MPI Quick Reference

Environment

Initialize MPI:
```
int MPI_Init(int *argc, char ***argv)
```

Cleanup MPI:
```
int MPI_Finalize()
```

Determine wall clock time:
```
double MPI_Wtime()
```

Related Functions: *MPI_Get_processor_name*, *MPI_Wtick*, *MPI_Initialized*, *MPI_Abort*, *MPI_Pcontrol*

Blocking Point-to-Point

Send a message to one process:
```
int MPI_Send(void *buf, int count,
    MPI_Datatype datatype, int dest, int
    tag, MPI_Comm comm)
```

Receive a message from one process:
```
int MPI_Recv(void *buf, int count,
    MPI_Datatype datatype, int source, int
    tag, MPI_Comm comm, MPI_Status *status)
```

Count received data elements:
```
int MPI_Get_count(MPI_Status *status,
    MPI_Datatype datatype, int *count)
```

Combined send and receive:
```
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)
```

Combined send and receive (using the same buffer):
```
int MPI_Sendrecv_replace(void *buf, int
    count, MPI_Datatype datatype, int
    dest, int sendtag, int source, int
    recvtag, MPI_Comm comm, MPI_Status
    *status)
```

Related Functions: *MPI_Bsend*, *MPI_Ssend*, *MPI_Rsend*, *MPI_Buffer_attach*, *MPI_Buffer_detach*, *MPI_Get_elements*, *MPI_Probe*

Non-Blocking Point-to-Point

Begins a non-blocking send:
```
int MPI_Isend(void *buf, int count,
    MPI_Datatype dtype, int dest, int tag,
    MPI_Comm comm, MPI_Request *request)
```

Begin to receive a message:
```
int MPI_Irecv(void *buf, int count,
    MPI_Datatype dtype, int src, int tag,
    MPI_Comm comm, MPI_Request *request)
```

Complete a non-blocking operation:
```
int MPI_Wait(MPI_Request *request,
    MPI_Status *status)
```

Check or complete a non-blocking operation:
```
int MPI_Test(MPI_Request *request, int
    *flag, MPI_Status *status)
```

Related Functions: *MPI_Request_free*, *MPI_Ibsend*, *MPI_Issend*, *MPI_Irsend*, *MPI_Waitany*, *MPI_Waitall*, *MPI_Waitsome*, *MPI_Testany*, *MPI_Testall*, *MPI_Testsome*, *MPI_Cancel*, *MPI_Test_cancelled*

Collective

Synchronize all processes:
```
int MPI_Barrier(MPI_Comm comm)
```

Send one message to all processes:
```
int MPI_Bcast(void *buf, int count,
    MPI_Datatype datatype, int root,
    MPI_Comm comm)
```

Combine messages from all processes:
```
int MPI_Reduce(void *sendbuf, void
    *recvbuf, int count, MPI_Datatype
    datatype, MPI_Op op, int root, MPI_Comm
    comm)
```

Receive from all processes:
```
int MPI_Gather(void *sendbuf, int
    sendcount, MPI_Datatype sendtype, void
    *recvbuf, int recvcount, MPI_Datatype
    recvtype, int root, MPI_Comm comm)
```

Send separate messages to all processes:
```
int MPI_Scatter(void *sendbuf, int
    sendcount, MPI_Datatype sendtype, void
    *recvbuf, int recvcount, MPI_Datatype
    recvtype, int root, MPI_Comm comm)
```

Scatter a buffer in parts to all processes:
```
int MPI_Scatterv(void *sendbuf, int
    *sendcnts, int *displs, MPI_Datatype
    sendtype, void *recvbuf, int recvcnt,
    MPI_Datatype recvtype, int root,
    MPI_Comm comm)
```

Related Functions: *MPI_Allgather*, *MPI_Allgatherv*, *MPI_Alltoall*, *MPI_Scan*, *MPI_Alltoallv*, *MPI_Allreduce*, *MPI_Gatherv*, *MPI_Op_create*, *MPI_Op_free*

Derived Datatypes

Create a strided homogeneous vector:
```
int MPI_Type_vector(int count, int
    blocklength, int stride, MPI_Datatype
    oldtype, MPI_Datatype *newtype)
```

Create a struct datatype:
```
int MPI_Type_create_struct(int count,
    int blocklens[], MPI_Aint indices[],
    MPI_Datatype old_types[], MPI_Datatype
    *newtype)
```

Register and release a derived datatype:
```
int MPI_Type_commit(MPI_Datatype *datatype)
int MPI_Type_free(MPI_Datatype *datatype)
```

Get the address of a location in memory:
```
int MPI_Get_address(void *location,
    MPI_Aint *address)
```

Pack data into a message buffer:
```
int MPI_Pack(void *inbuf, int incount,
    MPI_Datatype datatype, void *outbuf,
    int outsize, int *position, MPI_Comm
    comm)
```

Unpack data from a message buffer:
```
int MPI_Unpack(void *inbuf, int insize,
    int *position, void *outbuf, int
    outcount, MPI_Datatype datatype,
    MPI_Comm comm)
```

Related Functions: *MPI_Pack_size*, *MPI_Type_contiguous*, *MPI_Type_hvector*, *MPI_Type_indexed*, *MPI_Type_hindexed*, *MPI_Type_get_extent*, *MPI_Type_size*

Communicators and Groups

Count group members in communicator:
`int` **`MPI_Comm_size`**`(MPI_Comm comm, int *size)`

Determine group rank of self:
`int` **`MPI_Comm_rank`**`(MPI_Comm comm, int *rank)`

Duplicate with new context:
`int` **`MPI_Comm_dup`**`(MPI_Comm comm, MPI_Comm *newcomm)`

Split into categorized sub-groups:
`int` **`MPI_Comm_split`**`(MPI_Comm comm, int color, int key, MPI_Comm *newcomm)`

Related Functions: *MPI_Comm_create*, *MPI_Comm_free*, *MPI_Comm_group*, *MPI_Group_incl*, *MPI_Group_excl*, *MPI_Group_union*, *MPI_Group_intersection*

Topologies

Create communicator with cartesian topology:
`int` **`MPI_Cart_create`**`(MPI_Comm comm_old, int ndims, int *dims, int *periods, int reorder, MPI_Comm *comm_cart)`

Determine rank from cartesian coordinates:
`int` **`MPI_Cart_rank`**`(MPI_Comm comm, int *coords, int *rank)`

Determine cartesian coordinates from rank:
`int` **`MPI_Cart_coords`**`(MPI_Comm comm, int rank, int maxdims, int *coords)`

Determine ranks for cartesian shift:
`int` **`MPI_Cart_shift`**`(MPI_Comm comm, int direction, int disp, int *rank_source, int *rank_dest)`

Split into lower dimensional sub-grids:
`int` **`MPI_Cart_sub`**`(MPI_Comm comm, int *remain_dims, MPI_Comm *newcomm)`

Related Functions: *MPI_Graph_create*, *MPI_Graph_get*, *MPI_Cart_get*, *MPI_Graph_map*, *MPI_Cart_map*

Structures and Constants

The `MPI_Status` structure contains the following fields:

- `MPI_SOURCE`: id of processor sending the message
- `MPI_TAG`: the message tag
- `MPI_ERROR`: error status

Wildcards:
`MPI_ANY_TAG, MPI_ANY_SOURCE`

Elementary Datatypes:
`MPI_CHAR, MPI_SHORT, MPI_INT, MPI_LONG, MPI_UNSIGNED_CHAR, MPI_UNSIGNED_SHORT, MPI_UNSIGNED, MPI_UNSIGNED_LONG, MPI_FLOAT, MPI_DOUBLE, MPI_LONG_DOUBLE, MPI_BYTE, MPI_PACKED`

Reserved Communicators:
`MPI_COMM_WORLD, MPI_COMM_SELF`

Reduction Operations:
`MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD, MPI_BAND, MPI_BOR, MPI_BXOR, MPI_LAND, MPI_LOR, MPI_LXOR`

Other:
`MPI_STATUS_IGNORE`