MPI reference

First five MPI commands

C & Fortran bindings

```
int MPI_Init(int *argc, char **argv)
int MPI_Comm_size(MPI_Comm comm, int *size)
int MPI_Comm_rank(MPI_Comm comm, int *rank)
int MPI_Barrier(MPI_Comm comm)
MPI_Finalize()

MPI_INIT(ierror)
MPI_COMM_SIZE(comm, size, ierror)
MPI_COMM_RANK(comm, rank, ierror)
MPI_BARRIER(comm, ierror)
MPI_FINALIZE(ierror)
integer comm, size, rank, ierror
```

Send operation

- C/C++ binding

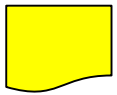
```
int MPI_Send(void *buffer, int count, MPI_Datatype datatype,int
   dest,  int tag, MPI_Comm comm)
```

  - The return value of the function is the error value

- Fortran binding

```
MPI_SEND(buffer, count, datatype,
   dest,tag, comm, ierror)
<type>, dimension(*) :: buf
integer :: count, datatype, dest, tag, comm, ierror
```

  - **ierror:** the error value



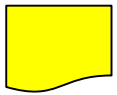
Receive operation

- C/C++ binding

```
int MPI_Recv( void *buf, int count, MPI_Datatype datatype, int
   source, int tag, MPI_Comm comm, MPI_Status *status )
```

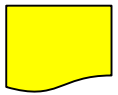
- Fortran binding

```
mpi_recv(buf, count, datatype, source, tag, comm, status, ierror)
<type>, dimension(*) :: buf
integer :: count, datatype, source, tag, comm, ierror
integer, dimension(MPI_STATUS_SIZE) :: status
```



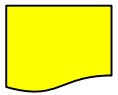
MPI datatypes

MPI type	C type
MPI_CHAR	signed char
MPI_SHORT	short int
MPI_INT	**int**
MPI_LONG	long int
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED_INT	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	**float**
MPI_DOUBLE	**double**
MPI_LONG_DOUBLE	long double
MPI_BYTE	



MPI datatypes

MPI type	Fortran type
MPI_CHARACTER	CHARACTER
MPI_INTEGER	**INTEGER**
MPI_REAL	**REAL**
MPI_REAL8	REAL*8 (nonstandard)
MPI_DOUBLE_PRECISION	**DOUBLE PRECISION**
MPI_COMPLEX	COMPLEX
MPI_DOUBLE_COMPLEX	DOUBLE COMPLEX
MPI_LOGICAL	LOGICAL
MPI_BYTE	



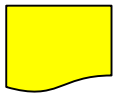
Combined send & receive

- C/C++ binding

```
int MPI_Sendrecv( void *sendbuf, int sendcount, MPI_Datatype
   sendtype, int dest, int sendtag, void *recvbuf, int recvcount,
   MPI_Datatype recvtype, int source, int recvtag, MPI_Comm comm,
   MPI_Status *status )
```

- Fortran binding

```
mpi_sendrecv(sendbuf, sendcount, sendtype, dest, sendtag, recvbuf,
   recvcount, recvtype, source, recvtag, comm, status, ierror)
<type>, dimension(*) :: sendbuf, recvbuf
integer :: sendcount, sendtype, dest, sendtag, recvcount, recvtype,
   source, recvtag, comm, ierror
integer, dimension(MPI_STATUS_SIZE) :: status
```



MPI_Bcast

C & Fortran bindings

```
int MPI_Bcast(void* buffer, int count, MPI_datatype datatype,
              int root, MPI_Comm comm)


MPI_BCAST(buffer, count, datatype, root, comm, ierror)
type buffer(*)
integer count, datatype, root, comm, ierror
```

MPI_Scatter

C & Fortran bindings

```
int MPI_Scatter(void* sendbuf, int sendcount, MPI_datatype
                sendtype, void* recvbuf, int recvcount,
                MPI_datatype recvtype, int root, MPI_Comm comm)


MPI_SCATTER(sendbuf, sendcount, sendtype, recvbuf, recvcount,
            recvtype, root, comm, ierror)

type sendbuf(*), recvbuf(*)
integer sendcount, recvcount, sendtype, recvtype, root, comm,
        ierror
```

MPI_Scatterv

C & Fortran bindings

```
int MPI_Scatterv(void* sendbuf, int *sendcounts, int *displs,
                 MPI_datatype sendtype, void* recvbuf,
                 int recvcount, MPI_datatype recvtype,
                 int root, MPI_Comm comm)


MPI_SCATTERV(sendbuf, sendcounts, displs, sendtype, recvbuf,
             recvcount, recvtype, root, comm, ierror)
type sendbuf(*), recvbuf(*)
integer sendcounts(*), displs(*), recvcount, sendtype,
        recvtype, root, comm, ierror
```

Reduce operation

- Available reduction operations (argument op)

Operation	Meaning
MPI_MAX	Max value
MPI_MIN	Min value
MPI_SUM	Sum
MPI_PROD	Product
MPI_MAXLOC	Max value + location
MPI_MINLOC	Min value + location
MPI_LAND	Logical AND
MPI_BAND	Bytewise AND
MPI_LOR	Logical OR
MPI_BOR	Bytewise OR
MPI_LXOR	Logical XOR
MPI_BXOR	Bytewise XOR

MPI_Gather

C and Fortran bindings

```
int MPI_Gather(void* sendbuf, int sendcount,
               MPI_datatype sendtype, void* recvbuf,
               int recvcount, MPI_datatype recvtype,
               int root, MPI_Comm comm)


MPI_GATHER(sendbuf, sendcount, sendtype, recvbuf, recvcount,
           recvtype, root, comm, ierror)
type     sendbuf(*), recvbuf(*)
integer sendcount, recvcount, sendtype, recvtype, root, comm,
        ierror
```

MPI_Gatherv

C and Fortran bindings

```
int MPI_Gatherv ( void *sendbuf, int sendcnt,
                  MPI_Datatype sendtype, void *recvbuf,
                  int *recvcnts, int *displs,
                  MPI_Datatype recvtype, int root,
                  MPI_Comm comm )

MPI_GATHERV(sendbuf, sendcount, sendtype, recvbuf, recvcounts,
            displs, recvtype, root, comm, ierror)

type    sendbuf(*), recvbuf(*)
integer sendcount, recvcounts(*), displs(*), sendtype,
        recvtype, root, comm, ierror
```

Reduce operation

C and Fortran bindings

```
int MPI_Reduce(void* sendbuf, void* recvbuf, int count,
               MPI_Datatype datatype, MPI_Op op,
               int root, MPI_Comm comm)


MPI_REDUCE(sendbuf, recvbuf, count, datatype, op, root,
           comm, ierror)

type sendbuf(*), recvbuf(*)
integer count, datatype, op, root, comm, ierror
```

MPI_Allreduce

C & Fortran bindings

```
int MPI_Allreduce(void* sendbuf, void* recvbuf, int count,
                  MPI_Datatype datatype, MPI_Op op,
                  MPI_Comm comm)
```

```
MPI_ALLREDUCE(sendbuf, recvbuf, count, datatype, op, comm,
              ierror)

type :: sendbuf(*), recvbuf(*)
integer :: count, datatype, op, comm, ierror
```

MPI_Allgather

C & Fortran bindings

```
int MPI_Allgather(void* sendbuf, int sendcount,
                  MPI_datatype sendtype, void* recvbuf,
                  int recvcount, MPI_datatype recvtype,
                  MPI_Comm comm)
```

```
MPI_ALLGATHER(sendbuf, sendcount, sendtype, recvbuf, recvcount,
              recvtype, comm, ierror)

type :: sendbuf(*), recvbuf(*)
integer :: sendcount,  recvcount, sendtype, recvtype, comm, ierror
```

MPI_Reduce_scatter

C & Fortran bindings

```
int MPI_Reduce_scatter(void* sendbuf, void* recvbuf,
                       int* recvcounts, MPI_Datatype datatype,
                       MPI_Op op, MPI_Comm comm)
```

```
MPI_REDUCE_SCATTER(sendbuf, recvbuf, recvcounts, datatype,
                   op, comm, ierror)

type :: sendbuf(*), recvbuf(*)
integer :: recvcounts(*), datatype, op, comm, ierror
```

MPI_Alltoall

C & Fortran bindings

```
int MPI_Alltoall(void* sendbuf, int sendcount,
                 MPI_datatype sendtype, void* recvbuf,
                 int recvcount, MPI_datatype recvtype,
                 MPI_Comm comm)


MPI_ALLTOALL(sendbuf, sendcount, sendtype, recvbuf,
             recvcount,  recvtype, comm, ierror)

type :: sendbuf(*), recvbuf(*)
integer :: sendcount, recvcount, sendtype, recvtype, comm, ierror
```

MPI_Alltoallv

C & Fortran bindings

```
int MPI_Alltoallv(void* sendbuf, int *sendcounts, int *sdispls,
                  MPI_Datatype sendtype, void* recvbuf,
                  int *recvcounts, int *rdispls,
                  MPI_Datatype recvtype,MPI_Comm comm)


MPI_ALLTOALLV(sendbuf,sendcounts, sdispls, sendtype, recvbuf,
              recvcounts, rdispls, recvtype, comm, ierror)

type :: sendbuf(*), recvbuf(*)
integer :: sendcounts(*), recvcounts(*), sdispls(*), rdispls(*),
        sendtype, recvtype, comm, ierror
```

Creating a communicator

- C and Fortran bindings

```
int MPI_Comm_split (MPI_Comm comm, int color, int key,
                    MPI_Comm newcomm)

MPI_COMM_SPLIT (comm, color, key, newcomm, rc)
integer :: comm, color, key, newcomm, rc
```

- Return code values

MPI_SUCCESS	No error; MPI routine completed successfully.
MPI_ERR_COMM	Invalid communicator. A common error is to use a null communicator in a call
MPI_ERR_INTERN	This error is returned when some part of the implementation is unable to acquire memory.

Communicator manipulation

- C and Fortran bindings

```
int MPI_Comm_compare ( MPI_Comm comm1, MPI_Comm comm2,
                       int result )

MPI_COMM_COMPARE ( comm1, comm2, result, rc )
integer :: comm1, comm2, result, rc

int MPI_Comm_dup ( MPI_Comm comm, MPI_Comm newcomm )

MPI_COMM_DUP ( comm, newcomm, rc )
integer :: comm, newcomm, rc

int MPI_Comm_free ( MPI_Comm comm )

MPI_COMM_FREE ( comm, rc )
integer :: comm, rc
```

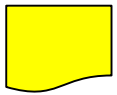
Non-blocking send

- C/C++ binding

```
int MPI_Isend( void *buf, int count, MPI_Datatype datatype, int
   dest, int tag, MPI_Comm comm, MPI_Request *request )
```

- Fortran binding

```
MPI_ISEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, REQUEST,IERROR)

<type> :: BUF(*)
INTEGER :: COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR
```



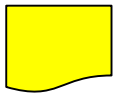
Non-blocking receive

- C/C++ binding

```
int MPI_Irecv( void *buf, int count, MPI_Datatype datatype, int
   source, int tag, MPI_Comm comm, MPI_Request *request )
```

- Fortran binding

```
MPI_IRECV(BUF, COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST,IERROR)
<type> :: BUF(*)
INTEGER :: COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERROR
```



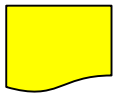
Wait for non-blocking operation

- C/C++ binding

```
int MPI_Wait(MPI_Request *request, MPI_Status *status)
```

- Fortran binding

```
MPI_WAIT(REQUEST, STATUS, IERROR)
INTEGER :: REQUEST, STATUS(MPI_STATUS_SIZE), IERROR
```



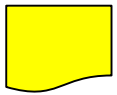
Wait for non-blocking operations

- C/C++ binding

```
int MPI_Waitall(int count, MPI_Request *array_of_requests,MPI_Status
   *array_of_statuses)
```

- Fortran binding

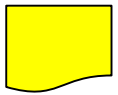
```
MPI_WAITALL(COUNT, ARRAY_OF_REQUESTS, ARRAY_OF_STATUSES, IERROR)
INTEGER :: COUNT, ARRAY_OF_REQUESTS(:),
   ARRAY_OF_STATUSES(MPI_STATUS_SIZE,:), IERROR
```



Creating a communication topology

- C and Fortran bindings

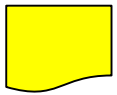
```
int MPI_Cart_create(MPI_Comm old_comm, int ndims, int *dims, int
   *periods, int reorder, MPI_Comm *comm_cart)

MPI_CART_CREATE(old_comm, ndims, dims, periods,
     reorder, comm_cart, rc)
   integer :: old_comm, ndims, dims(:), comm_cart, rc
   logical :: reorder, periods(:)
```



Ranks and coordinates

- C and Fortran bindings

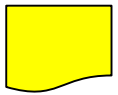
```
int MPI_Cart_coords(MPI_Comm comm, int rank, int maxdim, int *coords)

MPI_CART_COORDS(comm, rank, maxdim, coords, rc)
   integer :: comm, rank, maxdim, coords(:), rc

int MPI_Cart_rank(MPI_Comm comm, int *coords, int rank)

MPI_CART_RANK(comm, coords, rank, rc)
   integer :: comm, coords(:), rank, rc
```



Communication in a topology

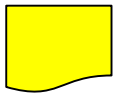
- C and Fortran bindings

```
int MPI_Cart_shift( MPI_Comm comm, int direction, int displ, int
   *source, int *dest )

MPI_CART_SHIFT(comm, direction, displ, source, dest, rc)
   integer :: comm, direction, displ, source, dest, rc
```



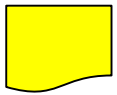
C interfaces for datatype routines

```
int MPI_Type_commit(MPI_Datatype *type)

int MPI_Type_free(MPI_Datatype *type)

int MPI_Type_contiguous(int count, MPI_Datatype oldtype,
  MPI_Datatype *newtype)

int MPI_Type_vector(int count, int block, int stride,
  MPI_Datatype oldtype, MPI_Datatype *newtype)

int MPI_Type_indexed(int count, int blocks[], int displs[], MPI_Datatype
  oldtype, MPI_Datatype *newtype)

int MPI_Type_create_subarray(int ndims, int array_of_sizes[], int
  array_of_subsizes[], int array_of_starts[], int order, MPI_Datatype
  oldtype, MPI_Datatype *newtype )
```



Fortran interfaces for datatype routines

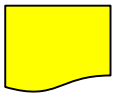
```
mpi_type_commit(type, rc)
  integer :: type, rc
mpi_type_free(type, rc)
  integer :: type, rc
mpi_type_contiguous(count, oldtype, newtype, rc)
  integer :: count, oldtype, newtype, rc
mpi_type_vector(count, block, stride, oldtype, newtype, rc)
  integer :: count, block, stride, oldtype, newtype, rc
mpi_type_indexed(count, blocks, displs, oldtype, newtype, rc)
  integer :: count, oldtype, newtype, rc
  integer, dimension(count) :: blocks, displs
mpi_type_create_subarray(ndims, sizes, subsizes, starts, order,
  oldtype, newtype, rc)
  integer :: ndims, order, oldtype, newtype, rc
  integer, dimension(ndims) :: sizes, subsizes, starts
```



ONE-SIDED COMMUNICATION

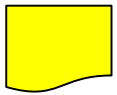
C interfaces for one-sided routines

```
int MPI_Win_create(void *base, MPI_Aint size, int disp_unit, MPI_Info
  info, MPI_Comm comm, MPI_Win *win)

int MPI_Win_fence(int assert, MPI_Win win)

int MPI_Put(const void *origin_addr, int origin_count, MPI_Datatype
  origin_datatype, int target_rank, MPI_Aint target_disp, int
  target_count, MPI_Datatype target_datatype, MPI_Win win)

int MPI_Get(void *origin_addr, int origin_count, MPI_Datatype
  origin_datatype, int target_rank, MPI_Aint target_disp, int
  target_count, MPI_Datatype target_datatype, MPI_Win win)

int MPI_Accumulate(const void *origin_addr, int origin_count,
  MPI_Datatype origin_datatype, int target_rank, MPI_Aint  target_disp,
  int target_count, MPI_Datatype target_datatype, MPI_Op op, MPI_Win
  win)
```



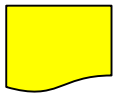
Fortran interfaces for one-sided routines

```
mpi_win_create(base, size, disp_unit, info, comm, win, rc)
  <type> :: base(*)
  integer(kind=mpi_address_kind) :: size
  integer :: disp_unit, info, comm, win, rc
mpi_win_fence(assert, win, rc)
  integer :: assert, win, rc
mpi_put(origin_addr, origin_count, origin_datatype, target_rank,
  target_disp, target_count, target_datatype, win, rc)
  <type> :: origin_addr(*)
  integer(kind=mpi_address_kind) :: target_disp
  integer :: origin_count, origin_datatype, target_rank, target_count,
  target_datatype, win, rc
```



Fortran interfaces for one-sided routines

```
mpi_get(origin_addr, origin_count, origin_datatype, target_rank,
  target_disp, target_count, target_datatype, win, rc)
  <type> :: origin_addr(*)
  integer(kind=mpi_address_kind) :: target_disp
  integer :: origin_count, origin_datatype, target_rank, target_count,
  target_datatype, win, rc
mpi_accumulate(origin_addr, origin_count, origin_datatype, target_rank,
  target_disp, target_count, target_datatype, op, win, rc)
  <type> :: origin_addr(*)
  integer(kind=mpi_address_kind) :: target_disp
  integer :: origin_count, origin_datatype, target_rank, target_count,
  target_datatype, op, win, rc
```



MPI I/O

C interfaces to MPI I/O routines

```
int MPI_File_open(MPI_Comm comm, char *filename, int amode, MPI_Info
   info, MPI_File *fh)

int MPI_File_close(MPI_File *fh)

int MPI_File_seek(MPI_File fh, MPI_Offset offset, int whence)

int MPI_File_read(MPI_File fh, void *buf, int count,
   MPI_Datatype datatype, MPI_Status *status)

int MPI_File_read_at(MPI_File fh, MPI_Offset offset, void *buf, int
   count, MPI_Datatype datatype, MPI_Status *status)

int MPI_File_write(MPI_File fh, void *buf, int count, MPI_Datatype
   datatype, MPI_Status *status)

int MPI_File_write_at(MPI_File fh, MPI_Offset offset, void *buf, int
   count, MPI_Datatype datatype, MPI_Status *status)
```

C interfaces to MPI I/O routines

```
int MPI_File_set_view(MPI_File fh, MPI_Offset disp, MPI_Datatype etype,
   MPI_Datatype filetype, char *datarep, MPI_Info info)
int MPI_File_read_all(MPI_File fh, void *buf, int count,
   MPI_Datatype datatype, MPI_Status *status)
int MPI_File_read_at_all(MPI_File fh, MPI_Offset offset, void *buf, int
   count, MPI_Datatype datatype, MPI_Status *status)
int MPI_File_write_all(MPI_File fh, void *buf, int count, MPI_Datatype
   datatype, MPI_Status *status)
int MPI_File_write_at_all(MPI_File fh, MPI_Offset offset, void *buf, int
   count, MPI_Datatype datatype, MPI_Status *status)
```

Fortran interfaces for MPI I/O routines

```
mpi_file_open(comm, filename, amode, info, fh, ierr)
   integer       ::  comm, amode, info, fh, ierr
   character* ::  filename
mpi_file_close(fh, ierr)
mpi_file_seek(fh, offset, whence)
integer :: fh, offset, whence
mpi_file_read(fh, buf, count, datatype, status)
   integer :: fh, buf, count, datatype, status(mpi_status_size)
mpi_file_read_at(fh, offset, buf, count, datatype, status)
   integer :: fh, offset, buf, count, datatype
   integer, dimension(mpi_status_size) :: status
mpi_file_write(fh, buf, count, datatype, status)
mpi_file_write_at(fh, offset, buf, count, datatype, status)
```

Fortran interfaces for MPI I/O routines

```
mpi_file_set_view(fh, disp, etype, filetype, datarep, info)
   integer :: fh, disp, etype, filetype, info
   character* :: datarep
mpi_file_read_all(fh, buf, count, datatype, status)
mpi_file_read_at_all(fh, offset, buf, count, datatype, status)
mpi_file_write_all(fh, buf, count, datatype, status)
mpi_file_write_at_all(fh, offset, buf, count, datatype, status)
```