



Intermediate MPI

Le Yan

HPC Consultant
HPC @ LONI & LSU





Goals of training

- Acquaint users with derived data types in MPI
- Acquaint users with the basics of MPI communicators





Outline

- Basic MPI recap
- Derived data type - communicate non-contiguous data
- Communicators





Outline

- Basic MPI recap
- Derived data type - communicate non-contiguous data
- Communicators





What is MPI

- Context
 - Distributed memory parallel computers
- MPI is a standard
 - What is in the standard: the syntax and semantics of a set of core functions
 - What is NOT in the standard: how to compile and link MPI programs and how many processors to use





MPI Program Structure

```
! MPI header file
include 'mpif.h'
...
! MPI initialization
call mpi_init(ierr)
...
call mpi_comm_size(comm,size,ierr)
call mpi_comm_rank(comm,rank,ierr)
...
<Main code>
...
! MPI termination
call mpi_finalize(ierr)
...
```





Point-to-point communication

- Fundamental message passing function
- One process sends message and another process receive it
- Blocking vs. Non-blocking





Collective Communication

- Collective communications involve all processes in a communicator
- Must be called by all involved processes
- All collective communications are blocking





Outline

- Basic MPI recap
- Derived data type - communicate non-contiguous data
- Communicators





Basic Data Types

MPI datatype	C datatype
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	singed int
MPI_LONG	singed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

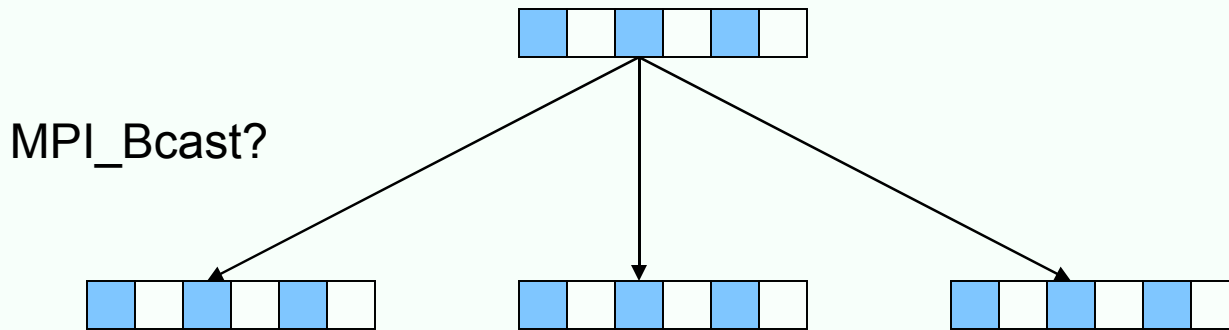
MPI datatype	Fortran datatype
MPI_INTEGER	INTEGER
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_LOGICAL	LOGICAL
MPI_CHARACTER	CHARACTER(1)
MPI_BYTE	
MPI_PACKED	





Why Derived Data Types?

- Basic communication calls so far have involved only **contiguous** data with a sequence of elements of a single type.
- What if the data to be transferred is not contiguous? Or not of the same type?





Possible solutions for non-contiguous data transfer (1)

- Make multiple communication calls

```
...  
call mpi_bcast(a(1), 1, mpi_integer, ...)  
call mpi_bcast(a(3), 1, mpi_integer, ...)  
call mpi_bcast(a(5), 1, mpi_integer, ...)  
...
```





Possible solutions for non-contiguous data transfer (2)

- Pack data into contiguous buffers manually

```
...  
tmp(1) = a(1)  
tmp(2) = a(3)  
tmp(3) = a(5)  
call mpi_bcast(tmp, 3, mpi_integer, ...)  
a(1) = tmp(1)  
a(3) = tmp(2)  
a(5) = tmp(3)  
...
```





Possible solutions for non-contiguous data transfer (3)

- Use derived data types
 - Tell the library what is desired
 - Let the library decide how the communication is done

```
...  
call mpi_type_vector(3,1,2,mpi_integer,newtype,ierr)  
call mpi_type_commit(newtype,ierr)  
call mpi_bcast(a(1),1,newtype,...)  
...
```






MPI_TYPE_CONTIGUOUS

- Allows replication of one data type into contiguous locations
- `MPI_TYPE_CONTIGUOUS(count, oldtype, newtype, ierror)`

```
CALL MPI_TYPE_CONTIGUOUS(4, MPI_INTEGER, NEWTYPE, IERR)
CALL MPI_TYPE_COMMIT(NEWTYPE, IERR)
```

MPI_INTEGER: 

NEWTYPE: 



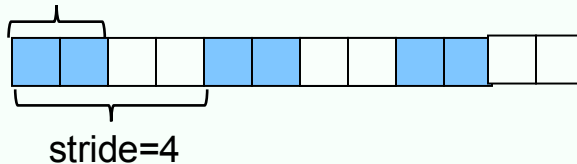


MPI_TYPE_VECTOR

- Allows replication of a datatype into locations that consist of equally spaced blocks
- `MPI_TYPE_VECTOR(count, blocklength, stride, oldtype, newtype, ierror)`

```
CALL MPI_TYPE_VECTOR(3,2,4,MPI_INTEGER,NEWTYP, IERR)
CALL MPI_TYPE_COMMIT(NEWTYP, IERR)
```

blocklength=2





Example: MPI_TYPE_VECTOR

```
...  
Integer a(6)  
...  
if (myrank.eq.0) then  
  do i=1,6  
    a(i)=i  
  enddo  
else  
  a=0  
endif  
call mpi_type_vector(3,1,2,mpi_integer,newtype,ierr)  
call mpi_type_commit(newtype,ierr)  
call mpi_bcast(a(1),1,newtype,0,mpi_comm_world,ierr)  
print
```





Example: MPI_TYPE_VECTOR

```
...  
Integer a(6)  
...  
if (myrank.eq.0) then  
  do i=1,6  
    a(i)=i  
  enddo  
else  
  a=0  
endif  
call mpi_type_vector(3,1,2,mpi_integer,newtype,ierr)  
call mpi_type_commit(newtype,ierr)  
call mpi_bcast(a(1),1,newtype,0,mpi_comm_world,ierr)  
print
```





Example: MPI_TYPE_VECTOR

```

...
Integer a(6)
...
if (myrank.eq.0) then
  do i=1,6
    a(i)=i
  enddo
else
  a=0
endif
call mpi_type_vector(3,1,2,mpi_integer,new
call mpi_type_commit(newtype,ierr)
call mpi_bcast(a(1),1,newtype,0,mpi_comm_w
print

```

Output (4 processes):

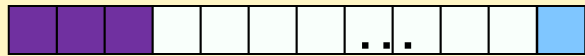
1	2	3	4	5	6
1	0	3	0	5	0
1	0	3	0	5	0
1	0	3	0	5	0





MPI_TYPE_STRUCT

- Most general data type constructor
- Allows a new data type that represents arrays of types, each of which has a different block length, displacement (in bytes) and type.



Struct: Count = 2, array_of_blocklengths={3, 1},
Array_of_displacements={0,12} (in bytes)
array_of_types={MPI_COMPLEX,MPI_INTEGER}

```
CALL MPI_TYPE_STRUCT(2, ARRAY_BLOCKLEN, ARRAY_DISP, ARRAY_TYPE, NEWTYPE, IERR)
CALL MPI_TYPE_COMMIT(NEWTYPE, IERR)
```





Example: broadcast a submatrix

	1	2	3	4	5	6	7
1							
2							
3							
4							
5							
6							
7							
8							

```

!Number of blocks
NBLCK=3
!Block length
BLCKLEN=4
!Stride
STRD=8

!Derived datatype
CALL MPI_TYPE_VECTOR(NBLCK, BLCKLEN, STRD,
&                   MPI_INTEGER, SUBMAT, IERR)
CALL MPI_TYPE_COMMIT(SUBMAT, IERR)

!Broadcast
CALL MPI_BCAST(AMAT(2, 2), 1, SUBMAT, 0,
&             MPI_COMM_WORLD, IERR)
    
```





Outline

- Basic MPI recap
- Derived data type - communicate non-contiguous data
- Communicators





Communicators

- A communicator is an identifier associated with a group of processes with certain attributes
 - Can think of it as an ordered list of processes
 - Each process has a unique rank (the rank starts from 0)
 - It is the context of MPI communications
 - MPI cannot understand “get this message to **all** processes” or “get this message from process **#1** to process **#2**”, unless a context is specified.





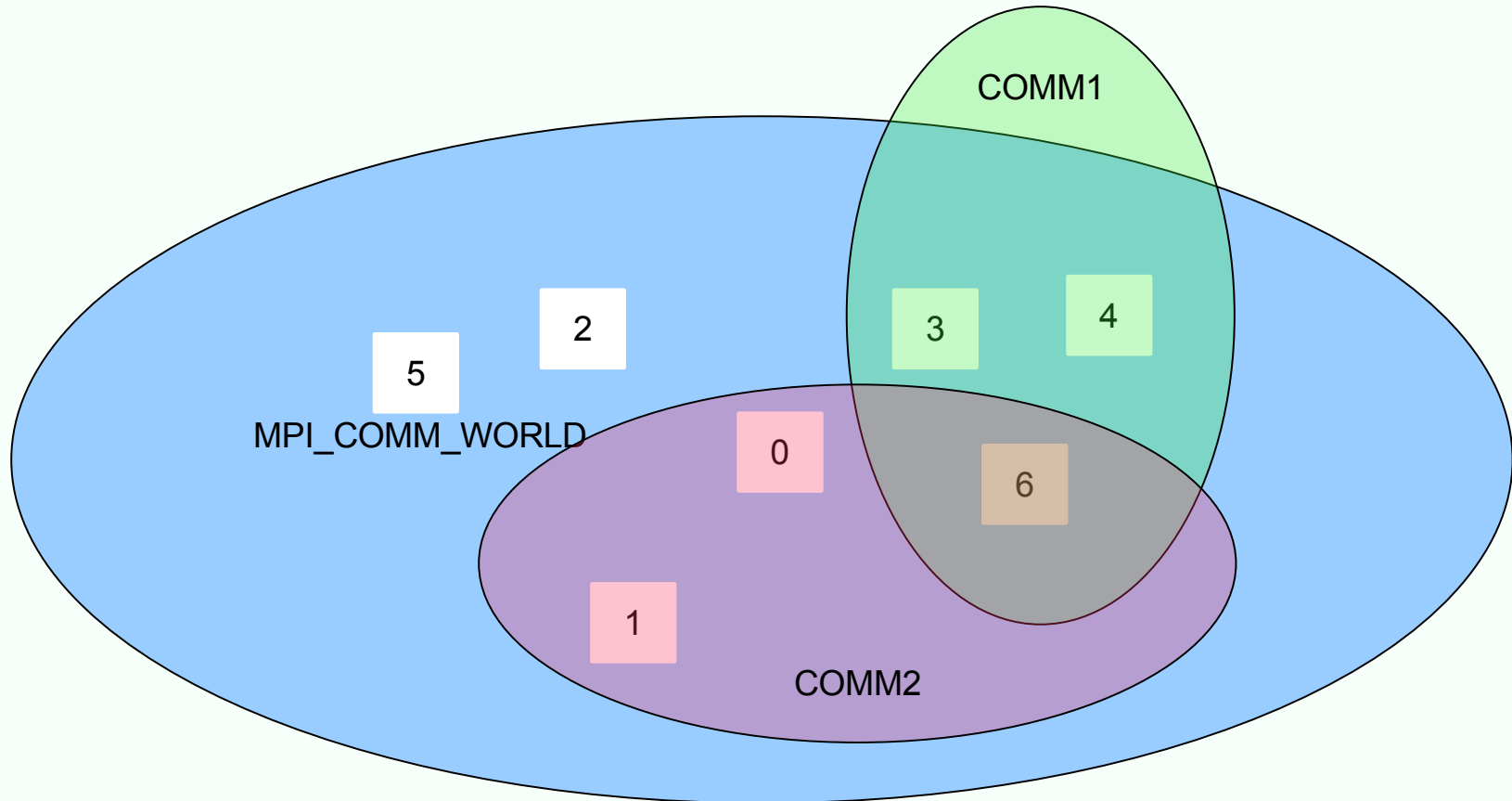
More on Communicators

- MPI_COMM_WORLD
 - The default communicator
 - Contains all processes
- Communicators in addition to MPI_COMM_WORLD
 - Useful when communications need to occur among a subset of the processes





Communicators





Create New Communicators (1)

- Split an existing communicator
 - Syntax: `MPI_COMM_SPLIT(Int comm, Int color, Int key, Int newcomm)`
 - Partitions the group associated with `comm` into disjoint subgroups, one for each value of `color`
 - Each subgroup contains all processes of the same `color`
 - Within each subgroup, the processes are ranked in the order defined by the value of the argument `key`, with ties broken according to their rank in the old group
 - A new communicator `newcomm` is created for each subgroup





Example: MPI_COMM_SPLIT

```
...  
call mpi_comm_rank(mpi_comm_world,myoldrank,ierr)  
  
color=mod(myoldrank,2)  
  
call mpi_comm_split(mpi_comm_world,color,myoldrank,newcomm,ierr)  
  
call mpi_comm_rank(newcomm,mynewrank,ierr)  
  
write(*,*) myoldrank,mynewrank  
...
```





Example: MPI_COMM_SPLIT

```
...  
call mpi_comm_rank(mpi_comm_world,myoldrank,ierr)  
  
color=mod(myoldrank,2)  
  
call mpi_comm_split(mpi_comm_world,color,myoldrank,newcomm,ierr)  
  
call mpi_comm_rank(newcomm,mynewrank,ierr)  
  
write(*,*) myoldrank,mynewrank  
...
```

Output (4 processes):

```
0 0  
1 0  
2 1  
3 1
```





Example: MPI_COMM_SPLIT

```

...
call mpi_comm_rank(mpi_comm_world, myoldrank, ierr)
color=mod(myoldrank,2)
call mpi_comm_split(mpi_comm_world, color, 2, newcomm, ierr)
call mpi_comm_rank(newcomm, mynewrank, ierr)
write(*,*) myoldrank,mynewrank

call mpi_bcast(a,1,mpi_integer,0,newcomm,ierr)
...

```

rank	a	a
in	before	after
MPI_COMM_WORLD	bcast	bcast
0	2	?
1	4	?
2	6	?
3	8	?

What will happen with this mpi_bcast call?





Example: MPI_COMM_SPLIT

```

...
call mpi_comm_rank(mpi_comm_world, myoldrank, ierr)
color=mod(myoldrank,2)
call mpi_comm_split(mpi_comm_world, color, 2, newcomm, ierr)
call mpi_comm_rank(newcomm, mynewrank, ierr)
write(*,*) myoldrank,mynewrank

call mpi_bcast(a,1,mpi_integer,0,newcomm,ierr)
...

```

rank in MPI_COMM_WORLD	a before bcast	a after bcast
0	2	2
1	4	4
2	6	2
3	8	4

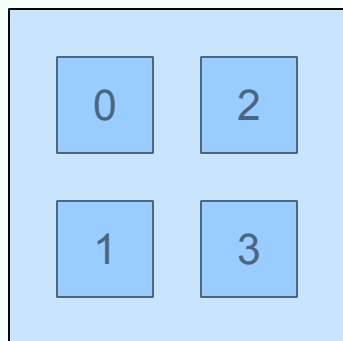




Example: MPI_COMM_SPLIT

- Apparently we created two new communicators with the `mpi_comm_split` call
- But from the point of view of each process, there is only ONE

MPI_COMM_WORLD



MPI_COMM_WORLD

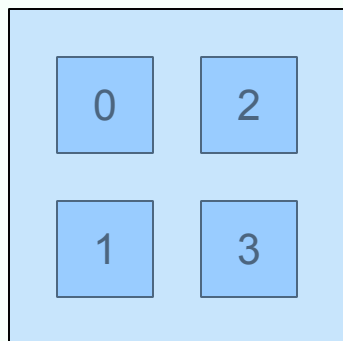




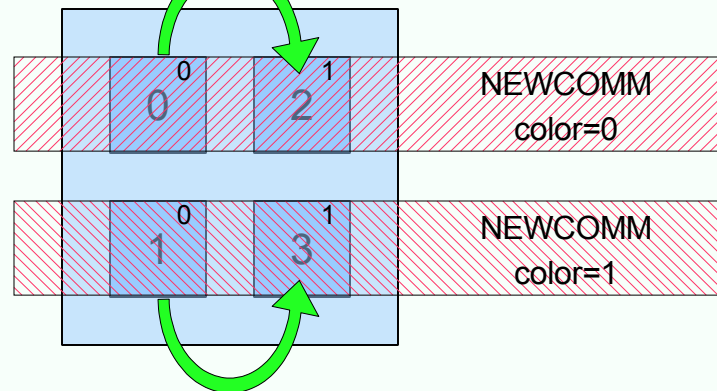
Example: MPI_COMM_SPLIT

- Apparently we created two new communicators with the `mpi_comm_split` call
- But from the point of view of each process, there is only ONE

MPI_COMM_WORLD



MPI_COMM_WORLD





Create new communicators (2)

- Map the old communicator to a group
 - MPI_COMM_GROUP
- Manipulate the group
 - Include, exclude, union, intersection etc.
- Create a new communicator from the modified group
 - MPI_COMM_CREATE





Example: Create A New Communicator

```
...  
  
call mpi_comm_rank(mpi_comm_world,myoldrank,ierr)  
  
! Map MPI_COMM_WORLD to the group ``oldgroup''  
call mpi_comm_group(mpi_comm_world,oldgroup,ierr)  
  
! Every process is included in ``newgroup'' except process 0  
rank_excl=0  
call mpi_group_excl(oldgroup,1,rank_excl,newgroup,ierr)  
  
! Create a new communicator  
call mpi_comm_create(mpi_comm_world,newgroup,newcomm,ierr)  
  
...
```





Example: Create A New Communicator

```
...  
  
call mpi_comm_rank(mpi_comm_world,myoldrank,ierr)  
  
! Map MPI_COMM_WORLD to the group ``oldgroup''  
call mpi_comm_group(mpi_comm_world,oldgroup,ierr)  
  
! Every process is included in ``newgroup'' except process 0  
rank_excl=0  
call mpi_group_excl(oldgroup,1,rank_excl,newgroup,ierr)  
  
! Create a new communicator  
call mpi_comm_create(mpi_comm_world,newgroup,newcomm,ierr)  
  
! What will happen?  
call mpi_comm_rank(newcomm,mynewrank,ierr)  
  
...
```





Example: Create A New Communicator

```
...  
  
call mpi_comm_rank(mpi_comm_world,myoldrank,ierr)  
  
! Map MPI_COMM_WORLD to the group ``oldgroup''  
call mpi_comm_group(mpi_comm_world,oldgroup,ierr)  
  
! Every process is included in ``newgroup'' except process 0  
rank_excl=0  
call mpi_group_excl(oldgroup,1,rank_excl,newgroup,ierr)  
  
! Create a new communicator  
call mpi_comm_create(mpi_comm_world,newgroup,newcomm,ierr)  
  
! What will happen?  
call mpi_comm_rank(newcomm,mynewrank,ierr)  
  
...  
  
ERROR!
```





Group management functions

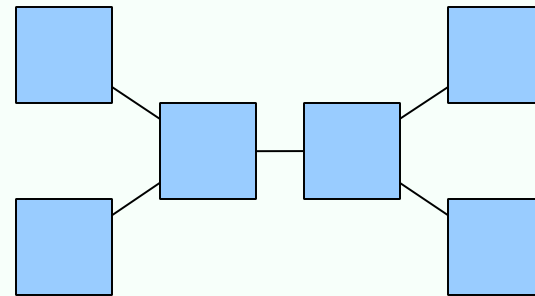
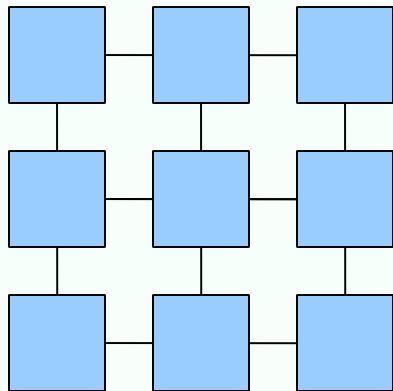
- `MPI_Group_union(MPI_Group group1, MPI_Group group2, MPI_Group *newgroup)`
- `MPI_Group_intersection(MPI_Group group1, MPI_Group group2, MPI_Group *newgroup)`
- `MPI_Group_difference(MPI_Group group1, MPI_Group group2, MPI_Group *newgroup)`
- `MPI_Group_incl(MPI_Group group, int n, int *ranks, MPI_Group *newgroup)`
 - New group with `n` elements of `group`
- `MPI_Group_excl(MPI_Group group, int n, int *ranks, MPI_Group *newgroup)`
 - New group with all but `n` elements of `group`





Topology

- An attribute of MPI communicators in addition to group
- Can be either Cartesian or Graph
- Useful for domain decomposition





Example: Create a 2-D Cartesian Topology

```
...  
! Create a 2-D cartesian topology  
call mpi_cart_create(mpi_comm_world, ndim, dim_size,  
                    periods, reorder, newcomm, ierr)  
  
! Returns the coordinates of the given rank  
call mpi_cart_coords(newcomm, rank, ndims, coords, ierr)  
  
! Found the neighbor processes along the given direction  
call mpi_cart_shift(newcomm, direction, displacement,  
                   source_rank, destination_rank, ierr)  
...
```





Questions?

