PCSE Lecture 10

MPI Data Structures and Communicators

Cyrus Proctor
Victor Eijkhout

April 7, 2015

MPI Datatypes

- MPI offers a suitable collection of basic datatypes to use in messages
- Data to be communicated must be in contiguous memory locations
- Data must consist of just one basic type and no other

Solution? MPI Derived Datatypes!

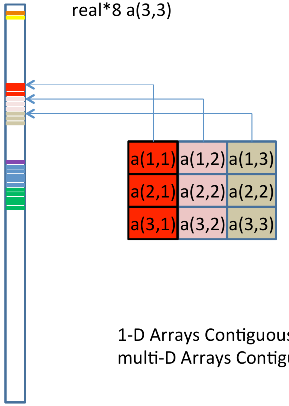
- Derived datatypes can be built up recursively
- They can be created conditionally, at runtime
- The associated packing and unpacking are done for you automatically

Motivational Example

- Create 2D matrix (matrix[NROWS][NCOLS]) (matrix(NROWS)(NCOLS))
- First, send entire 2D matrix from process a to process b
- Second, send one row of 2D matrix from process a to process b
- Third, send one column of 2D matrix from process a to process b

Fortran Language

```
real*8  sa, sb
real*8 sc, d1(5),d2(5)
real*8 a(3,3)
```

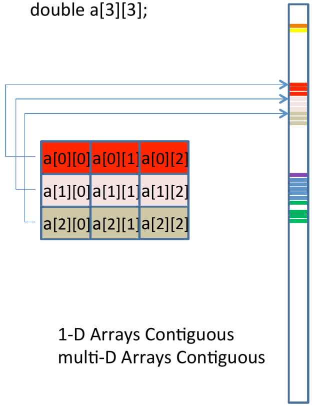


1-D Arrays Contiguous
multi-D Arrays Contiguous

Memory Layout for Compiled Program

C Language

```
double  sa, sb;
double sc, d1[5],d2[5];
double a[3][3];
```



1-D Arrays Contiguous
multi-D Arrays Contiguous

Memory Layout for Compiled Program

C – Send Full 2D Matrix

```c
// Sending a full matrix from process 0 to process 1
#define NROWS 3
#define NCOLS 4
  double matrix[NROWS][NCOLS] = { 0 };

  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);

  // Initialize matrix on process 0 only
  // Print matrix before Send/Recv
  // Send/Recv
  tag1 = 1; dest=1; src=0;
  if(rank == 0) {
    printf("\nCompleting Send/Recv from");
    printf(" %i to %i...\n\n",src,dest);
  }
  if(rank == 0){
    MPI_Send(&matrix[0][0],NROWS*NCOLS,MPI_DOUBLE,
             dest,tag1,MPI_COMM_WORLD);
  }
  if(rank == 1){
    MPI_Recv(&matrix[0][0],NROWS*NCOLS,MPI_DOUBLE,
             src,tag1,MPI_COMM_WORLD,&Stat);
  }

  // Print matrix after Send/Recv
  ...
```

Output – 2 Processes

```
(Rank 0): Matrix:
   0    1    2    3
   4    5    6    7
   8    9   10   11
(Rank 1): Matrix:
   0    0    0    0
   0    0    0    0
   0    0    0    0

Completing Send/Recv from 0 to 1...

(Rank 0): Matrix:
   0    1    2    3
   4    5    6    7
   8    9   10   11
(Rank 1): Matrix:
   0    1    2    3
   4    5    6    7
   8    9   10   11
```

C – Send One Row (default)

```
// Default way of sending contingous one row of 2D array
#define NROWS 3
#define NCOLS 4
  double matrix[NROWS][NCOLS] = { 0 };

  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);

  // Initialize matrix on process 0 only
  // Print matrix before Send/Recv
  // Send/Recv
  tag1 = 1; dest=1; src=0; row=1;
  if(rank == 0) {
    printf("\nCompleting Send/Recv from");
    printf(" %i to %i of row %i...\n\n",src,dest,row);
  }
  if(rank == 0){
    MPI_Send(&matrix[row][0],NCOLS,MPI_DOUBLE,
             dest,tag1,MPI_COMM_WORLD);
  }
  if(rank == 1){
    MPI_Recv(&matrix[row][0],NCOLS,MPI_DOUBLE,
             src,tag1,MPI_COMM_WORLD,&Stat);
  }

  // Print matrix after Send/Recv
  ...
```

Output – 2 Processes

```
(Rank 0): Matrix:
   0   1   2   3
   4   5   6   7
   8   9  10  11
(Rank 1): Matrix:
   0   0   0   0
   0   0   0   0
   0   0   0
Completing Send/Recv from 0 to 1 of
      row 1...

   0
(Rank 0): Matrix:
   0   1   2   3
   4   5   6   7
   8   9  10  11
(Rank 1): Matrix:
   0   0   0   0
   4   5   6   7
   0   0   0   0
```

Derived types
Three main classifications

- Contiguous Arrays (easy to use)
  - send contiguous blocks of the same datatype
- Noncontiguous Vectors (relatively easy to use)
  - send noncontiguous blocks of the same datatype
- Abstract types (more difficult)
  - send C or Fortran 90 structures

Derived types

• Elementary:	MPI names for language types
• Contiguous:	Array with stride of one
• Vector:	Array separated by constant stride
• Hvector:	Vector, with stride in bytes
• Indexed:	Array of indices (like gatherv)
• Hindexed:	Indexed, with displacements in bytes
• Struct:	General mixed types (C structs etc.)

- Pack and Unpack

Derived types, how to use them

- Three step process
- Define the type (e.g.)

  `MPI_Type_contiguous` for contiguous arrays

  `MPI_Type_vector` for noncontiguous arrays

  `MPI_Type_struct` for structures
- Commit the type
  - Tells MPI when to compile an internal representation

  `MPI_Type_commit(... my_type...)`
- Use in normal communication calls

```
MPI_Send( data, count, my_type,
                 dest, tag, comm …)
```

- Free space when done:
  `MPI_Type_free`

Contiguous Type Example #1

C – Send One Row (MPI_Type_contiguous)

```c
// Using MPI_Type_contiguous to send one row of 2D array
#define NROWS 3
#define NCOLS 4
  double matrix[NROWS][NCOLS] = { 0 };
  // Initialize matrix on process 0 only
  // Print matrix before Send/Recv
  // Create derived contiguous type
  MPI_Datatype MY_MPI_ROW;
  MPI_Type_contiguous(NCOLS,MPI_DOUBLE,&MY_MPI_ROW);
  MPI_Type_commit(&MY_MPI_ROW);
  // Send/Recv
  tag1 = 1; dest=1; src=0; row=1;
  if(rank == 0) {
    printf("\nCompleting Send/Recv from");
    printf(" %i to %i of row %i...\n\n",src,dest,row);
  }
  if(rank == 0){
    MPI_Send(&matrix[row][0],1,MY_MPI_ROW,
             dest,tag1,MPI_COMM_WORLD);
  }
  if(rank == 1){
    MPI_Recv(&matrix[row][0],1,MY_MPI_ROW,
             src,tag1,MPI_COMM_WORLD,&Stat);
  }

  // Print matrix after Send/Recv
  ...
```

Output – 2 Processes

```
(Rank 0): Matrix:
   0    1    2    3
   4    5    6    7
   8    9   10   11
(Rank 1): Matrix:
   0    0    0    0
   0    0    0    0
   0    0    0    0

Completing Send/Recv from 0 to 1 of
     row 1...

(Rank 0): Matrix:
   0    1    2    3
   4    5    6    7
   8    9   10   11
(Rank 1): Matrix:
   0    0    0    0
   4    5    6    7
   0    0    0    0
```

Continguous Type Example #2

C – Send One Row from 2D Array to 1D Array

```
// Using MPI_Type_contiguous to send one row of 2D array
      to 1D array
#define NROWS 3
#define NCOLS 4
  double matrix[NROWS][NCOLS] = { 0 };
  double vector[NCOLS] = { 0 };
  // Initialize matrix on process 0 only
  // Print matrix before Send/Recv
  // Print vector before Send/Recv
  // Create derived contiguous type
  MPI_Datatype MY_MPI_ROW;
  MPI_Type_contiguous(NCOLS,MPI_DOUBLE,&MY_MPI_ROW);
  MPI_Type_commit(&MY_MPI_ROW);
  // Send/Recv
  tag1 = 1; dest=1; src=0; row=1;
  if(rank == 0) {
    printf("\nCompleting Send/Recv from");
    printf(" %i to %i of row %i...\n\n",src,dest,row);
  }
  if(rank == 0){
    MPI_Send(&matrix[row][0],1,MY_MPI_ROW,
             dest,tag1,MPI_COMM_WORLD);
  }
  if(rank == 1){
    MPI_Recv(&vector[0],NCOLS,MPI_DOUBLE,
             src,tag1,MPI_COMM_WORLD,&Stat);
  }

  // Print matrix after Send/Recv
  // Print vector after Send/Recv
```

Output – 2 Processes

```
(Rank 0): Matrix:
   0   1   2   3
   4   5   6   7
   8   9  10  11
(Rank 1): Matrix:
   0   0   0   0
   0   0   0   0
   0   0   0   0
(Rank 0): Row Vector:
   0   0   0   0
(Rank 1): Row Vector:
   0   0   0   0

Completing Send/Recv from 0 to 1 of
      row 1...

(Rank 0): Matrix:
   0   1   2   3
   4   5   6   7
   8   9  10  11
(Rank 1): Matrix:
   0   0   0   0
   0   0   0   0
   0   0   0   0
(Rank 0): Row Vector:
   0   0   0   0
(Rank 1): Row Vector:
   4   5   6   7
```

C – Send One Column from 2D Array (Incorrect)

```
// Incorrect attempt at sending one column of a 2d array
#define NROWS 3
#define NCOLS 4
  double matrix[NROWS][NCOLS] = { 0 };

  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);

  // Initialize matrix on process 0 only
  // Print matrix before Send/Recv
  // Send/Recv
  tag1 = 1; dest=1; src=0; col=1;
  if(rank == 0) {
    printf("\nCompleting Send/Recv from");
    printf(" %i to %i of col %i...\n\n",src,dest,col);
  }
  if(rank == 0){
    MPI_Send(&matrix[0][col],NROWS,MPI_DOUBLE,
             dest,tag1,MPI_COMM_WORLD);
  }
  if(rank == 1){
    MPI_Recv(&matrix[0][col],NROWS,MPI_DOUBLE,
             src,tag1,MPI_COMM_WORLD,&Stat);
  }

  // Print matrix after Send/Recv
  ...
```

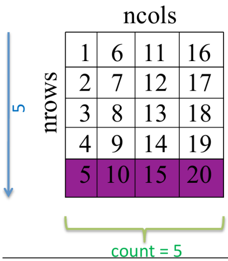
Output – 2 Processes

```
(Rank 0): Matrix:
   0    1    2    3
   4    5    6    7
   8    9   10   11
(Rank 1): Matrix:
   0    0    0    0
   0    0    0    0
   0    0    0    0

Completing Send/Recv from 0 to 1 of
      col 1...

(Rank 0): Matrix:
   0    1    2    3
   4    5    6    7
   8    9   10   11
(Rank 1): Matrix:
   0    1    2    3
   0    0    0    0
   0    0    0    0
```

Vector Types

- `MPI_Type_vector`: create a type for non-contiguous vectors with constant stride

```
MPI_Type_vector(count,blklen,stride, oldtype,newtype, ierr)
```



1	6	11	16
2	7	12	17
3	8	13	18
4	9	14	19
5	10	15	20

```
integer row_type
...
call MPI_Type_vector(ncols, 1, nrows,
          MPI_REAL8, row_type, ierr)
call MPI_Type_commit(row_type, ierr)
```

Vector Type Example #1

C – Send One Column from 2D Array (MPI_Type_vector)

```
// Sending one column of 2D array via MPI_Type_vector
#define NROWS 3
#define NCOLS 4
  double matrix[NROWS][NCOLS] = { 0 };

  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);

  // Initialize matrix on process 0 only
  // Print matrix before Send/Recv
  // Create derived noncontiguous vector type
  MPI_Datatype MY_MPI_COLUMN;
  MPI_Type_vector(NROWS,1,NCOLS,MPI_DOUBLE,&MY_MPI_COLUMN);
  MPI_Type_commit(&MY_MPI_COLUMN);
  // Send/Recv
  tag1 = 1; dest=1; src=0; col=2;
  if(rank == 0) {
    printf("\nCompleting Send/Recv from");
    printf(" %i to %i of col %i...\n\n",src,dest,col);
  }
  if(rank == 0){
    MPI_Send(&matrix[0][col],1,MY_MPI_COLUMN,
             dest,tag1,MPI_COMM_WORLD);
  }
  if(rank == 1){
    MPI_Recv(&matrix[0][col],1,MY_MPI_COLUMN,
             src,tag1,MPI_COMM_WORLD,&Stat);
  }
  / P      rix after Send/Recv
  ...
```

Output – 2 Processes

```
(Rank 0): Matrix:
   0   1   2   3
   4   5   6   7
   8   9  10  11
(Rank 1): Matrix:
   0   0   0   0
   0   0   0   0
   0   0   0   0

Completing Send/Recv from 0 to 1 of
      col 2...

(Rank 0): Matrix:
   0   1   2   3
   4   5   6   7
   8   9  10  11
(Rank 1): Matrix:
   0   0   2   0
   0   0   6   0
   0   0  10   0
```

Vector Type Example #2

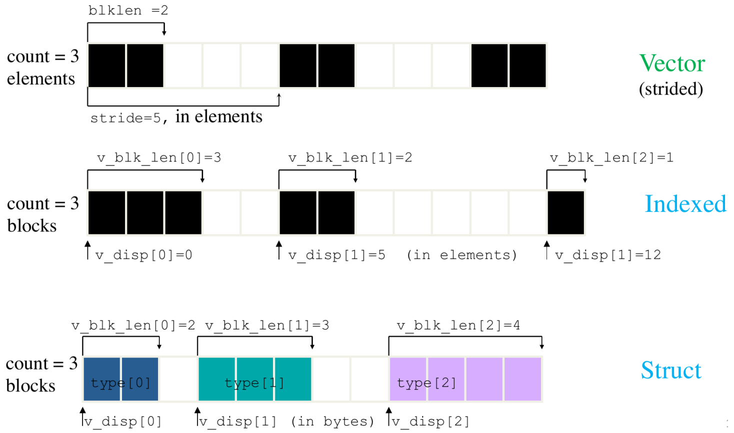
C – Send One Column from 2D Array to 1D Array

```c
// Sending one column of 2D array to 1D array
#define NROWS 3
#define NCOLS 4
  double matrix[NROWS][NCOLS] = { 0 };
  double vector[NCOLS] = { 0 };
  // Initialize matrix on process 0 only
  // Print matrix before Send/Recv
  // Print vector before Send/Recv
  // Send/Recv
  tag1 = 1; dest=1; src=0; col=2;
  if(rank == 0) {
    printf("\nCompleting Send/Recv from");
    printf(" %i to %i of col %i...\n\n",src,dest,col);
  }
  MPI_Datatype MY_MPI_COLUMN;
  MPI_Type_vector(NROWS,1,NCOLS,MPI_DOUBLE,&MY_MPI_COLUMN);
  MPI_Type_commit(&MY_MPI_COLUMN);
  if(rank == 0){
    MPI_Send(&matrix[0][col],1,MY_MPI_COLUMN,
             dest,tag1,MPI_COMM_WORLD);
  }
  if(rank == 1){
    MPI_Recv(&vector[0],NROWS,MPI_DOUBLE,
             src,tag1,MPI_COMM_WORLD,&Stat);
  }

  // Print matrix after Send/Recv
  // Print vector after Send/Recv
```

Output – 2 Processes

```
(Rank 0): Matrix:
   0   1   2   3
   4   5   6   7
   8   9  10  11
(Rank 1): Matrix:
   0   0   0   0
   0   0   0   0
   0   0   0   0
(Rank 0): Col Vector:
   0...
(Rank 1): Col Vector:
   0...

Completing Send/Recv from 0 to 1 of
      col 2...

(Rank 0): Matrix:
   0   1   2   3
   4   5   6   7
   8   9  10  11
(Rank 1): Matrix:
   0...
(Rank 0): Col Vector:
   0
   0
   0
(Rank 1): Col Vector:
   2
   6
  10
```

Derived types (arguments)



Fortran Gotcha!

Fortran – Don't Use Subarray Notation!

```
! Incorrect attempt at sending one row of a 2D array
! Temporary contiguous copy goes out of scope with
      non-blocking calls!
  integer, parameter :: NROWS = 3
  integer, parameter :: NCOLS = 4
  double precision, dimension(NROWS,NCOLS) :: matrix
  integer reqs(2)
  ! Initialize matrix on process 0 only
  ! Print matrix before Send/Recv
  ! Send/Recv
  tag1 = 1; dest = 1; src = 0; row = 2
  if (rank == 0) then ! NO NO NO
   call
         MPI_ISEND(matrix(row,:),NCOLS,MPI_DOUBLE_PRECISION,
         &
                  dest,tag1,MPI_COMM_WORLD,reqs(1),ierr)
   call MPI_WAIT(reqs(1),stats(1,1),ierr)
  end if
  if (rank == 1) then ! NO NO NO
   call
         MPI_IRECV(matrix(row,:),NCOLS,MPI_DOUBLE_PRECISION,
         &
                  src,tag1,MPI_COMM_WORLD,reqs(2),ierr)
   call MPI_WAIT(reqs(2),stats(1,2),ierr)
  end if
  ! Print matrix after Send/Recv
  ...
```

Output – 2 Processes

```
(Rank 0): Matrix:
  0.  1.  2.  3.
  4.  5.  6.  7.
  8.  9. 10. 11.
(Rank 1): Matrix:
  0.  0.  0.  0.
  0.  0.  0.  0.
  0.  0.  0.  0.

Completing Send/Recv from 0 to 1 from
      row 2...

(Rank 0): Matrix:
  0.  1.  2.  3.
  4.  5.  6.  7.
  8.  9. 10. 11.
(Rank 1): Matrix:
  0.  0.  0.  0.
  0.  0.  0.  0.
  0.  0.  0.  0.
Program received signal SIGSEGV:
      Segmentation fault - invalid
      memory reference.
```

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”.

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"

Communicators and libraries

- Sharing communicator between main and library:
- Library can receive messages from the main program. Oops.

```
main (){
  if (me==0) MPI_Send( ..to 1.. ,
                 MPI_COMM_WORLD )
  library_call()
  if (me==1) MPI_Recv( .. from 0 .. ,
                 MPI_COMM_WORLD )
}

void library_call() {
  other = me-1;
  MPI_Recv( .. from other .. ,
                 MPI_COMM_WORLD )
}
```

Duplicate communicators

- Duplicate communicator with MPI_Comm_dup:
- Same group of processors, but different context: no confusion possible.

```
main (){
  if (me==0) MPI_Send( ..to 1..,
                MPI_COMM_WORLD )
  library_call(MPI_COMM_WORLD)
  if (me==1) MPI_Recv( .. from 0 .. ,
                MPI_COMM_WORLD )
}

void library_call(comm) {
  MPI_Comm my_comm = // copy of comm
  other = me-1;
  MPI_Recv( .. from other ..,
            my_comm )
}
```

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`.

Communicators

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

Communicator Example #1

C Version

```c
#define NPROCS 8
int main(int argc, char *argv[])  {
  int rank, new_rank, sendbuf, recvbuf, numprocs;
  int ranks1[4]={0,1,2,3}, ranks2[4]={4,5,6,7};
  MPI_Group orig_group, new_group;
  MPI_Comm new_comm;
  sendbuf = rank;
  // Extract the original group handle
  MPI_Comm_group(MPI_COMM_WORLD, &orig_group);

  // Divide tasks into two distinct groups based upon
        rank
  if (rank < NPROCS/2) {
    MPI_Group_incl(orig_group, NPROCS/2, ranks1,
          &new_group);
    }
  else {
    MPI_Group_incl(orig_group, NPROCS/2, ranks2,
          &new_group);
    }

  // Create new new communicator and then perform
        collective communications
  MPI_Comm_create(MPI_COMM_WORLD, new_group, &new_comm);
  MPI_Allreduce(&sendbuf, &recvbuf, 1, MPI_INT, MPI_SUM,
        new_comm);

  MPI_Group_rank (new_group, &new_rank);
  printf("rank= %d newrank= %d recvbuf=
        %d\n",rank,new_rank,recvbuf);
```

Output – 8 Processes

```
rank= 0 newrank= 0 recvbuf= 6
rank= 1 newrank= 1 recvbuf= 6
rank= 2 newrank= 2 recvbuf= 6
rank= 3 newrank= 3 recvbuf= 6
rank= 4 newrank= 0 recvbuf= 22
rank= 5 newrank= 1 recvbuf= 22
rank= 6 newrank= 2 recvbuf= 22
rank= 7 newrank= 3 recvbuf= 22
```

Communicator Example #1

Fortran Version

```fortran
! Divide MPI_COMM_WORLD into two distinct Comms using
! COMM_GROUP, GROUP_INCL, COMM_CREATE
  integer, parameter :: NPROCS = 8
  integer rank, new_rank, sendbuf, recvbuf, numprocs
  integer ranks1(4), ranks2(4), ierr
  integer orig_group, new_group, new_comm
  data ranks1 /0, 1, 2, 3/, ranks2 /4, 5, 6, 7/
  sendbuf = rank
  ! Extract the original group handle
  call MPI_COMM_GROUP(MPI_COMM_WORLD, orig_group, ierr)
  ! Divide tasks into two distinct groups based upon rank
  if (rank .lt. NPROCS/2) then
    call MPI_GROUP_INCL(orig_group,NPROCS/2, &
                        ranks1,new_group,ierr)
  else
    call MPI_GROUP_INCL(orig_group,NPROCS/2, &
                        ranks2,new_group,ierr)
  endif
  call MPI_COMM_CREATE(MPI_COMM_WORLD,new_group, &
                       new_comm,ierr)
  call MPI_ALLREDUCE(sendbuf,recvbuf,1,MPI_INTEGER, &
                     MPI_SUM, new_comm, ierr)
  call MPI_GROUP_RANK(new_group, new_rank, ierr)
  write(*,*)"(Rank ",rank,"): newrank: ",new_rank,"
      recvbuf: ",recvbuf
```

Output – 8 Processes

```
(Rank 0 ): newrank: 0  recvbuf:  6
(Rank 1 ): newrank: 1  recvbuf:  6
(Rank 2 ): newrank: 2  recvbuf:  6
(Rank 3 ): newrank: 3  recvbuf:  6
(Rank 4 ): newrank: 0  recvbuf: 22
(Rank 5 ): newrank: 1  recvbuf: 22
(Rank 6 ): newrank: 2  recvbuf: 22
(Rank 7 ): newrank: 3  recvbuf: 22
```

MPI_Comm_split

- Provides a short cut method to create a <u>collection of communicators</u>
- 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)
```

Communicator Example #2

C Version

```
#define NPROCS 8
int main(int argc, char *argv[])  {
  int rank, new_rank, sendbuf, recvbuf, numprocs, color;
  MPI_Comm new_comm;

  sendbuf = rank;

  // Divide tasks into two distinct colors based upon
        rank
  if (rank < NPROCS/2) { color = 1; }
  else { color = 2; }

  // Create new new communicator and then perform
        collective communications
  printf("(Rank %i): color: %i\n",rank,color);
  MPI_Comm_split(MPI_COMM_WORLD, color, rank, &new_comm);
  MPI_Allreduce(&sendbuf, &recvbuf, 1, MPI_INT, MPI_SUM,
        new_comm);

  MPI_Comm_rank(new_comm, &new_rank);
  printf("(Rank %i): newrank: %i recvbuf:
        %i\n",rank,new_rank,recvbuf);
```

Output – 8 Processes

```
(Rank 1): color: 1
(Rank 2): color: 1
(Rank 3): color: 1
(Rank 4): color: 2
(Rank 5): color: 2
(Rank 6): color: 2
(Rank 7): color: 2
(Rank 0): color: 1
(Rank 0): newrank: 0 recvbuf: 6
(Rank 1): newrank: 1 recvbuf: 6
(Rank 2): newrank: 2 recvbuf: 6
(Rank 3): newrank: 3 recvbuf: 6
(Rank 4): newrank: 0 recvbuf: 22
(Rank 5): newrank: 1 recvbuf: 22
(Rank 6): newrank: 2 recvbuf: 22
(Rank 7): newrank: 3 recvbuf: 22
```

Communicator Example #2

Fortran Version

```
! Divide MPI_COMM_WORLD into two distinct Comms using
! COMM_GROUP, GROUP_INCL, COMM_CREATE
  integer, parameter :: NPROCS = 8
  integer rank, new_rank, sendbuf, recvbuf, numprocs
  integer color, new_comm, ierr

  sendbuf = rank

  ! Divide tasks into two distinct groups based upon rank
  if (rank .lt. NPROCS/2) then
    color = 1
  else
    color = 2
  endif

  write(*,*)"(Rank ",rank,"): color: ",color
  call MPI_COMM_SPLIT(MPI_COMM_WORLD, color, rank,
       new_comm, ierr)
  call MPI_ALLREDUCE(sendbuf,recvbuf,1,MPI_INTEGER, &
                    MPI_SUM, new_comm, ierr)

  call MPI_COMM_RANK(new_comm, new_rank, ierr)
  write(*,*)"(Rank ",rank,"): newrank: ",new_rank,"
       recvbuf: ",recvbuf
```

Output – 8 Processes

```
(Rank 1): color:   1
(Rank 2): color:   1
(Rank 3): color:   1
(Rank 5): color:   2
(Rank 6): color:   2
(Rank 7): color:   2
(Rank 0): color:   1
(Rank 4): color:   2
(Rank 0): newrank:   0  recvbuf:   6
(Rank 1): newrank:   1  recvbuf:   6
(Rank 2): newrank:   2  recvbuf:   6
(Rank 3): newrank:   3  recvbuf:   6
(Rank 4): newrank:   0  recvbuf:  22
(Rank 5): newrank:   1  recvbuf:  22
(Rank 7): newrank:   3  recvbuf:  22
(Rank 6): newrank:   2  recvbuf:  22
```

Topologies

- Use the MPI library for common grid topologies (local functions)
- A *topology* maps process-ranks onto a set of N-tuples.
- E.g. {0, 1, 2, 3}->{(0,0), (0,1), (1,0), (1,1)} (row-major in ranks)
- Cartesian Maps (arbitrary number of dimensions):

MPI_Cart_create	Creates map (ranks $\rightarrow$ 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 Nth neighbor's coords.

- **graph** constructors go beyond the *N*-dimensional rectilinear mapping of the Cartesian topology (MPI_Graph_create)

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

(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 ***programmed*** **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.
- Can be used within an intra-communicator; cannot be added to inter-communicators.

References

- Victor Eijkhout, “Introduction to High Performance Scientific Computing”
- Victor Eijkhout, “Parallel Computing for Science and Engineering”
- Mark Lubin, “Introduction into new features of MPI-3.0 Standard”
- “MPI: A Message-Passing Interface Standard Version-3.0”