needed
    - **E.g., 9 point stencil**
- Halo
  - To calculate the new data mesh points of a sub-domain, additional mesh points from other sub-domains are needed.
  - They are stored in halos (ghost cells, shadows)
  - Halo depends on form of stencil

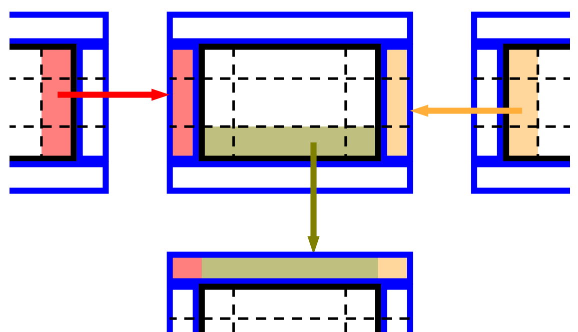


Back to Ch.1 on next slide

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

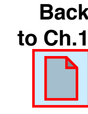
Diagonals Problem

- Stencil with diagonal point, e.g., 
  - i.e., halos include corners →→→ substitute small corner messages:
    - **one may use 2-phase-protocol:**
    - **normal horizontal halo communication**
    - **include corner into vertical exchange**



Chris Ding and Yun He: A ghost cell expansion method for reducing communications in solving PDE problems. Proc. SC2001. DOI:10.1145/582034.582084

Back to Ch.1



Course Chap. 9-(2):
Sparse Collective Operations on Process Topologies

New in MPI-3.0

- MPI process topologies (Cartesian and (distributed) graph) usable for communication
  - MPI_(I)NEIGHBOR_ALLGATHER(V)
  - MPI_(I)NEIGHBOR_ALLTOALL(V,W)
- If the topology is the full graph, then neighbor routine is identical to full collective communication routine
  - Exception: s/rdispls in MPI_NEIGHBOR_ALLTOALLW are MPI_Aint
- Allows for optimized communication scheduling and scalable resource binding
- Cartesian topology:
  - Sequence of buffer segments is communicated with:
    - **direction=0 source, direction=0 dest, direction=1 source, direction=1 dest, …**
  - Defined only for disp=1 (direction, source, dest and disp are defined as in MPI_CART_SHIFT)
  - If a source or dest rank is MPI_PROC_NULL then the buffer location is still there but the content is not touched.
  - See exercise 5 and advanced exercise 6

Exercise 1 — One-dimensional ring topology

Exercise 1

- Use a one-dimensional virtual Cartesian topology in the pass-around-the-ring program:
  Add a call to **MPI_Cart_create**, of course with reorder == non-zero or .TRUE. (e.g., 1)
- In MPI/tasks/… Use C C/Ch9/cart-create-skel.c or Fortran F_30/Ch9/cart-create-skel_30.f90
  or Python PY/Ch9/cart-create-skel.py
- **Caution**: Do only the prepared **one-dimensional virtual Cartesian topology**
- Hints:
  - After calling MPI_Cart_create,
    - there should be no further usage of MPI_COMM_WORLD, and
    - the my_rank must be recomputed on the base of comm_cart.
  - Only **one**-dimensional:
    - → coordinates are not necessary, because coord==rank

    In this exercise not relevant, because the skeleton already uses arrays:
    - → In C: `dims` and `period` as normal variables, i.e., no arrays, but call by reference with &dims, …
    - → In Fortran: `dims` and `period` must be arrays (i.e., with only 1 element, e.g., `(/.TRUE./)`)

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

Slide from Chap. 4 — Rotating information around a ring

Initialization: ①

Each iteration: ② ③ ④ ⑤

(1) Communication through a new reordered Cartesian communicator

(2) my_rank based on this new communicator

my_rank → ① → snd_buf → ② → rcv_buf (③) → ④ → snd_buf; rcv_buf → ⑤ → sum

Fortran:

```
dest   = mod(my_rank+1,size)
source = mod(my_rank–1+size,size)
```

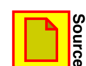
C/C++:

```
dest   = (my_rank+1) % size;
source = (my_rank–1+size) % size;
```

Single Program !!!
no IF-statements !!!

From
Chap.4 Nonblocking Communication

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online



Exercise 2 — One-dimensional ring topology

Exercises 2-4

- Use a one-dimensional in the pass-around-the-ring program:
  Add a call to **MPI_Cart_shift** to calculate left and right
- Use C C/Ch9/cart-shift-skel.c or Fortran F_30/Ch9/cart-shift-skel_30.f90
  or Python PY/Ch9/cart-shift-skel.py
- Goal:
  – the cryptic way to compute the neighbor ranks should be substituted by one call to MPI_Cart_shift, that should be before starting the loop.

© 2000-2022 HLRS, Rolf Rabenseifner

Slide from Chap. 4 — Rotating information around a ring

Initialization: (1)
Each iteration:
(2) (3) (4) (5)

Fortran:
dest = mod(my_rank+1,size)
source = mod(my_rank–1+size,size)
C/C++:
dest = (my_rank+1) % size;
source = (my_rank–1+size) % size;

Done in Exercise 1: (1) Communication through a new reordered Cartesian communicator

Done in Exercise 1: (2) my_rank based on this new communicator

(3) To be substituted by MPI_Cart_shift(… source, dest …), called only once, before starting the loop

my_rank
1
snd_buf
2
3
4
rcv_buf
5
sum

Single Program !!!
no IF-statements !!!

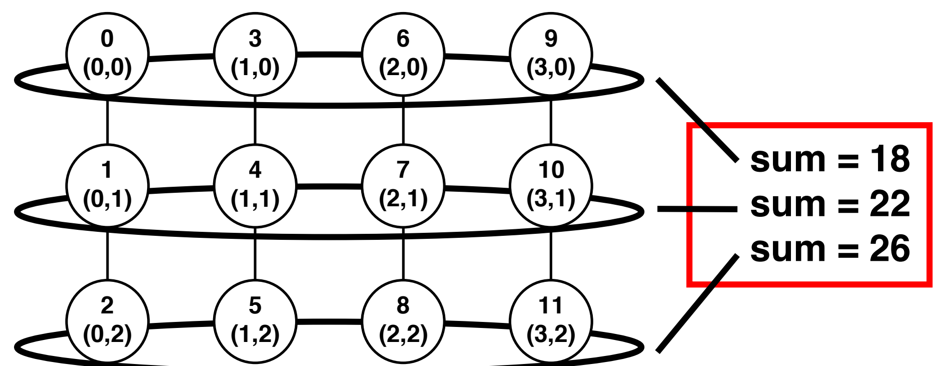
From
Chap.4 Nonblocking Communication

© 2000-2022 HLRS, Rolf Rabenseifner

Advanced Exercise 1b — Two-dimensional topology

Advanced Exercise 1b

- Task: Rewrite the exercise in two dimensions, as a cylinder.
  - Each row of the cylinder, i.e. each ring, should compute its own separate sum of the original ranks in the two dimensional comm_cart.
  - Compute the two dimensional factorization with MPI_Dims_create().
  - Array *dims* must be **initialized** with **(0,0) !**
  - Execute the ring algorithm in direction 0, i.e., communicating only to its left and right neighbors.
  - Calculate the neighbor ranks `left` and `right` using MPI_Cart_rank().
- Use **C** C/Ch9/cylinder-skel.c or **Fortran** F_30/Ch9/cylinder-skel_30.f90 or **Python** PY/Ch9/cylinder-skel.py
- Run with `mpirun -np 12 ./a.out | sed -e 's/PE//' | sort`

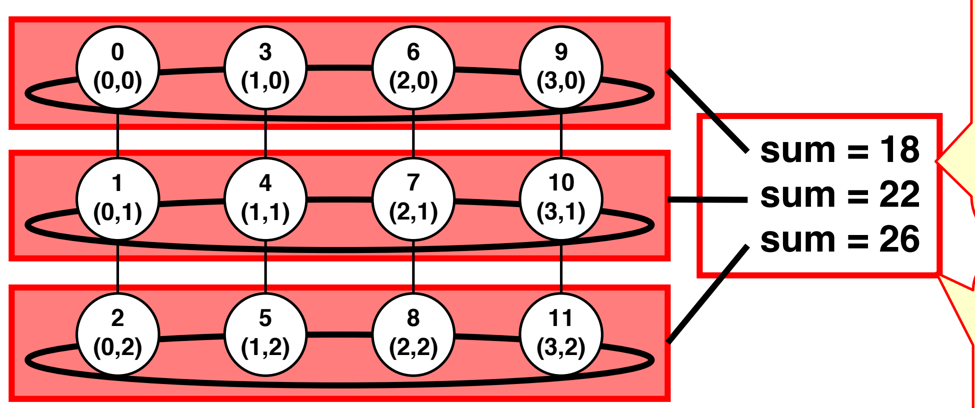


Exercise 3+4 (advanced) — Two-dimensional topology

C
Fortran
Python

- **Exercise 3**: Rewrite the exercise in two dimensions, as a cylinder.
  - Each row of the cylinder, i.e. each ring, should compute its own separate sum of the original ranks in the two dimensional comm_cart.
  - Task: substitute 2x MPI_Cart_rank by 1x MPI_Cart_shift
  - **Use** (your) solution of Ch.9-(1) Advanced exercise 1b:
    - Your modified **C, F_30, PY/Ch9/cylinder-skel.c, _30.f90, .py**
    - Or copy provided **C, F_30, PY/Ch9/solutions/cylinder.c, _30.f90, .py**

- **Exercise 4:** Use MPI_Cart_sub to create the one-dimensional slice communicators
  - Results are the same



Summing up the myrank of the 2-dimensional Cartesian topology:
Advanced Exercise 4a:
Ring-communication in the comm_slice, and using the ring with myrank, left, right and size of the comm_slice.

Additional Advanced Exercise 4b:
Using MPI_Allreduce within the comm_slice instead of the ring communication algorithm.
Solution, see 2nd Adv. Exe Chapter 6-(1)

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

Exercise 3+4 (advanced) — Two-dimensional ring topology

- In the solutions directories
  C C/Ch9/solutions/ & Fortran F_30/Ch9/solutions/ & Python PY/Ch9/solutions/
  - topology_advanced3_cylinder.c / _30.f90 / .py → 2-dim topology (Exa.3)
  - topology_advanced4_cart_sub.c / _30.f90 / .py → using MPI_Cart_sub (Exa. 4a)
  - And in directories

    C C/**Ch6**/solutions/ & Fortran F_30/**Ch6**/solutions/ & Python PY/**Ch6**/solutions/
    - cylinder_advanced2_subtopology.c / _30.f90 / .py
      → MPI_Cart_sub and MPI_allreduce (Exa. 4b)

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

Chapter 9 – Exercise 1: Ring with virtual Cartesian topology

C — MPI/tasks/C/Ch9/solutions/cart-create.c

```
MPI_Comm     comm_cart;
int          dims[1], periods[1], reorder;
-----------------------------------------------------------------
dims[0] = size;
periods[0] = 1;
reorder = 1;
MPI_Cart_create(MPI_COMM_WORLD, 1, dims, periods, reorder, &comm_cart);
MPI_Comm_rank(comm_cart, &my_rank);
right = (my_rank+1)        % size;
left  = (my_rank-1+size) % size;
-----------------------------------------------------------------
    MPI_Issend(&snd_buf, 1, MPI_INT, right, 17, comm_cart, &request);
    MPI_Recv ( &rcv_buf, 1, MPI_INT, left,  17, comm_cart, &status);
```

Fortran — MPI/tasks/F_30/Ch9/solutions/cart-create_30.f90

```
TYPE(MPI_Comm)  :: comm_cart
INTEGER  :: dims(1)
LOGICAL  :: periods(1), reorder
----------------------------------------
dims(1)     = size
periods(1)  = .TRUE.
reorder     = .TRUE.
CALL MPI_Cart_create(MPI_COMM_WORLD,&
& 1, dims, periods, reorder, comm_cart)
CALL MPI_Comm_rank(comm_cart, my_rank)
right = mod(my_rank+1,       size)
left  = mod(my_rank-1+size, size)
----------------------------------------
   CALL MPI_Issend(snd_buf,…,comm_cart,…)
   CALL MPI_Recv ( rcv_buf,…,comm_cart,…)
```

Python — MPI/tasks/PY/Ch9/solutions/cart-create.py

```
dims[0] = size
periods[0] = True
reorder = True

comm_cart =
 comm_world.Create_cart(dims=dims,
 periods=periods, reorder=reorder)
my_rank = comm_cart.Get_rank()
right = (my_rank+1)        % size
left  = (my_rank-1+size) % size
----------------------------------------
  request = comm_cart.Issend(
      (snd_buf,1,MPI.INT),right,17)
  comm_cart.Recv(
      (rcv_buf,1,MPI.INT),left, 17,
      status)
```

© 2000-2022 HLRS, Rolf Rabenseifner

Chapter 9 – Exercise 2: Ring with virtual Cartesian topology

C — MPI/tasks/C/Ch9/solutions/cart-shift.c

```c
MPI_Comm     comm_cart;
int          dims[1], periods[1], reorder;
-----------------------------------------------------------------------
dims[0] = size;  periods[0] = 1;  reorder = 1;
MPI_Cart_create(MPI_COMM_WORLD, 1, dims, periods, reorder, &comm_cart);
MPI_Comm_rank(comm_cart, &my_rank);
right = (my_rank+1)        % size;
left  = (my_rank-1+size) % size;
MPI_Cart_shift(comm_cart, 0, 1, &left, &right);
-----------------------------------------------------------------------
    MPI_Issend(&snd_buf, 1, MPI_INT, right, 17, comm_cart, &request);
    MPI_Recv ( &rcv_buf, 1, MPI_INT, left,  17, comm_cart, &status);
```

(The lines `right = ...` and `left = ...` are struck through.)

Fortran — MPI/tasks/F_30/Ch9/solutions/cart-shift_30.f90

```fortran
TYPE(MPI_Comm) :: comm_cart
INTEGER :: dims(1)
LOGICAL :: periods(1), reorder
-----------------------------------------------------------------------
dims(1) = size ;  periods(1) = .TRUE. ;  reorder = .TRUE.
CALL MPI_Cart_create(MPI_COMM_WORLD, 1, dims, periods, reorder, comm_cart)
CALL MPI_Comm_rank(comm_cart, my_rank)
right = mod(my_rank+1,        size)
left  = mod(my_rank-1+size, size)
CALL MPI_Cart_shift(comm_cart, 0, 1, left, right)
-----------------------------------------------------------------------
    CALL MPI_Issend(snd_buf,1,MPI_INTEGER, right, 17, comm_cart, request)
    CALL MPI_Recv ( rcv_buf,1,MPI_INTEGER, left,  17, comm_cart, status)
```

(The lines `right = mod(...)` and `left = mod(...)` are struck through.)

Python — MPI/tasks/PY/Ch9/solutions/cart-shift.py

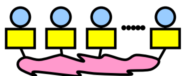
```python
right = (my_rank+1)        % size
left  = (my_rank-1+size) % size
(left,right) = comm_cart.Shift(0, 1)
```

(The lines `right = ...` and `left = ...` are struck through.)

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

Chap.12 Derived Datatypes

1. MPI Overview



2. Process model and language bindings

MPI_Init()
MPI_Comm_rank()

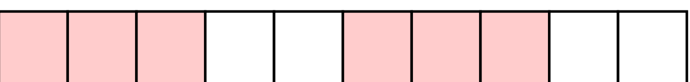
3. Messages and point-to-point communication
4. Nonblocking communication

5. The New Fortran Module mpi_f08
6. Collective communication
7. Error Handling
8. Groups & communicators, environment management

9. Virtual topologies
10. One-sided communication
11. Shared memory one-sided communication

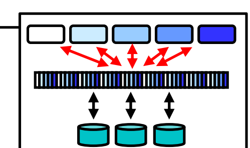
put
get

12. Derived datatypes



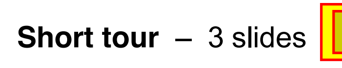
– **(1) transfer of any combination of typed data**
– **(2) advanced features, alignment, resizing**

13. Parallel file I/O



14. MPI and threads
15. Probe, Persistent Requests, Cancel
16. Process creation and management
17. Other MPI features
18. Best Practice

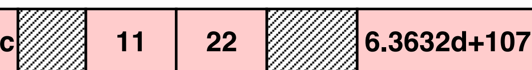
Short tour – 3 slides



© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

MPI Datatypes

- In the previous chapters:
  - A messages was a contiguous sequence of elements of basic types:
  - buf, count, datatype_handle
- New goals in this course chapter:
  - Transfer of any data in memory in one message

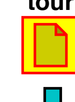


    - **Strided data (portions of data with holes between the portions)**
    - **Various basic datatypes within one message**
  - No multiple messages → **no multiple latencies**
  - No copying of data into contiguous scratch arrays → **no waste of memory bandwidth**
- Method: **Datatype handles**
  - Memory layout of send / receive buffer
  - Basic types / **derived types**:
    - **vectors**
    - **subarrays**
    - **structs**
    - **others**

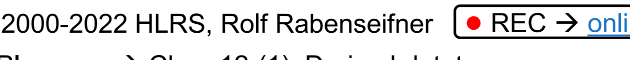
Message passing:
- **Goal and reality may differ !!!**

Parallel file I/O:
- Derived datatypes are **important** to express I/O patterns

tour



© 2000-2022 HLRS, Rolf Rabenseifner



Data Layout and the Describing Datatype Handle

```
struct buff_layout
  { int      i_val[3];
    double d_val[5];
  } buffer;
```

Compiler

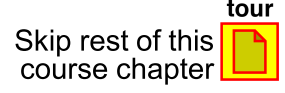
```
array_of_types[0]=MPI_INT;
array_of_blocklengths[0]=3;
array_of_displacements[0]=0;
array_of_types[1]=MPI_DOUBLE;
array_of_blocklengths[1]=5;
array_of_displacements[1]=…;
MPI_Type_create_struct(2, array_of_blocklengths,
    array_of_displacements, array_of_types,
                              &buff_datatype);
MPI_Type_commit(&buff_datatype);
```

```
MPI_Send(&buffer, 1, buff_datatype, …)
```

&buffer = the start address of the data

the datatype handle describes the data layout

int				double				



© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

Derived Datatypes — Type Maps

- A derived datatype is logically a pointer to a list of entries:
  - *basic datatype* at *displacement*

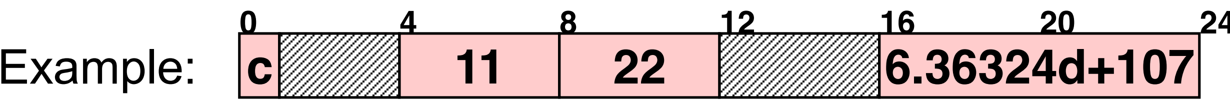
basic datatype 0	displacement of datatype 0
basic datatype 1	displacement of datatype 1
...	...
basic datatype n-1	displacement of datatype n-1

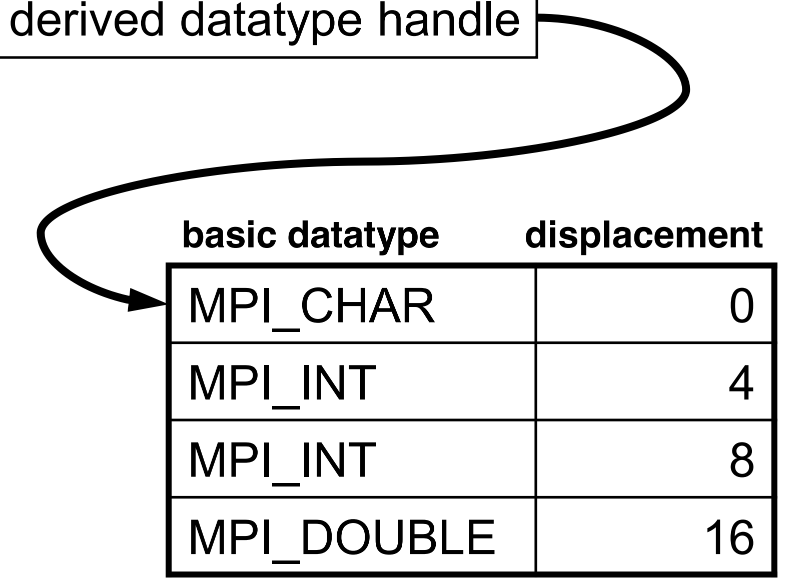
- Matching datatypes:
  - List of basic datatypes must be identical,
  - *(Displacements irrelevant)*

basic datatype 0	disp 0
basic datatype 1	disp 1
...	...
basic datatype n-1	disp n-1

Derived Datatypes — Type Maps

Example:





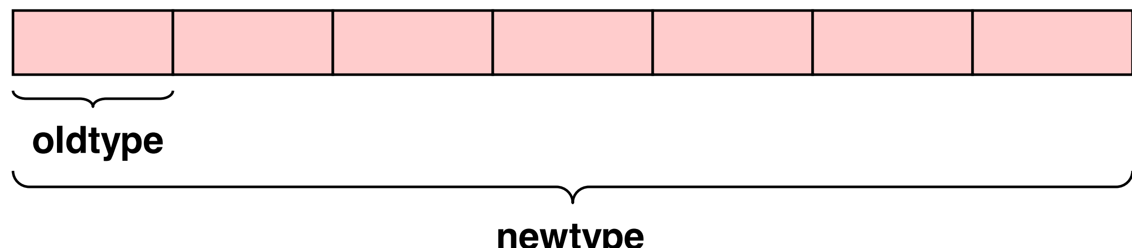
basic datatype	displacement
MPI_CHAR	0
MPI_INT	4
MPI_INT	8
MPI_DOUBLE	16

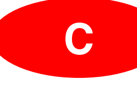
A derived datatype describes the memory layout of, e.g., structures, common blocks, subarrays, some variables in the memory

© 2000-2022 HLRS, Rolf Rabenseifner

Contiguous Data

- The simplest derived datatype
- Consists of a number of contiguous items of the same datatype





- C/C++: int MPI_Type_contiguous(int count, MPI_Datatype oldtype, MPI_Datatype *newtype)

- Fortran: MPI_TYPE_CONTIGUOUS(count, oldtype, *newtype, ierror*)

mpi_f08:

```
INTEGER                  :: count
TYPE(MPI_Datatype)       :: oldtype, newtype
INTEGER, OPTIONAL        :: ierror
```

mpi & mpif.h: INTEGER count, oldtype, newtype, ierror

- Python: *newtype* = oldtype.Create_contiguous(int count)

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

Committing and Freeing a Datatype

- Before a datatype handle is used in message passing communication, **it needs to be committed with MPI_TYPE_COMMIT**.

- This need be done only once (by each MPI process).
  (Using more than once ☹ corresponds to additional no-operations.)

- C/C++: int MPI_Type_commit(MPI_Datatype **datatype*);

- Fortran: MPI_TYPE_COMMIT(*datatype*, *IERROR*)
  mpi_f08: TYPE(MPI_Datatype) :: datatype
  INTEGER, OPTIONAL :: ierror
  mpi & mpif.h: INTEGER datatype, ierror

- Python: datatype.Commit()

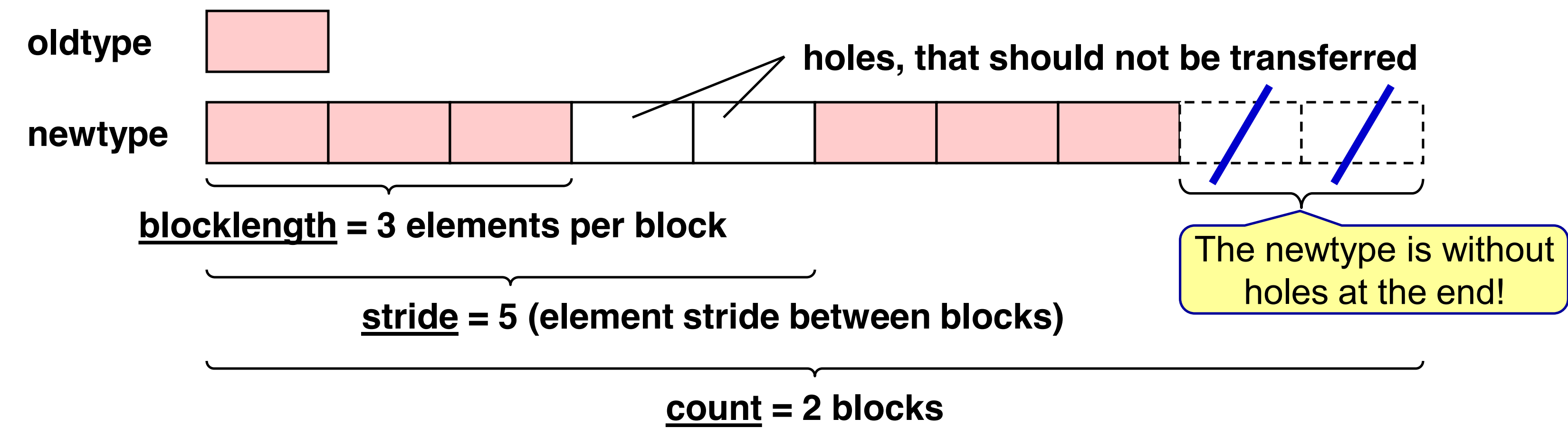
IN-OUT argument
(although handle
is not modified)

- If usage is over, one may call MPI_TYPE_FREE()
  to free a datatype and its internal resources.

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

Vector Datatype

MPI_Type_create_subarray is more flexible and usable for any dimensions, see course chapter 12-(2) and example in 13-(2)



C

Fortran

Python

- C/C++: int MPI_Type_vector(int count, int blocklength, int stride, MPI_Datatype oldtype, MPI_Datatype *newtype)
- Fortran: MPI_TYPE_VECTOR(count, blocklength, stride, oldtype, *newtype*, *ierror*)

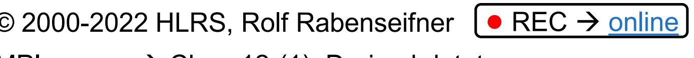
mpi_f08:

```
INTEGER                 :: count, blocklength, stride
TYPE(MPI_Datatype)      :: oldtype, newtype
INTEGER, OPTIONAL       :: ierror
```

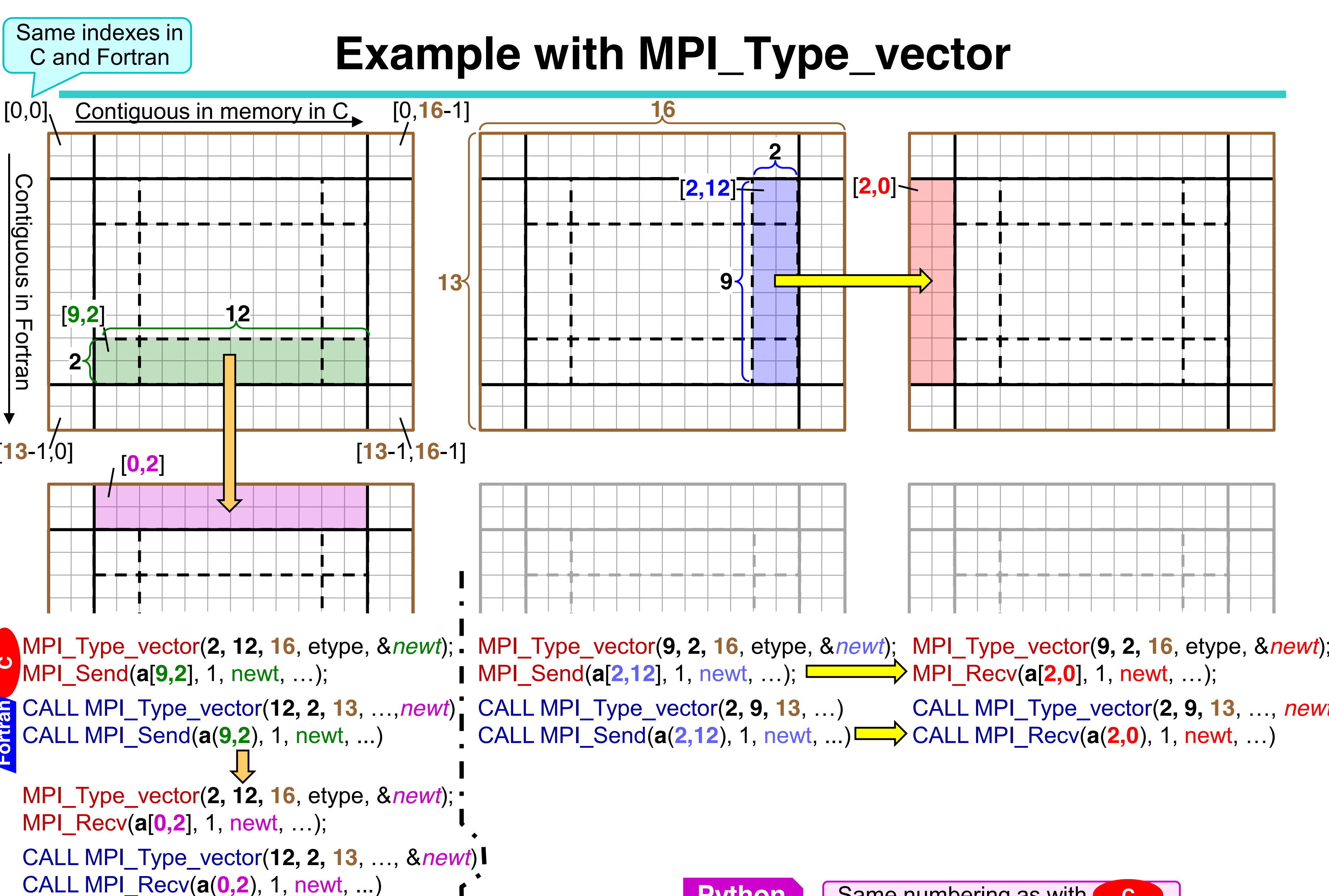
mpi & mpif.h: INTEGER count, blocklength, stride, oldtype, newtype, ierror

- Python: *newtype* = oldtype.Create_vector(int count, int blocklength, int stride)

© 2000-2022 HLRS, Rolf Rabenseifner

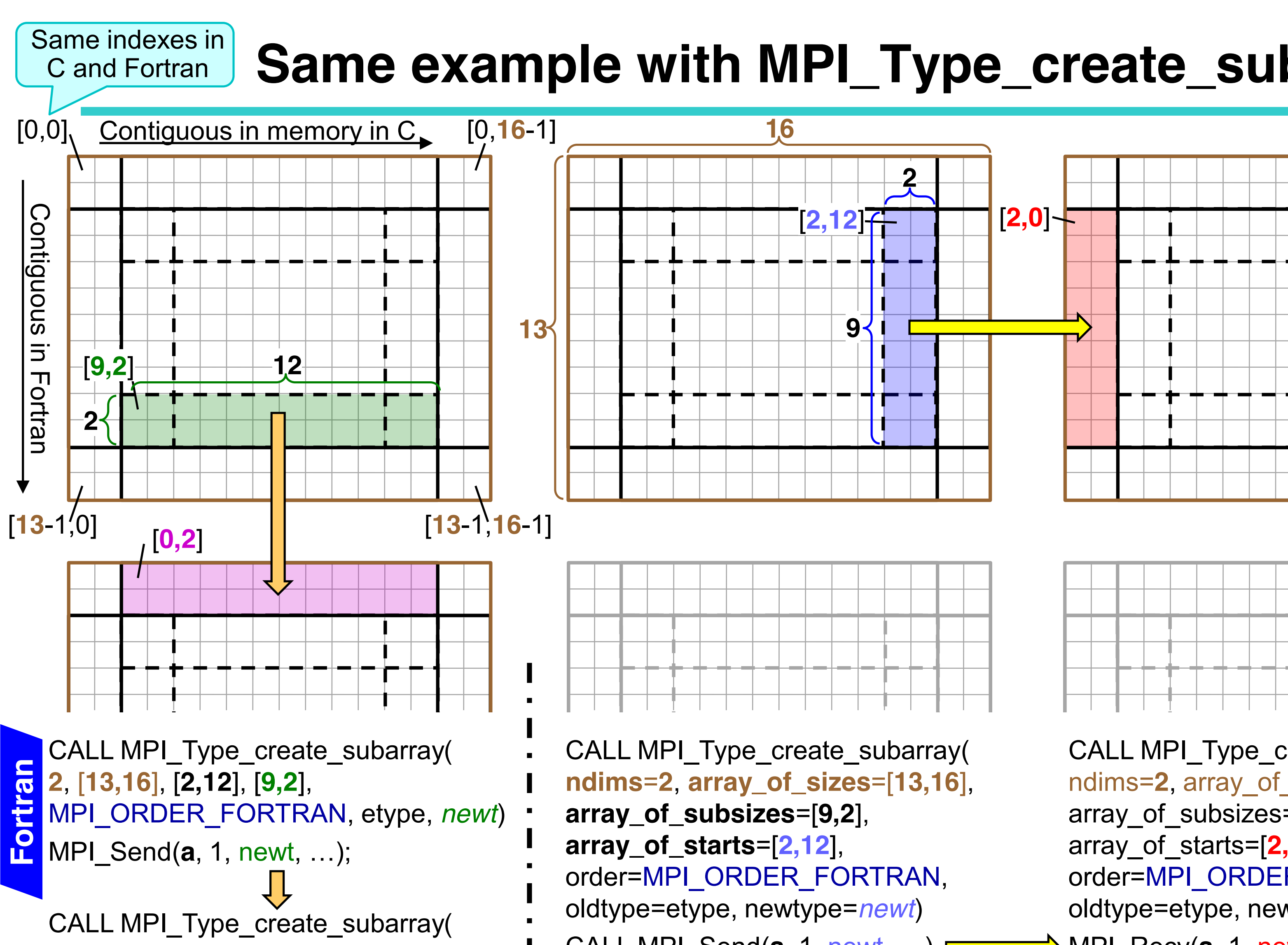


Example with MPI_Type_vector



© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

Same example with MPI_Type_create_subarray




```
CALL MPI_Type_create_subarray(
2, [13,16], [2,12], [9,2],
MPI_ORDER_FORTRAN, etype, newt)
MPI_Send(a, 1, newt, …);
```

```
CALL MPI_Type_create_subarray(
2, [13,16], [2,12], [0,2],
MPI_ORDER_FORTRAN, etype, newt)
MPI_Recv(a, 1, newt, …);
```

Fortran

```
CALL MPI_Type_create_subarray(
ndims=2, array_of_sizes=[13,16],
array_of_subsizes=[9,2],
array_of_starts=[2,12],
order=MPI_ORDER_FORTRAN,
oldtype=etype, newtype=newt)
CALL MPI_Send(a, 1, newt, …)
```

```
CALL MPI_Type_create_subarray(
ndims=2, array_of_sizes=[13,16],
array_of_subsizes=[9,2],
array_of_starts=[2,0],
order=MPI_ORDER_FORTRAN,
oldtype=etype, newtype=newt)
MPI_Recv(a, 1, newt, …)
```

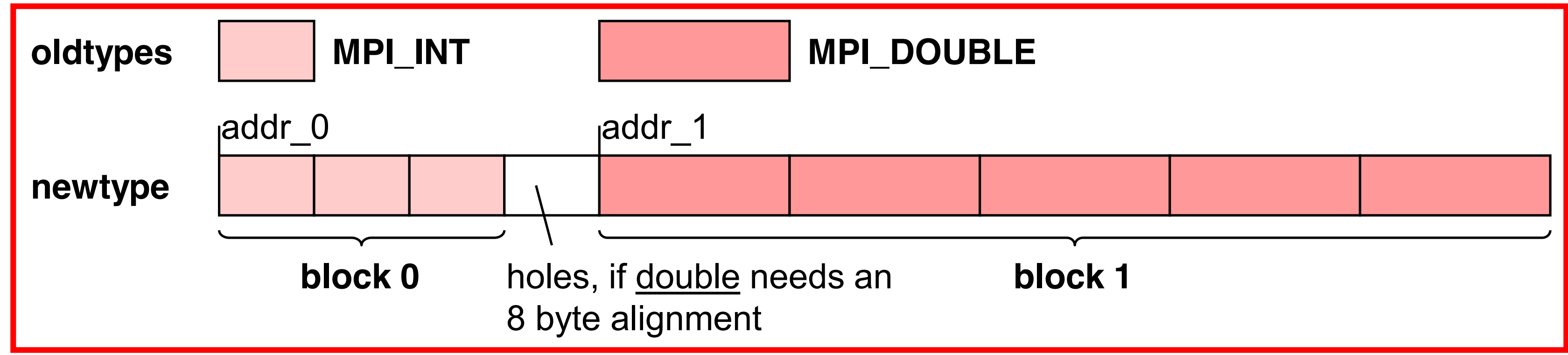
C
Python

Same numbers in C and Fortran, only the **order** is different: MPI_ORDER_C in C and Python

See also 13-(2) subarray

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

Struct Datatype



C

- C/C++: int MPI_Type_create_struct(int count, int *array_of_blocklengths,
  MPI_Aint *array_of_displacements, MPI_Datatype *array_of_types,
  MPI_Datatype *_newtype_)

Fortran

- Fortran: MPI_TYPE_CREATE_STRUCT(count, array_of_blocklengths,
  array_of_displacements[1], array_of_types, _newtype_, _ierror_)

Python

- Python: _newtype_ = MPI.Datatype.Create_struct(array_of_blocklengths,
  array_of_displacements, array_of_types)

count	**= 2**		
array_of_blocklengths	**= (3,**	**5**	**)**
array_of_displacements	**= (0,**	**addr_1 – addr_0**	**)** [2]
array_of_types	**= (MPI_INT,**	**MPI_DOUBLE**	**)**

[1] INTEGER**(KIND=MPI_ADDRESS_KIND)** array_of_displacements
[2] Via MPI_Get_address and MPI_Aint_diff, see following slides

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

Memory Layout of Struct Datatypes

buf_datatype	int				double				

Fixed memory layout:

- C

```
struct buff
    {  int        i_val[3];
       double  d_val[5];  }
```

- Fortran, derived types

```
TYPE buff_type
  SEQUENCE
  INTEGER, DIMENSION(3):: i_val
  DOUBLE PRECISION, &
                DIMENSION(5):: d_val
END TYPE buff_type
TYPE (buff_type) :: buff_variable
```

!!! (SEQUENCE)

Alternative, in MPI-3.0:

```
TYPE, BIND(C) :: buff_type
```

- Fortran, common block

```
integer i_val(3)
double precision d_val(5)
common /bcomm/ i_val, d_val
```

- Python – mpi4py with numpy:

```
buff_type = np.dtype([('i', np.intc, 3), ('d', np.double, 5)], align=True)
buff = np.empty((),dtype=buff_type)
```

True: with hole, as in C.
Default = False: no holes,
problematic, see course Chapter 12-(2).

Alternatively, arbitrary memory layout:

- Each array is allocated independently.
- Each buffer is a pair of a 3-int-array and a 5-double-array.
- The length of the hole may be any arbitrary positive or negative value!
- For each buffer, one needs a specific datatype handle
- **CAUTION – Fortran register optimization:** MPI_Send & _Recv of **…d_val** is invi-sible for the compiler → add MPI_Address

in_buf_datatype

in_i_val				in_d_val				

out_buf_datatype

out_i_val				out_d_val				

Not portable, because address differences are allowed only inside of structures or arrays

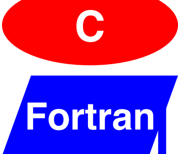
→ MPI-3.1/-4.0, Sect. 4/5.1.12 "Correct Use of Addresses"

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

How to compute the displacement (1)

- array_of_displacements[i] := address(block_i) – address(block_0)

Retrieve an absolute address:



– C/C++: int MPI_**Get**_address(void* location, MPI_Aint **address*)

– Fortran: MPI_**GET**_ADDRESS(location, *address*, *ierror*)

mpi_f08:
```
TYPE(*), DIMENSION(..), ASYNCHRONOUS :: location
INTEGER(KIND=MPI_ADDRESS_KIND) :: address
INTEGER, OPTIONAL :: ierror
```

mpi & mpif.h:
```
<type>    location(*)
INTEGER(KIND=MPI_ADDRESS_KIND) address
INTEGER ierror
```

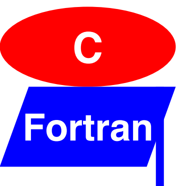
Python

– Python: *address* = MPI.**Get**_address(location)

How to compute the displacement (2)

New in MPI-3.1

Relative displacement:= absolute address 1 – absolute address 2

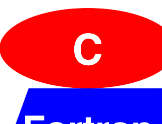


- C/C++: *MPI_Aint* MPI_Aint_diff(MPI_Aint addr1, MPI_Aint *addr2*)
- Fortran: MPI_AINT_DIFF(addr1, addr2)
  mpi_f08: INTEGER(KIND=MPI_ADDRESS_KIND) :: addr1, addr2
  mpi & mpif.h: INTEGER(KIND=MPI_ADDRESS_KIND) addr1, addr2
- Python: *int* MPI.Aint_diff(addr1, *addr2*)

Python's int allows 64 bit

New in MPI-3.1

New absolute address := existing absolute address + relative displacement:



- C/C++: *MPI_Aint* MPI_Aint_add(MPI_Aint base, MPI_Aint *disp*)
- Fortran: MPI_AINT_ADD(base, disp)
  mpi_f08: INTEGER(KIND=MPI_ADDRESS_KIND) :: base, disp
  mpi & mpif.h: INTEGER(KIND=MPI_ADDRESS_KIND) base, disp
- Python: *int* MPI.Aint_add(base, *disp*)

Python

© 2000-2022 HLRS, Rolf Rabenseifner

Example for array_of_displacements[i] := address(block_i) – address(block_0)

See also MPI-3.1/MPI-4.0, Example 4.8/5.8, page 102/142
and Example 4.17/5.17, pp 125-127/168-171

C

New in MPI-3.1

```
struct buff
    {       int       i[3];
            double d[5];
    } snd_buf;
MPI_Aint iaddr0, iaddr1, disp;
MPI_Get_address( &snd_buf.i[0], &iaddr0);  // the address value &snd_buf.i[0] is stored into variable iaddr0
MPI_Get_address(&snd_buf.d[0], &iaddr1);
disp = MPI_Aint_diff(iaddr1, iaddr0);         // MPI-3.0 & former: disp = iaddr1–iaddr0
```

Fortran

New in MPI-3.1

```
TYPE buff_type
  SEQUENCE
  INTEGER,                    DIMENSION(3) :: i
  DOUBLE PRECISION,           DIMENSION(5) :: d
END TYPE buff_type
TYPE (buff_type) :: snd_buf
INTEGER(KIND=MPI_ADDRESS_KIND) iaddr0, iaddr1, disp;  INTEGER ierror
CALL MPI_GET_ADDRESS( snd_buf%i(1), iaddr0, ierror)  ! The address of snd_buf%i(1) is stored in iaddr0
CALL MPI_GET_ADDRESS(snd_buf%d(1), iaddr1, ierror)
disp = MPI_AINT_DIFF(iaddr1, iaddr0)                   ! MPI-3.0 & former: disp = iaddr2–iaddr1
```

Python

```
np_dtype = np.dtype([('i', np.intc, 3), ('d', np.double, 4)])
snd_buf = np.empty((),dtype=np_dtype)
addr0 = MPI.Get_address(snd_buf['i'])
addr1 = MPI.Get_address(snd_buf['d'])
disp = MPI.Aint_diff(addr1, addr0)
```

Scope & Performance options

Scope of MPI derived datatypes:

- Fixed memory layout
- but not a linked list/tree,
  i.e., if the location of data portions depend on data (pointers/indexes) in this list

→ C++ data structures often require external libraries for flattening such data

→ E.g., Boost serialization methods

Which is the fastest neighbor communication with strided data?

- **Copying** the strided data in a contiguous scratch send-buffer, communicating this send-buffer into a contiguous recv-buffer, and copying the recv-buffer back into the strided application array
- Using derived datatype handles
- And which of the communication routines should be used?

Especially with **hybrid MPI+OpenMP**, multiple threads may be used for such **copying**, whereas an MPI call may internally process **derived types** only with one thread.

No answer by the MPI standard, because:

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

© 2000-2022 HLRS, Rolf Rabenseifner REC → online

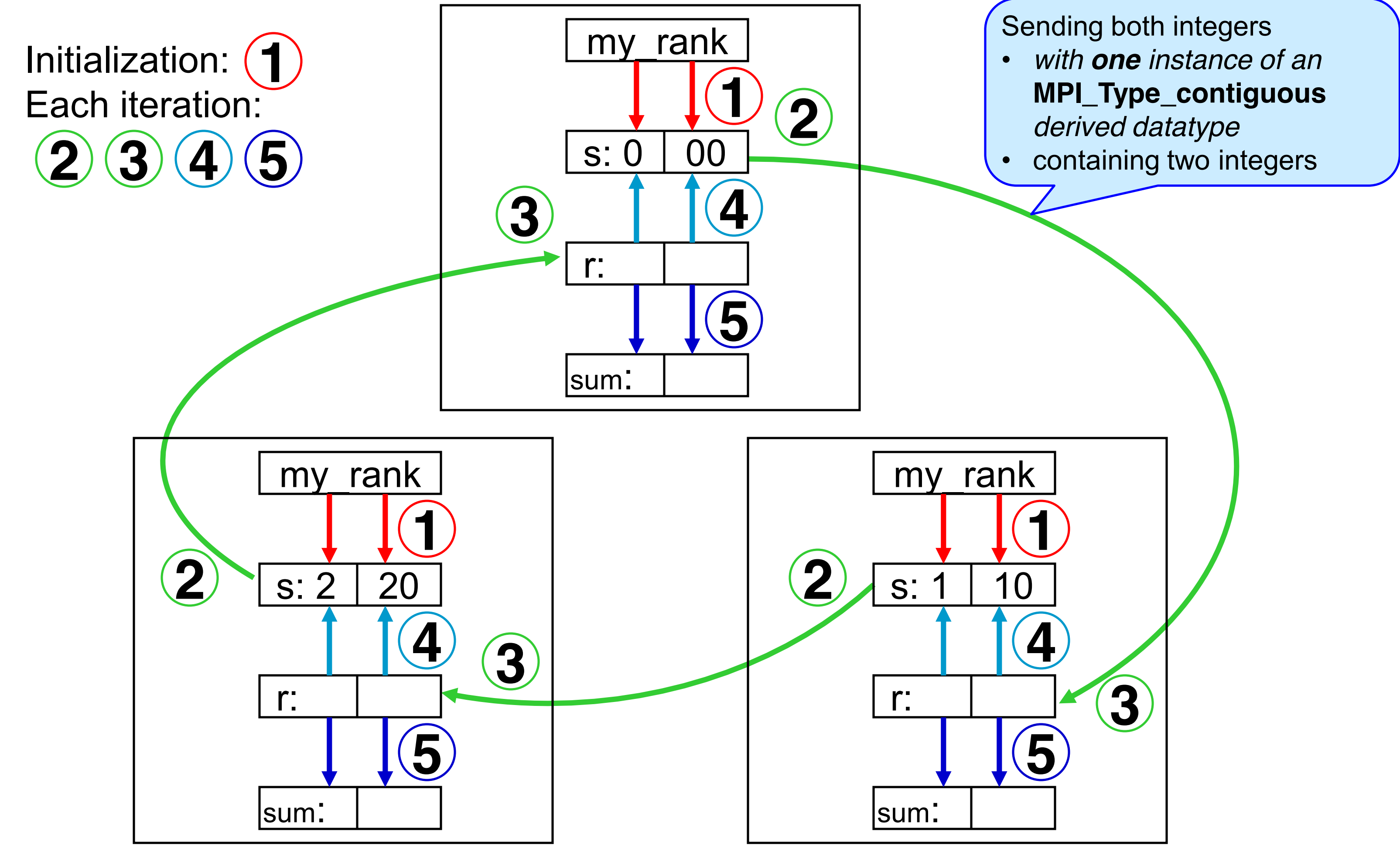
Exercise 1 — Derived Datatypes

In MPI/tasks/…

- Use C C/Ch12/derived-contiguous-skel.c
  or Fortran F_30/Ch12/derived-contiguous-skel_30.f90
  or Python PY/Ch12/derived-contiguous-skel.py
- We us a modified pass-around-the-ring exercise:
  It sends a struct with two integers
- They are initialized with **my_rank** and **10*my_rank**
- Therefore we calulate two separate sums.
- Currently, the data is send with the description
  - "snd_buf, 2, MPI_INTEGER"
- Please substitute this by using a
  - derived datatype
  - with a type map of "two integers"
  - Of course produced with the two routines on the previous slides

© 2000-2022 HLRS, Rolf Rabenseifner

Exercise 1 — Derived Datatypes



© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

Exercise 2 — Derived Datatypes

Exercise 2-4

- Modify the pass-around-the-ring exercise.
- Use the following skeletons to reduce software-coding time:

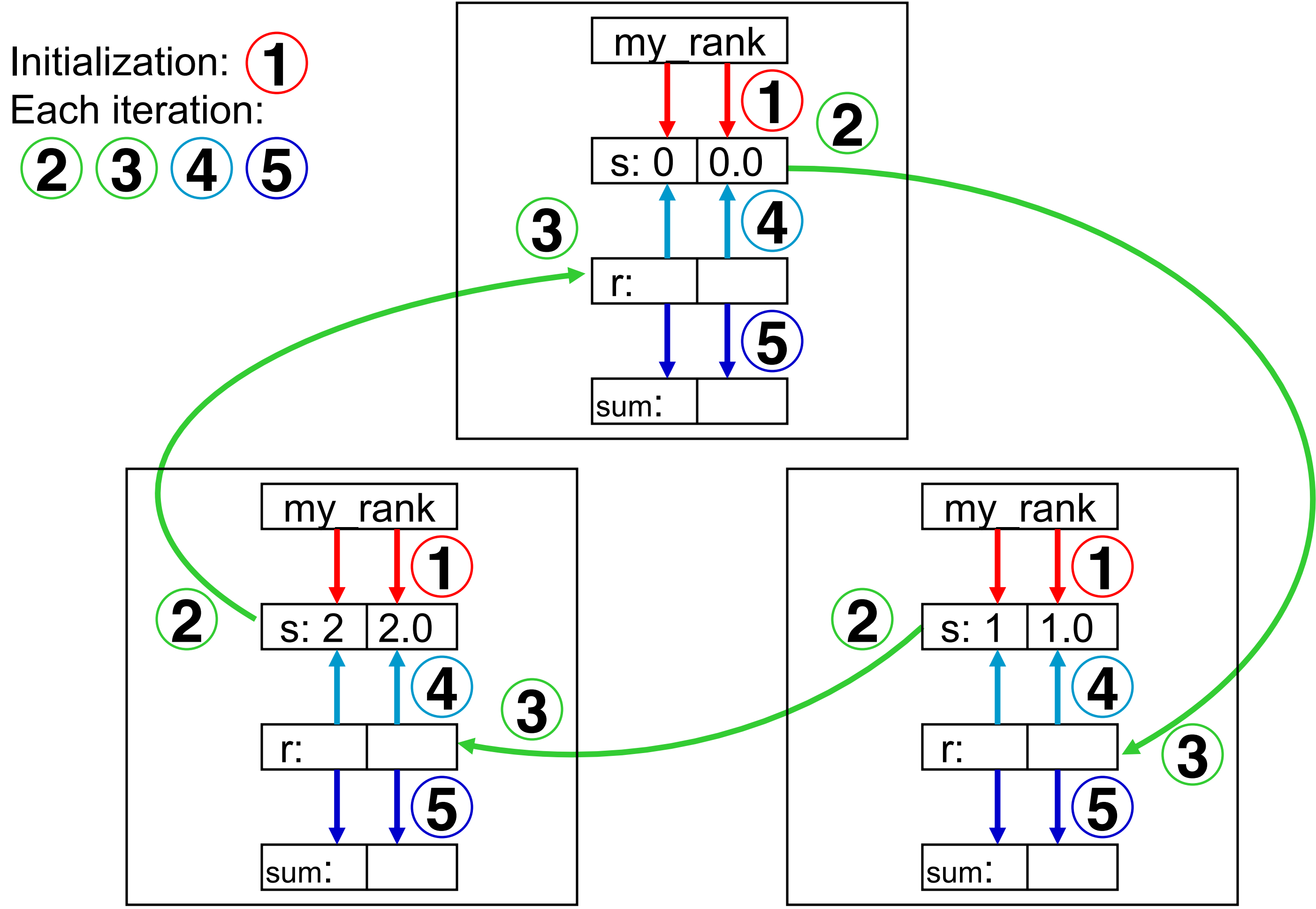
C	cd ~/MPI/tasks/**C**/Ch12/ ;	cp -p	derived-struct-skel.c	derived-struct.c
Fortran	cd ~/MPI/tasks/**F_30**/Ch12/ ;	cp -p	derived-struct-skel_30.f90	derived-struct_30.f90
Python	cd ~/MPI/tasks/**PY**/Ch12/ ;	cp -p	derived-struct-skel.py	derived-struct.py

- Calculate two separate sums:
  - rank integer sum (as before)
  - rank floating point sum
- Use a *struct* datatype for this
- with same fixed memory layout for send and receive buffer.
- Substitute all ___ within the skeleton
  and modify the second part, i.e., steps 1-5 of the ring example

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

Exercise 2 — Derived Datatypes

Initialization: 1
Each iteration:
2 3 4 5



© 2000-2022 HLRS, Rolf Rabenseifner

Exercises 1b (advanced) — MPI_Sendrecv

3. Substitute your Issend–Recv–Wait method by **MPI_Sendrecv** in your ring-with-datatype program:
   - MPI_Sendrecv is a *deadlock-free* combination of MPI_Send and MPI_Recv: ② ③
   - MPI_Sendrecv is described in the MPI standard.
     (You can find MPI_Sendrecv by looking at the function index on the last pages of the standard document.)
   - Start from your solution of Exercise 1
   - Solution: MPI/tasks/C/Ch12/solutions/derived-contiguous-advanced-sendrecv.c
     and MPI/tasks/F_30/Ch12/solutions/derived-contiguous-advanced-sendrecv_30.f90
     and MPI/tasks/PY/Ch12/solutions/derived-contiguous-advanced-sendrecv.py

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

Exercises 3+4 (advanced) — Sendrecv & Sendrecv_replace

3. Substitute your Issend–Recv–Wait method by **MPI_Sendrecv** in your ring-with-datatype program:
   - MPI_Sendrecv is a *deadlock-free* combination of MPI_Send and MPI_Recv: (2) (3)
   - MPI_Sendrecv is described in the MPI standard.
     - **You can find MPI_Sendrecv by looking at the function index on the last pages of the standard document.**
   - Solution: MPI/tasks/C/Ch12/solutions/derived-struct-advanced-sendrecv.c
     and MPI/tasks/F_30/Ch12/solutions/derived-struct-advanced-sendrecv_30.f90
     and MPI/tasks/PY/Ch12/solutions/derived-struct-advanced-sendrecv.py

Same Exercise as Advanced Exercise 1b

If you solved already Advanced Exercise 1b then move to Exercise 4

4. Substitute MPI_Sendrecv by **MPI_Sendrecv_replace**:
   - Three steps are now combined: (2) (3) (4)
   - The receive buffer (rcv_buf) must be removed.
   - The iteration is now reduced to three statements:
     - **MPI_Sendrecv_replace to pass the ranks around the ring,**
     - **computing the integer sum,**
     - **computing the floating point sum.**
   - Solution: MPI/tasks/C/Ch12/solutions/derived-struct-advanced-sendrecv-replace.c
     and MPI/tasks/F_30/Ch12/solutions/derived-struct-advanced-sendrecv-replace_30.f90
     and MPI/tasks/PY/Ch12/solutions/derived-struct-advanced-sendrecv-replace.py

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

Chapter 12-(1), Exercise 1: MPI_TYPE_CONTIGUOUS

MPI/tasks/C/Ch12/solutions/derived-contiguous.c

C

```
struct buff{
      int   i;
      int   j;
} snd_buf, rcv_buf, sum;
```

Provided in the skeleton

```
MPI_Datatype send_recv_type;
```

```
MPI_Type_contiguous(2, MPI_INT, &send_recv_type);
MPI_Type_commit(&send_recv_type);
```

```
sum.i = 0;                  sum.f = 0;
snd_buf.i = my_rank;   snd_buf.j = 10*my_rank;

for( i = 0; i < size; i++)
{ MPI_Issend(&snd_buf,1,send_recv_type,right,17,MPI_COMM_WORLD, &request);
  MPI_Recv ( &rcv_buf,1,send_recv_type,left, 17,MPI_COMM_WORLD, &status);
  MPI_Wait(&request, &status);
  snd_buf = rcv_buf;
  sum.i += rcv_buf.i;  sum.j += rcv_buf.j;
}

printf ("PE %i: Sum = %i and %i \n", my_rank, sum.i, sum.j);
```

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

Chapter 12-(1), Exercise 1: MPI_TYPE_CONTIGUOUS

Fortran

MPI/tasks/F_30/Ch12/solutions/derived_contiguous_30.f90

```
TYPE t
   SEQUENCE
   INTEGER :: i
   INTEGER :: j
END TYPE t
```

Provided in the skeleton

```
TYPE(MPI_Datatype) :: send_recv_type
CALL MPI_Type_contiguous(2, MPI_INT, send_recv_type)
CALL MPI_Type_commit(send_recv_type)
sum%i = 0            ; sum%r = 0 ;
snd_buf%i = my_rank ; snd_buf%j = my_rank
DO i = 1, size
  CALL MPI_Issend(snd_buf,1,send_recv_type,right,17,MPI_COMM_WORLD,request)
  CALL MPI_Recv ( rcv_buf,1,send_recv_type,left, 17,MPI_COMM_WORLD,status)
  CALL MPI_Wait(request, status)
  IF (.NOT.MPI_ASYNC_PROTECTS_NONBLOCKING) CALL MPI_F_sync_reg(snd_buf)
  snd_buf = rcv_buf
  sum%i = sum%i + rcv_buf%i ;  sum%j = sum%j + rcv_buf%j
END DO
WRITE(*,*) 'PE', my_rank, ': Sum%i =', sum%i, ' Sum%j =', sum%j
```

Python

MPI/tasks/PY/Ch12/solutions/derived_contiguous.py

```
np_dtype = np.dtype([('i', np.intc), ('j', np.intc)])
snd_buf = np.empty((),dtype=np_dtype)
rcv_buf = np.empty_like(snd_buf)
```

Provided in the skeleton

```
send_recv_type = MPI.INT.Create_contiguous(2)
send_recv_type.Commit()
   request = comm_world.Issend((snd_buf, 1, send_recv_type), right, 17)
   comm_world.Recv((rcv_buf, 1, send_recv_type), left, 17, status)
```

© 2000-2022 HLRS, Rolf Rabenseifner

Chapter 12-(1), Exercise 2: Halo-copy with derived types

C

MPI/tasks/C/Ch12/solutions/derived-struct.c

Provided in the skeleton

```
struct buff{
      int   i;
      float f;
} snd_buf, rcv_buf, sum;
int       array_of_blocklengths[2];
MPI_Aint array_of_displacements[2], first_var_address, second_var_address;
MPI_Datatype array_of_types[2], send_recv_type;
------------------------------------------------------------------------
array_of_types[0] = MPI_INT;  array_of_types[1] = MPI_FLOAT;
array_of_blocklengths[0] = 1;  array_of_blocklengths[1] = 1;
MPI_Get_address(&snd_buf.i, &first_var_address);
MPI_Get_address(&snd_buf.f, &second_var_address);
array_of_displacements[0] = (MPI_Aint) 0;
array_of_displacements[1]=MPI_Aint_diff(second_var_address,first_var_address);
MPI_Type_create_struct(2, array_of_blocklengths,  array_of_displacements,
                          array_of_types, &send_recv_type);
MPI_Type_commit(&send_recv_type);
------------------------------------------------------------------------
sum.i = 0;              sum.f = 0;
snd_buf.i = my_rank;    snd_buf.f = 10*my_rank;
for( i = 0; i < size; i++)
{ MPI_Issend(&snd_buf,1,send_recv_type,right,17,MPI_COMM_WORLD, &request);
  MPI_Recv ( &rcv_buf,1,send_recv_type,left, 17,MPI_COMM_WORLD, &status);
  MPI_Wait(&request, &status);
  snd_buf = rcv_buf;
  sum.i += rcv_buf.i;  sum.f += rcv_buf.f;
}
printf ("PE %i: Sum = %i and %f \n", my_rank, sum.i, sum.f);
```

© 2000-2022 HLRS, Rolf Rabenseifner ● REC → online

Chapter 12-(1), Exercise 2: Halo-copy with derived types

Fortran

MPI/tasks/F_30/Ch12/solutions/derived_struct_30.f90

Provided in the skeleton

```
TYPE t
   SEQUENCE
   INTEGER :: i
   REAL    :: r
END TYPE t
TYPE(t), ASYNCHRONOUS :: snd_buf
TYPE(t) :: rcv_buf, sum
TYPE(MPI_Datatype) :: send_recv_type
INTEGER(KIND=MPI_ADDRESS_KIND) :: array_of_displacements(2)
INTEGER(KIND=MPI_ADDRESS_KIND) :: first_var_address, second_var_address
-------------------------------------------------------------------------
CALL MPI_Get_address(snd_buf%i, first_var_address)
CALL MPI_Get_address(snd_buf%r, second_var_address)
array_of_displacements(1) = 0
array_of_displacements(2)=MPI_Aint_diff(second_var_address,first_var_address)
CALL MPI_Type_create_struct(2, (/1,1/), &
 &      array_of_displacements, (/MPI_INTEGER,MPI_REAL/), send_recv_type)
CALL MPI_Type_commit(send_recv_type)
-------------------------------------------------------------------------
sum%i = 0             ; sum%r = 0 ;
snd_buf%i = my_rank ; snd_buf%r = REAL(10*my_rank)
DO i = 1, size
  CALL MPI_Issend(snd_buf,1,send_recv_type,right,17,MPI_COMM_WORLD,request)
  CALL MPI_Recv ( rcv_buf,1,send_recv_type,left, 17,MPI_COMM_WORLD,status)
  CALL MPI_Wait(request, status)
  IF (.NOT.MPI_ASYNC_PROTECTS_NONBLOCKING) CALL MPI_F_sync_reg(snd_buf)
  snd_buf = rcv_buf
  sum%i = sum%i + rcv_buf%i ;  sum%r = sum%r + rcv_buf%r
END DO
WRITE(*,*) 'PE', my_rank, ': Sum%i =', sum%i, ' Sum%r =', sum%r
```

© 2000-2022 HLRS, Rolf Rabenseifner

● REC → online

Chapter 12-(1), Exercise 2: Halo-copy with derived types

Python

MPI/tasks/PY/Ch12/solutions/derived_struct.py

Provided in the skeleton

```
np_dtype = np.dtype([('i', np.intc), ('f', np.single)])
snd_buf = np.empty((),dtype=np_dtype)
rcv_buf = np.empty_like(snd_buf); sum = np.empty_like(snd_buf)
array_of_blocklengths = [None]*2
array_of_displacements = [None]*2
array_of_types = [None]*2
-------------------------------------------------------------------------
array_of_blocklengths[0] = 1;     array_of_types[0] = MPI.INT
array_of_blocklengths[1] = 1;     array_of_types[1] = MPI.FLOAT
first_var_address = MPI.Get_address(snd_buf['i'])
second_var_address = MPI.Get_address(snd_buf['f'])
array_of_displacements[0]= 0
array_of_displacements[1]=MPI.Aint_diff(second_var_address,first_var_address)
send_recv_type = MPI.Datatype.Create_struct(array_of_blocklengths,
                                           array_of_displacements, array_of_types)
send_recv_type.Commit()
-------------------------------------------------------------------------
sum['i'] = 0;              sum['f'] = 0
snd_buf['i'] = my_rank;  snd_buf['f'] = 10*my_rank  # Step 1 = init
for i in range(size):
   request = comm_world.Issend((snd_buf, 1, send_recv_type), right, 17)#St.2a
   comm_world.Recv((rcv_buf, 1, send_recv_type), left, 17, status)  # Step 3
   request.Wait(status)                                              # Step 2b
   np.copyto(snd_buf,rcv_buf)                                        # Step 4
   sum['i'] += rcv_buf['i'];  sum['f'] += rcv_buf['f']               # Step 5
print(f"PE{my_rank}:\tSum = {sum['i']}\t{sum['f']}")
```

© 2000-2022 HLRS, Rolf Rabenseifner