TACC
TEXAS ADVANCED COMPUTING CENTER
WWW.TACC.UTEXAS.EDU

TEXAS
The University of Texas at Austin

MPI Programming

IHPCSS

Communicators

July 7-12, 2019

PRESENTED BY:

John Cazes

cazes@tacc.utexas.edu

Lecture and Lab slides available at: tinyurl.com/tacc-2019-ihpcss

Outline

Advantages of Message Passing
Background on MPI
Basic Information
Point to Point Communication
Nonblocking Communication
Wildcards
Probing
Collective Communications
"V" Operations
Derived Datatypes
Communicators

Communicators

A communicator is a “context” for communicating only among a group of tasks.

`MPI_COMM_WORLD` is the default communicator and consists of all tasks.

Communication is isolated to context of the group– i.e. no messages from other contexts are “seen”.

New communicators are formed recursively from `MPI_COMM_WORLD`.

Why Communicators?

Isolate communication to a small number of processors

Useful for creating libraries

Collective communication between subgroups (in lieu of all tasks) can drastically reduce communication costs if only some need to participate

Useful for communicating with "nearest neighbors"

Duplication

Simplest new communicator: identical to a previous one.

```
MPI_Comm_dup(MPI_Comm comm, MPI_Comm *newcomm)
```

This is useful for library writers:

```
MPI_Isend(...); MPI_Irecv(...);
// library call
MPI_Waitall(...);
```

Groups, Contexts, Communicators

- Group: An ordered set of processes, each associated with a rank (within a continuous range).
- Context: A property of a communicator that partitions the communication space.
  - Not externally visible.
  - Contexts allow Pt2Pt and collective calls not to interfere with each other.
- Communicator: Group + Context + other info
  - context that differentiates one communicator from another
  - group of processes contained by the communicator
  - Predefined: MPI_COMM_WORLD,MPI_COMM_SELF

Groups

A new communication group can only be created from a previously defined group. A group must also have a context for communication and, therefore, must have a communicator created for it. The basic steps to form a group are:

Obtain a complete set of task IDs from a communicator

```
MPI_Comm_group.
```

Create a group as a subset of the complete set by

```
MPI_Group_excl, MPI_Group_incl, ...
```

Create the new communicator for group (subset) using

```
MPI_Comm_create.
```

Group Constructors

Routine	Function
MPI_Comm_group	returns group reference of a communicator
MPI_Group_incl	forms new group from inclusion list
MPI_Group_excl	forms new group from exclusion list
MPI_Group_{union, intersection, difference}	Forms new group from union, intersection, or difference of 2 groups.
MPI_Comm_create	creates communicator from a group reference

Group Constructors

```
MPI_Comm_group(MPI_Comm comm, MPI_Group *group)

MPI_Group_union(MPI_Group group1, MPI_Group group2, MPI_Group *newgroup)

MPI_Group_intersection(MPI_Group group1, MPI_Group group2, MPI_Group *newgroup)

MPI_Group_difference(MPI_Group group1, MPI_Group group2, MPI_Group *newgroup)

MPI_Group_incl(MPI_Group group, int n, int *ranks, MPI_Group *newgroup)

MPI_Group_excl(MPI_Group group, int n, int *ranks, MPI_Group *newgroup)

MPI_Group_range_incl(MPI_Group group, int n, int ranges[][3], MPI_Group *newgroup)

MPI_Group_range_excl(MPI_Group group, int n, int ranges[][3], MPI_Group *newgroup)
```

Master/Worker C Example

```
main(int argc, char **argv) {
  int my_PE, send_data=1, recv_data1, recv_data2;
  MPI_Group group_world, group_worker;
  MPI_Comm  comm_worker;
  int ranks[] = {0};
  //Ranks to exclude, just 0 here
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_PE);
  MPI_Comm_group(MPI_COMM_WORLD, &group_world);
  MPI_Group_excl(group_world, 1, ranks, &group_worker);
  MPI_Comm_create(MPI_COMM_WORLD, group_worker, &comm_worker);
  //Reduce just for workers
  if(my_PE != 0)
    MPI_Reduce(&send_data, &recv_data1, 1, MPI_INT, MPI_SUM, 0,
comm_worker);
  MPI_Reduce(&send_data, &recv_data2, 1, MPI_INT, MPI_SUM, 0,
MPI_COMM_WORLD);
  //Proper clean up
  if (comm_worker != MPI_COMM_NULL)
    MPI_Comm_free(&comm_worker); //comm_worker is NULL on PE 0
  MPI_Group_free(&group_world); MPI_Group_free(&group_worker);
  MPI_Finalize(); }
```

Creating Communicators for Groups

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#define MAXEVEN 128
main(int argc, char **argv){
   int npes, irank,   ierr;
   int neven,iegid,   iogid,   i,  iranks[MAXEVEN];

   MPI_Group iegroup, iogroup, iwgroup;
   MPI_Comm  iecomm,  iocomm;

   ierr = MPI_Init(&argc, &argv);
   ierr = MPI_Comm_size(MPI_COMM_WORLD, &npes);
   ierr = MPI_Comm_rank(MPI_COMM_WORLD, &irank);

                        /* Extract group from World Comm. */

   ierr =  MPI_Comm_group(MPI_COMM_WORLD, &iwgroup);
```

Creating Communicators for Groups

```
                        /* Make list of even ranks. */

neven = (npes+1)/2;
if(neven > MAXEVEN) exit(1);
for(i=0; i < npes; i+=2) iranks[i/2] = i;

                        /* Form even and odd groups. */

ierr = MPI_Group_incl(iwgroup,neven,iranks, &iegroup);
ierr = MPI_Group_excl(iwgroup,neven,iranks, &iogroup);

ierr = MPI_Comm_create(MPI_COMM_WORLD,iegroup,&iecomm);
ierr = MPI_Comm_create(MPI_COMM_WORLD,iogroup,&iocomm);
```

Creating Communicators for Groups

```
    ierr = MPI_Group_rank(iegroup, &iegid);

       if(iegid != MPI_UNDEFINED) {
         printf("PE: %d, id %d of even group.\n", irank,iegid);
       }
       else {
         ierr =  MPI_Group_rank(iogroup, &iogid);
         printf("PE: %d, id %d of odd  group.\n", irank,iogid);
       }
    MPI_Comm_free( iecomm ); MPI_Comm_free( iocomm );
    MPI_Group_free(iegroup); MPI_Group_free(iogroup);

    ierr = MPI_Finalize();
}
```

MPI_Comm_split

Provides a short cut method to create a collection of communicators

All processors with the "same color" will be in the same communicator

Index controls relative rank in group

Fortran

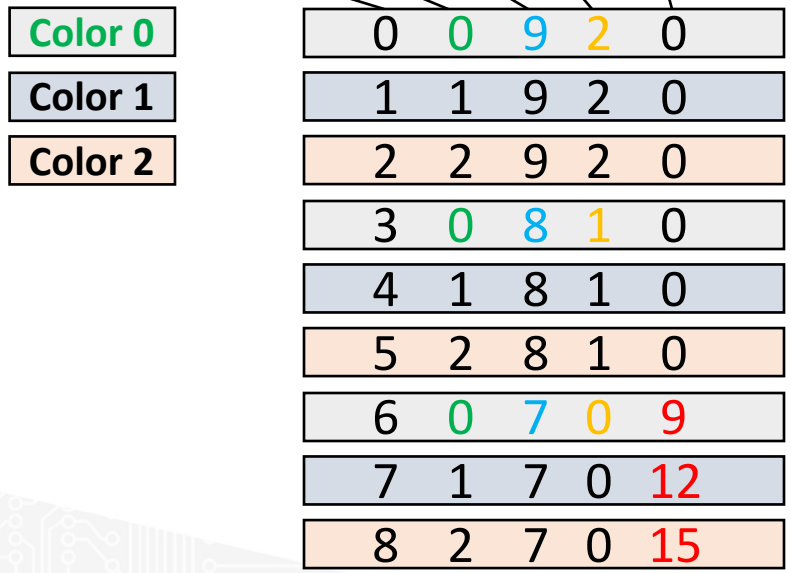
```
MPI_Comm_split(OLD_COMM, color, index,  NEW_COMM, ierr)
```

C

```
MPI_Comm_split(OLD_COMM, color, index, &NEW_COMM)
```

MPI_Comm_split

```
call MPI_Comm_rank(MPI_COMM_WORLD,irank,ierr)
icolor = modulo(irank,3)
key    = npes - irank/3   ! reverse the ordering

call MPI_Comm_split(MPI_COMM_WORLD, icolor, key, newcom, ierr)
call MPI_Comm_rank(newcom,mysrank,ierr)

psum  = irank
call MPI_Reduce(psum,tot,1,MPI_INTEGER,MPI_SUM,0,newcom,ierr)
print*, irank,icolor,key,mysrank,tot
```



Colors are 0, 1 and 2
Keys are 9, 8 and 7
Lowest keys are roots

(Virtual) Topologies

In terms of MPI, a virtual topology describes a **mapping** and **ordering** of MPI processes into a geometric shape.

The two main types of topology supported by MPI are **Cartesian**(grid) and Graph.

MPI topologies are **virtual** – there may be no relation between the physical structure of parallel machine and the process topology.

Virtual topologies are **built upon MPI communicator and groups**.

Must be ***built*** **by the application developer**.

Useful for applications with **specific communication pattern**.

A particular **implementation may optimize process mapping** based on the physical characteristics of a given parallel machine.

Topologies

Use the MPI library for common grid topologies (local functions)

A *topology* maps each rank onto a set of N-tuples (N dimensions)

E.g. For a 2x2 Cartesian grid:

Ranks:{0, 1, 2, 3}->{(0,0), (0,1),
(1,0), (1,1)} tuples (row-major in ranks)

Cartesian map operations:

MPI_Cart_create	Creates map (ranks → coordinates).
MPI_Cart_get	Returns info created in MPI_Cart_create.
MPI_Cart_coords	Returns coordinates from rank.
MPI_Cart_rank	Returns rank from coordinates.
MPI_Cart_shift	Returns rank of neighbors in nth dimension.

Note: the virtual topology does not necessarily map the hardware processor grid to the process grid in the most efficient manner.

Topologies

Use Case:

1. Build 2-/3-D Cartesian Grid
2. Use shift operator to get neighbor rank
   for destination and source
   in x, y and z directions

Creating a Cartesian Topology

C: `int MPI_Cart_create(MPI_Comm comm_old,`
`int ndims, int *dims, int *periods,`
`int reorder, MPI_Comm *comm_cart)`

Fortran: `MPI_CART_CREATE(COMM_OLD, NDIMS, DIMS,`
`PERIODS, REORDER, COMM_CART, IERROR)`
`INTEGER COMM_OLD, NDIMS, DIMS(*), COMM_CART, IERROR`
`LOGICAL PERIODS(*), REORDER`

```
comm_old: [in] input communicator
ndims:    [in] number of dimensions of cartesian grid
dims:     [in] number of processes in each dimension (size ndims)
periods:  [in] grid is periodic or not in each dimension (size ndims)
reorder:  [in] ranking may be reordered or not
comm_cart:[out]communicator with new cartesian topology
```

Balanced Processor Distribution

C: `int MPI_Dims_create( int nnodes, int ndims, int *dims )`
Fortran: `MPI_DIMS_CREATE( NNODES, NDIMS, DIMS, IERROR)`
`INTEGER NNODES, NDIMS, DIMS(*), IERROR`

```
nnodes: [in] number of nodes in a grid
ndims:  [in] number of cartesian dimensions
dims:   [in/out] number of nodes in each dimension (size ndims)
```

- Call tries to set dimensions as close to each other as possible
- Non-zero values in dims will not be changed

dims before call	function call	dims on return
(0, 0)	MPI_DIMS_CREATE(6, 2, dims)	(3, 2)
(0, 0)	MPI_DIMS_CREATE(7, 2, dims)	(7, 1)
(0, 3, 0)	MPI_DIMS_CREATE(6, 3, dims)	(2, 3, 1)
(0, 3, 0)	MPI_DIMS_CREATE(7, 3, dims)	erroneous call

Cartesian Query Functions

Library support and convenience!

- `MPI_Cartdim_get(comm_cart, ndims)`
  - Gets dimensions of a Cartesian communicator
- `MPI_Cart_get(comm_cart, ndims, dims, periods, coords)`
  - Gets size of dimensions
  - coords: coordinates of calling process in cartesian structure (array of integer)
- `MPI_Cart_rank(comm_cart, coords, rank)`
  - Translate coordinates to rank
- `MPI_Cart_coords(comm_cart, rank, ndims, coords)`
  - Translate rank to coordinates

Topologies (Shift)

C

```
MPI_Cart_shift(comm_cart, direct, disp, &rank_src, &rank_dst)
```

Fortran

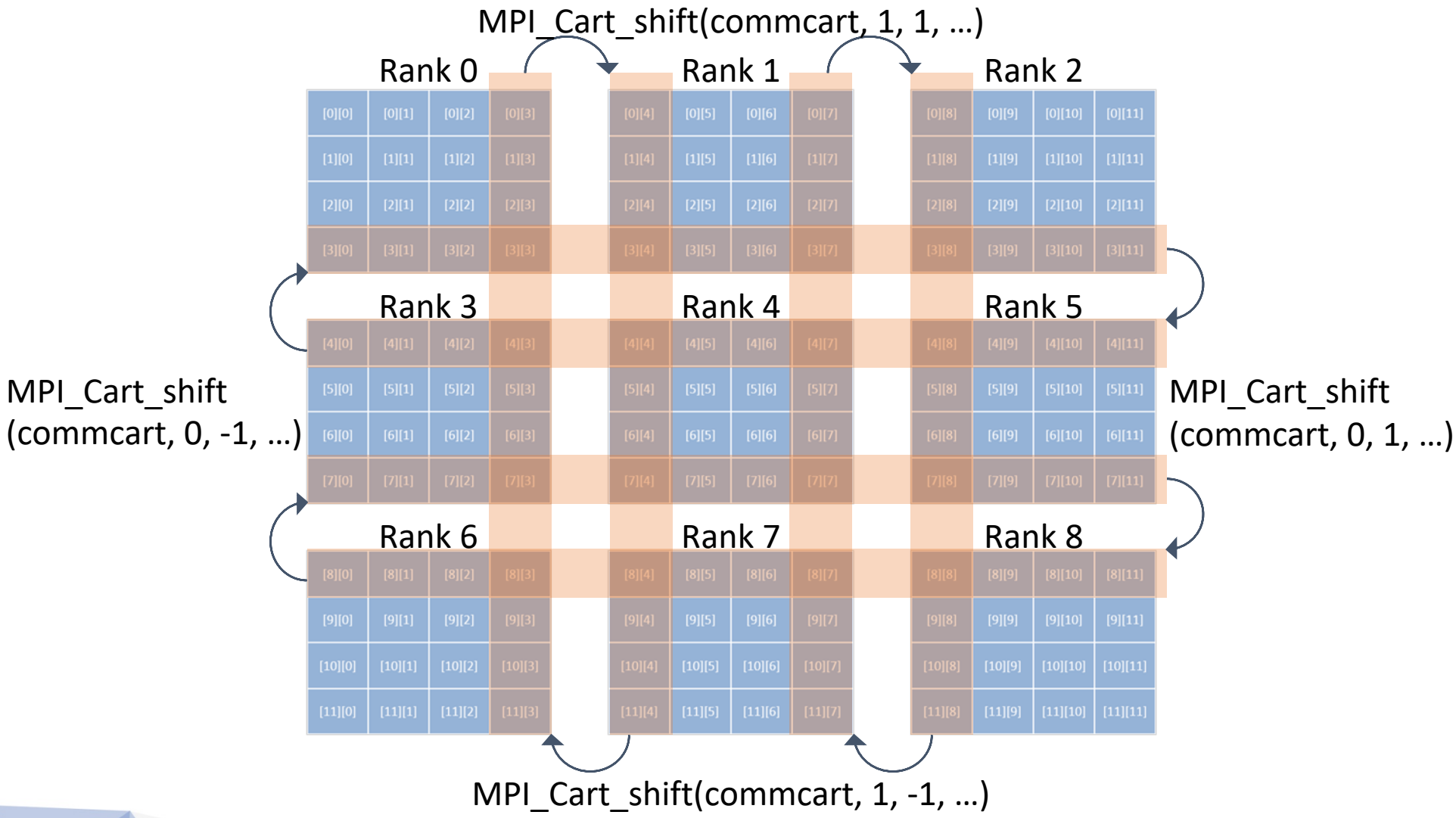
```
MPI_Cart_shift(comm_cart, direct, disp,  rank_src,  rank_dst,ierr)
```

Parameters

comm_cart: communicator with Cartesian structure
direct : coordinate dimension of shift
disp : >0 = upwards shift, <0 = downwards shift
end-off/circular shift (see periods of MPI_Cart_create)
rank_src : output: rank of source process
rank_dst : output: rank of destination process

MPI_PROC_NULL for shifts at non-periodic boundaries

Shift Example (non-periodic)



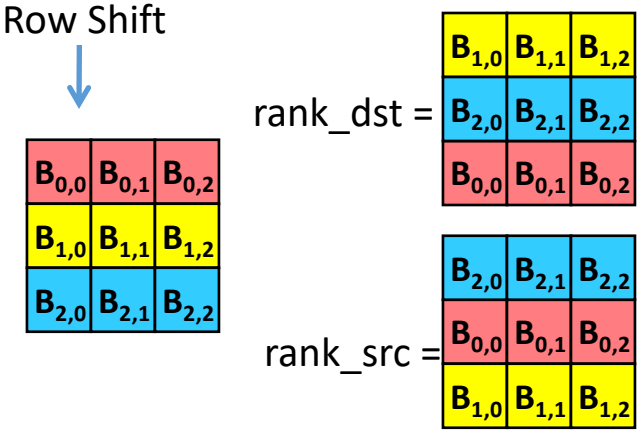
Topology Illustrations

Rank map onto 2-D Cartesion Topology

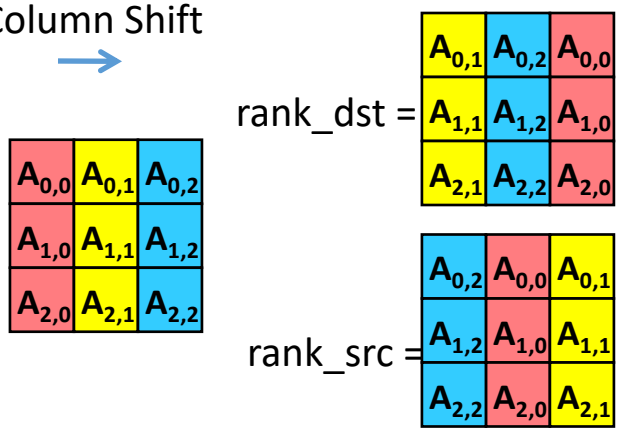
0 (0,0)	1 (0,1)	2 (0,2)
3 (1,0)	4 (1,1)	5 (1,2)
6 (2,0)	7 (2,1)	8 (2,2)

Column/Row Shift (periodic)

Periodic Displacement of 1 in Dimension “0”
Row Shift



Periodic Displacement of 1 in Dimension “1”
Column Shift



Fortran Example

```
 integer,parameter :: NP=3
 logical, dimension(2)   :: lvper=(/.true.,.true./), lvperx
 integer, dimension(2)   :: ivdim=(/    NP,    NP/), ivdimx
 integer, dimension(2)   :: mygrid
...
 call MPI_Cart_create(iwcomm,2,ivdim ,lvper, .false.,igcomm,ir)
 call MPI_Cart_get(   igcomm,2,ivdimx,lvperx, mygrid, ir)
 call MPI_Cart_shift( igcomm,1,1, isrca,idesa, ierr)
 call MPI_Cart_shift( igcomm,0,1, isrcb,idesb, ierr)
 myrow=mygrid(1);mycol=mygrid(2);mype=rank

 print*,'A:',isrca,')- ',mype,'[',myrow,',',mycol,'] ->',idesa,
& '      B:',isrcb,')- ',mype,'[',myrow,',',mycol,'] ->',idesb
```

Will receive from

Who I am

Will send to

```
        column shift @[0,0]                    row shift @[0,0]``
 A: 2)-  0 [ 0 , 0 ] -> 1          B: 6 )-  0 [ 0 , 0 ] -> 3
```

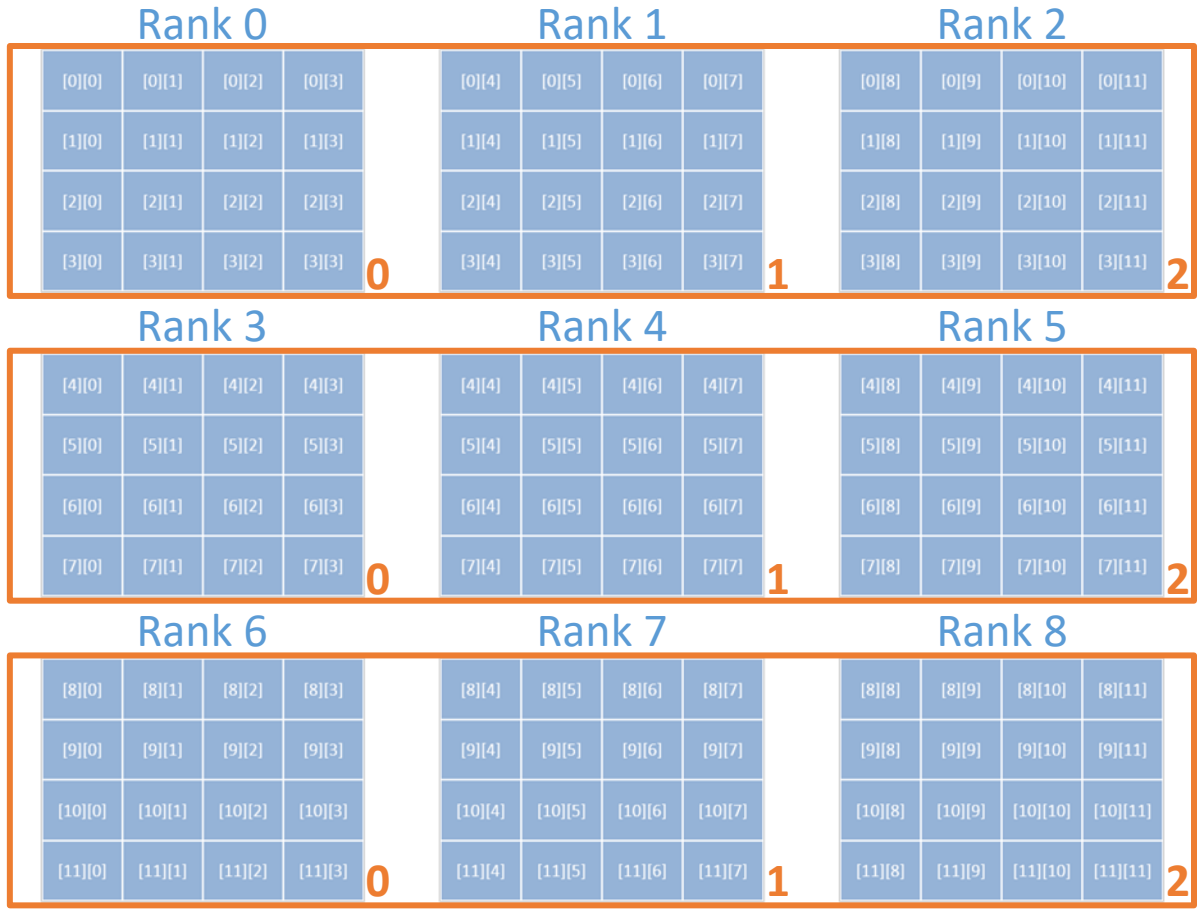
C Example

```
#include <mpi.h>
#include <stdio.h>
#define NP 3

main(int argc, char **argv){
   int npes,   mype, ierr,  myrow, mycol;
   int isrca, isrcb, idesa, idesb;
   MPI_Comm IWCOMM = MPI_COMM_WORLD, igcomm;
/*  MPI Cartesian Grid information */
   int  ivdim[2] = {NP,NP}, ivper[2]={1,1};
   int ivdimx[2],           ivperx[2], mygrids[2];
...
/*  Create Cartesian Grid and extract information */

    ierr= MPI_Cart_create(IWCOMM,2,ivdim ,ivper, 0,&igcomm);
    ierr= MPI_Cart_get(   igcomm,2,ivdimx,ivperx, mygrids);
    ierr= MPI_Cart_shift( igcomm,1,1, &isrca,&idesa);
    ierr= MPI_Cart_shift( igcomm,0,1, &isrcb,&idesb);
```

Generic Example: Send “a” blocks down, and “b” blocks right.

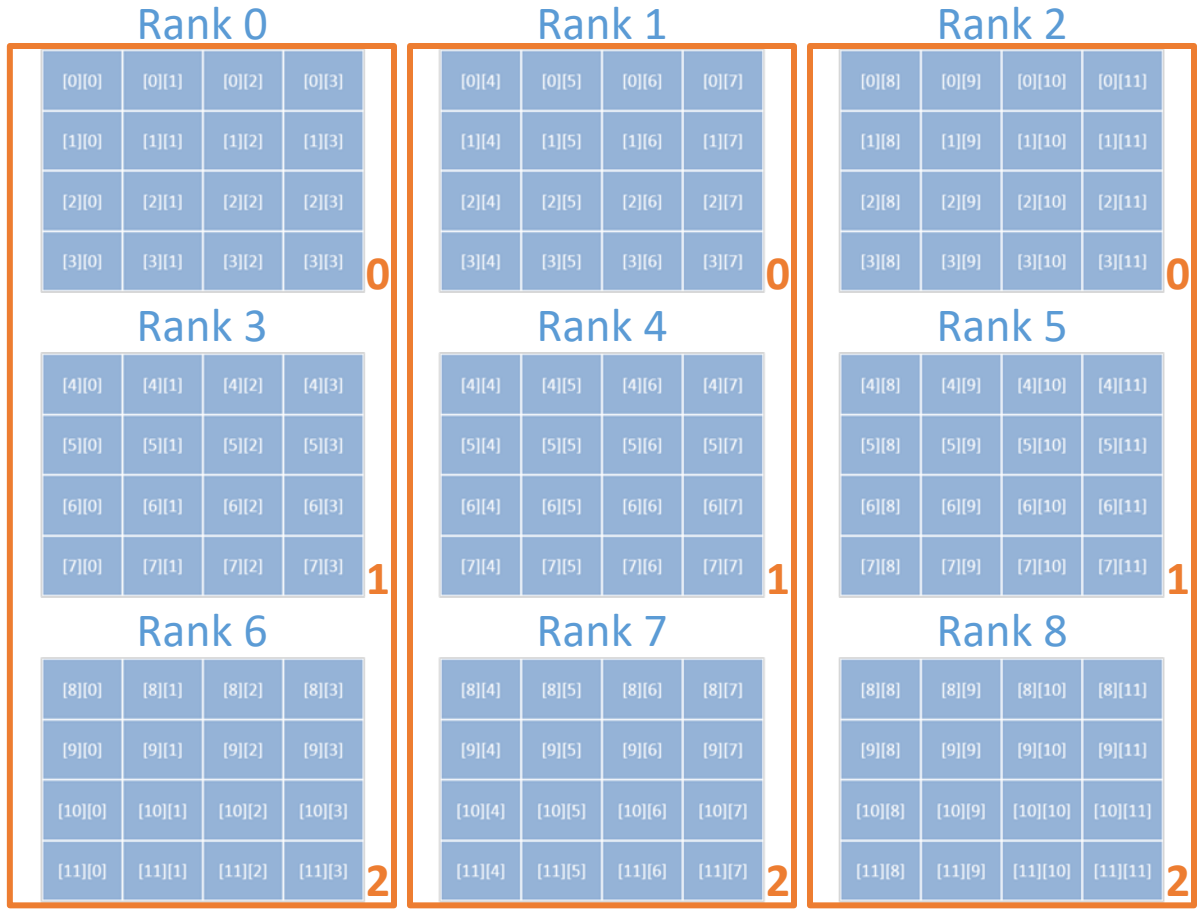
```
MPI_CART_SHIFT(cartcomm, 0, 1, UP,   DOWN   )
MPI_CART_SHIFT(cartcomm, 1, 1, LEFT, RIGHT  )

…

MPI_ISEND(a1, N,MPI_INTEGER, DOWN,  1,MPI_COMM_WORLD,reqa1  )
MPI_IRECV(a2, N,MPI_INTEGER, UP,    1,MPI_COMM_WORLD,reqa2  )

…

MPI_ISEND(b1, N,MPI_INTEGER, RIGHT, 2,MPI_COMM_WORLD,reqb1  )
MPI_IRECV(b2, N,MPI_INTEGER, LEFT,  2,MPI_COMM_WORLD,reqb2  )
```

Cartesian subspaces




```
MPI_Cart_sub (comm_cart, remain_dims, comm_sub)
                         {1, 0}
                         (true, false)
```

Cartesian subspaces




```
MPI_Cart_sub (comm_cart, remain_dims, comm_sub)
                         {0, 1}
                         (false, true)
```

Outline

Advantages of Message Passing
Background on MPI
Basic Information
Point to Point Communication
Nonblocking Communication
Wildcards
Probing
Collective Communications
"V" Operations
Derived Datatypes
Communicators

License