Scientific Computing on Parallel Machines

Cálculo científico en ordenadores paralelos

Markus Uhlmann

CIEMAT

www.ciemat.es/sweb/comfos/personal/uhlmann

UCM – Abril 2008

Schedule

Part I	Introduction to parallel programming	lecture 1
Part II	Introduction to MPI	
	General introduction & MPI program structure	lecture 2
	Point-to-point communication	lecture 3,4
	Collective communication	lecture 5
	2D Poisson example	lecture 6,7
	Non-contiguous data & Mixed datatypes	lecture 7
	Virtual topologies & Communication subsets	lecture 8
	Use of linear algebra libraries	lecture 9
	Extensions	lecture 10

Part II Introduction to MPI

Virtual topologies

Communication subgroups

Virtual topologies

MPI Communicator

- a collection of processes with a linear rank
- often a more complex ordering of processes is useful

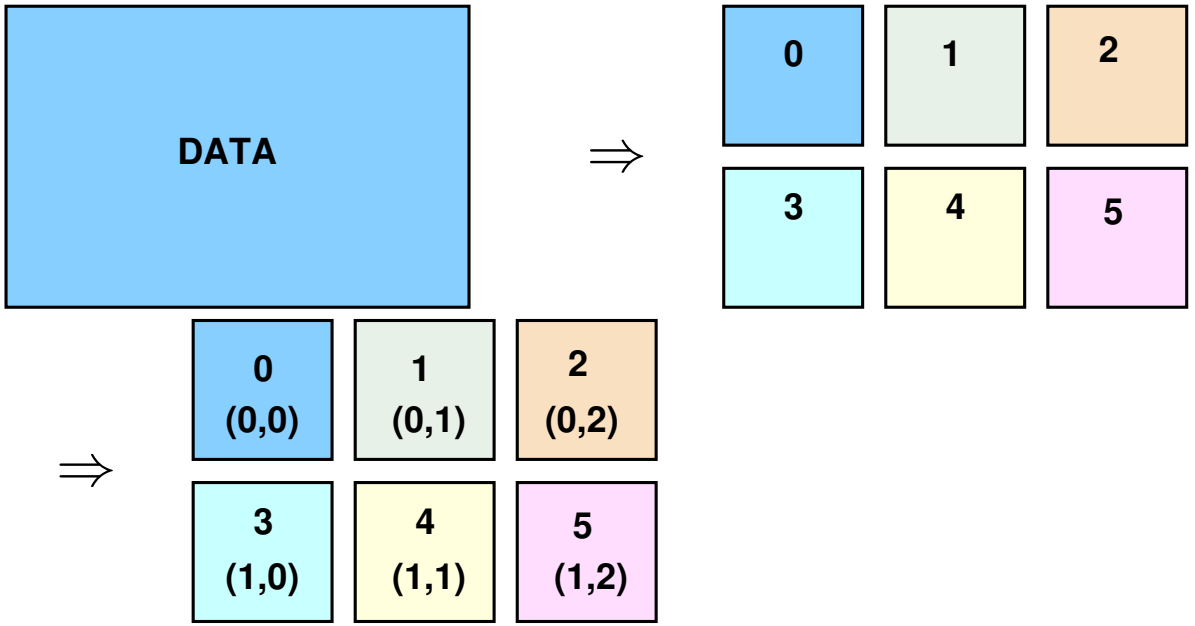
MPI virtual topology

- additional property attached to intracommunicator
- describes the topological interrelation between processes
- NOT related with physical topology of hardware (network)
- implementations of MPI might perform corresponding mapping

Virtual topologies

Layout of processors in relation to data

- example: 2D domain decomposition



- simplifies addressing of processors

MPI topologies

Cartesian topology constructor

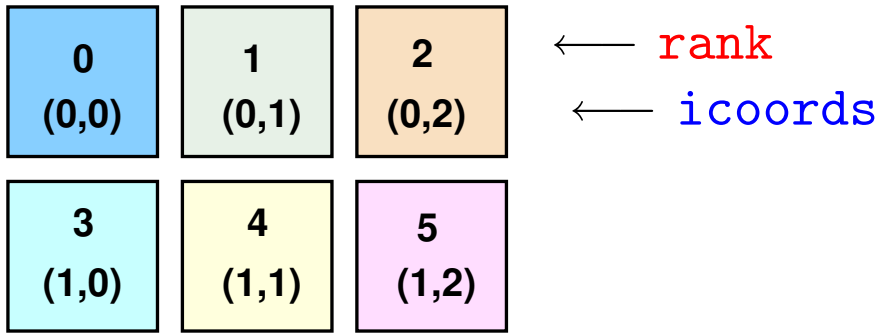
```
MPI_CART_CREATE(comm_old,ndims,idim,lperiod,reorder,new_comm,ierr)
```

- `comm_old` – existing communicator
- `ndims` – number of dimensions
- `idim(1:ndims)` – array of processes per dimension
- `lperiod(1:ndims)` – array of logicals:
  indicate periodicity in each dimension
- `reorder` – logical indicating whether the ranks in `new_comm` may be different
- `new_comm` – new communicator (output)

$\rightsquigarrow$ this is a collective call!

Example

2×3 processor grid

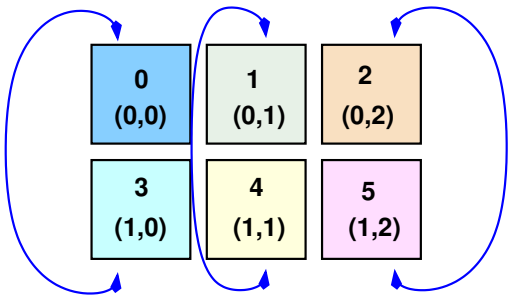



```
idim(1)=2
idim(2)=3
call MPI_CART_CREATE(MPI_COMM_WORLD,2,idim,lperiod,
                .FALSE.,my_comm,ierr)
call MPI_COMM_RANK(my_comm,rank,ierr)
call MPI_CART_COORDS(my_comm,rank,2,icoords,ierr)
```

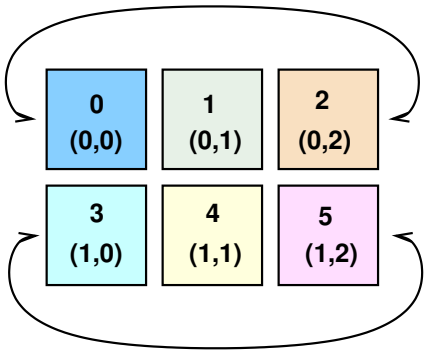
Example – continued

Periodicity

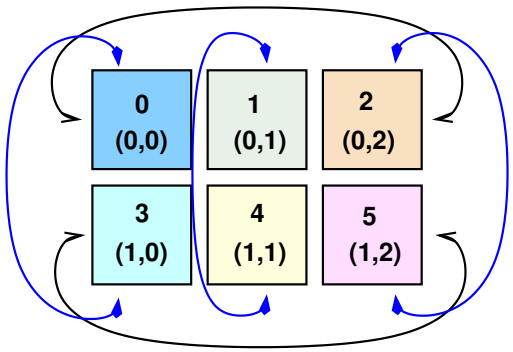
```
lperiod(1)=.TRUE.
lperiod(2)=.FALSE.
```




```
lperiod(1)=.FALSE.
lperiod(2)=.TRUE.
```




```
lperiod(1)=.TRUE.
lperiod(2)=.TRUE.
```



Cartesian query functions

Cartesian coordinates from process rank

- `call MPI_CART_COORDS(comm,rank,ndims,icoords,ierr)`
- returns the coordinates `icoords(1:ndims)` from given rank in communicator `comm`

Process rank from Cartesian coordinates

- inverse operation of `MPI_CART_COORDS`
- `call MPI_CART_RANK(comm,icoords,rank,ierr)`
- returns the process rank in communicator `comm` from given coordinates `icoords`

Cartesian shift function

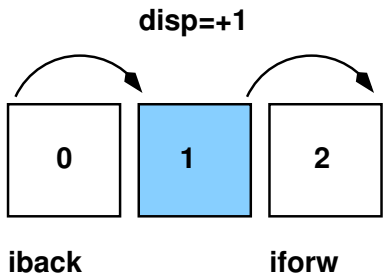
Need for communication with neighbors

- send/receive operations along each direction
- `MPI_CART_SHIFT` determines ranks of "neighbors"

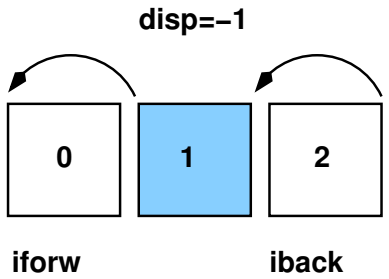
  `call MPI_CART_SHIFT(comm,dir,disp,iback,iforw,ierr)`
- `comm` – Cartesian communicator
- `dir` – coordinate direction of shift, range `(0:ndims-1)`
- `disp` – displacement (positive or negative integer)
- `iback` – rank of the neighbor process looking backward (out)
- `iforw` – rank of the neighbor process looking forward (out)
- this is a local call (query – no action)

Cartesian shift function – "neighbor" processes

Positive shift

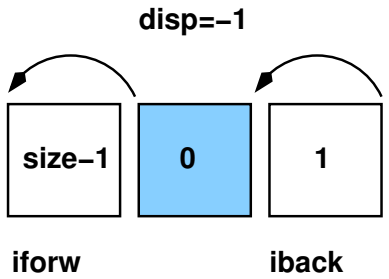


Negative shift

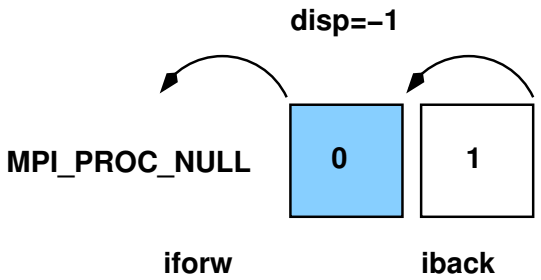


Cartesian shift function – "neighbor" processes

Negative shift, periodic case



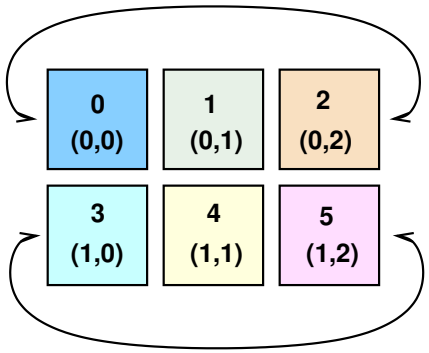
Off-end shift (non-periodic case)



Note: `MPI_PROC_NULL` is valid in all send/recv routines

Example

Result of shift operation




```
if rank=0:
iback=3
iforw=MPI_PROC_NULL
if rank=5:
iback=MPI_PROC_NULL
iforw=2  if
rank=0:
iback=2
iforw=1  if
rank=5:
iback=4
iforw=3
```

```
idim(1)=2
idim(2)=3
lperiod(1)=.FALSE.
lperiod(2)=.TRUE.
call MPI_CART_CREATE(MPI_COMM_WORLD,2,idim,lperiod,
                     FALSE, my_comm, ierr)
```

Communication subgroups

Need for defininining subgroup communicators

- different sets of processors are performing different tasks
- allow for collective communication of sub-groups
- increase efficiency of communication
- more readable code, less prone to bugs
- avoid message conflicts
  (e.g. isolate communication in libraries)

Constructing MPI Communicators

There is always one communicator `MPI_COMM_WORLD`

Methods for constructing a new communicators

- splitting an existing communicator
- duplicating an existing communicator
- modifying a group of processes
- reordering

Splitting existing MPI Communicators

Syntax:

```
call MPI_COMM_SPLIT(old_comm,color,key,new_comm,ierr)
```

- `old_comm` – existing communicator handle
- `color` – indicates new association (non-negative integer or `MPI_UNDEFINED`)
- `key` – indicates rank in new comm
- `new_comm` – the new communicator (output)
- collective call

Splitting an existing MPI Communicator

Example: odd/even communicator

```
color=mod(myrank,2)
key=myrank
call MPI_COMM_SPLIT(old_comm,color,key,new_comm,ierr)
call MPI_COMM_RANK(new_comm,myrank2,ierr)
```

- two communicators are created

myrank	0	1	2	3	← rank in old_comm
color	0	1	0	1	
myrank2	0	0	1	1	← rank in new_comm

(example code)

MPI group operations

An MPI group does not allow for communication

- serves as a definition of a collection of processes

Steps for creation of new communicator via group:

1. Extract a group (`MPI_COMM_GROUP`)
2. Modify the group
   `MPI_GROUP_INCL`, `MPI_GROUP_EXCL`, ...
3. Create a new communicator (`MPI_COMM_CREATE`)

MPI group operations

Example code:

```
call MPI_COMM_GROUP(MPI_COMM_WORLD,group,ierr)
ranks(1)=size-1
call MPI_GROUP_EXCL(group,1,ranks,group2,ierr)
call MPI_COMM_CREATE(MPI_COMM_WORLD,group2,comm2,ierr)
```

- generates communicator `comm2` with rank `(size-1)` excluded
- `MPI_COMM_CREATE` is collective
- all processes in old communicator need to participate
- excluded processes receive `comm2=MPI_COMM_NULL`

  (example code)

Exercise: Virtual topologies & communicators

Parallel search – problem statement

- search entries in a large table
- return indices and average with neighbor procs' first entry: `(index+aloc_left(1)+aloc_right(1))/3`
- processors are organized in a ring (1D Cartesian, periodic)
- communication between workers is isolated in communicator

Strategy

- modify previous code by using `MPI_CART_xx` functionality
- use either `MPI_COMM_SPLIT` or `MPI_COMM_CREATE`

  (previous source: struct, pack) (input data)

Code example: parallel search

Correct result (4 processors, 3 workers):

```
P 1 62 -0.333
P 3 271 22.666
P 3 291 22.666
P 3 296 22.666
P 2 183 3.666
```

(solution source)