Abstract

This is by no means a comprehensive list of MPI Functions, just the ones I have implemented successfully into my codes. Hopefully this is a little more detailed explanation of what the functions do and how to use them.

1 MPI

`int main(int argc, char**argv)`

`MPI_Init(&argc, &argv)`

`MPI_Finalize()`

2 Derived Data Type Routines

`MPI_Type_struct(count, blk_lens, index, data_types, &new_type)`

Creates a new MPI data type similar to a struct

- `count` {`int`} (input) - Number of blocks
- `blk_lens` {`int`} (input) - Array of the number of elements in each block
- `index` {`MPI_Aint`} (input) - Array of the byte displacement of each block
- `data_types` {`MPI_Datatype`} (input) - Array of the MPI datatypes in each block.
- `new_type` {`MPI_Datatype`} (output) - Name of the new struct

Example:
```
int count = 2;
int blk_lens[count] = {2,3};
MPI_Aint index[count] = {0,2*sizeof(int)};
MPI_Datatype old_types[count] = {MPI_INT, MPI_DOUBLE};
MPI_Datatype new_Struct;

MPI_Type_Struct(count, blk_lens, index, old_types, &new_Struct);
```

`MPI_Type_commit(&type)`

Must follow the creation of a new data type in MPI.

3 Group Routines

`MPI_Group_incl(old_GROUP, size, ranks, new_GROUP)`

Creates a new group from the processors specified in the previous group

- `old_GROUP` {`MPI_Group`} (input) - Original group from which the subset is taken
- `size` {`int`} (input) - Size of the subset taken from `old_Group`
- `ranks` {`int`} (input) - Ranks of the processors in the desired subset of `old_Group`
- `new_GROUP` {`MPI_Group`} (output) - Group created function, subset of `old_Group`

Example:
```
int size = 3;
int ranks[size] = {0,1,2};

MPI_Comm new_COMM;
MPI_Group old_GROUP, new_GROUP;

MPI_Group_incl(old_GROUP, size, ranks, new_GROUP);
```

4 Communicator Routines

`MPI_Comm_size(COMM, &size)`

Determines the number of processors that are being used by the specified communicator

- `COMM` {`MPI_Comm`} (input) - Communicator
- `size` {`int`} (output) - Returns the number of processes on the given communicator

Example:
```
MPI_Comm_size(MPI_COMM_WORLD, &size)
```

`MPI_Comm_rank(COMM, &rank)`

Determines the specific rank of the current processor

- `COMM` {`MPI_Comm`} (input) - Communicator
- `rank` {`int`} (output) - Returns the rank of the current processor

Example:
```
MPI_Comm_rank(MPI_COMM_WORLD, &rank)
```

`MPI_Comm_create(COMM, GROUP, &new_COMM)`

Creates a new communicator from a group on a previous communicator. Note, usually preceded by `MPI_Group_incl`

- `COMM` {`MPI_Comm`} (input) - Original Communicator
- `GROUP` {`MPI_Group`} (input) - Group of processors on original communicator
- `new_COMM` {`MPI_Comm`} (output) - Name of new communicator created by routine

Example (4 Processors):
```
int size = 3;
int ranks[size] = {0,1,2};

MPI_Comm new_COMM;
MPI_Group old_GROUP, new_GROUP;

MPI_Group_incl(old_GROUP, size, ranks, new_GROUP);
MPI_Comm_create(MPI_COMM_WORLD, new_GROUP, &new_COMM)
```

`MPI_Comm_group(COMM, &GROUP)`

Creates a group on the specified communicator that includes all processors on that communicator

- `COMM` {`MPI_Comm`} (input) - Communicator
- `GROUP` {`MPI_Group`} (output) - Group that includes all processors on `COMM`

Example:
```
MPI_Comm_group(MPI_COMM_WORLD, &new_GROUP)
```

`MPI_Comm_free(COMM)`

Deallocates the specified communicator.

- `COMM` {`MPI_Comm`} (input) - Communicator

Example:
```
MPI_Comm_free(new_COMM)
```

5 Virtual Topology Routines

`MPI_Cart_create(COMM, num_dims, dims, periodic, reorder, &new_COMM )`

Creates a new communicator with only the processors involved in the cartesian grid.

- `COMM` {`MPI_Comm`} (input) - Communicator
- `num_dims` {`int`} (input) - The number of dimensions in the grid (generally 2 or 3)
- `dims` {`int`} (input) - Array of size [`num_dims`] that specifies how many processors in each dimension
- `periodic` {`int`} (input) - Logical array of size [`num_dims`] specifying whether there is a periodic boundary condition in each dimension (0 for false, 1 for true)
- `reorder` {`int`} (input) - Specifies whether or not the processors can be reordered according to the new cartesian grid (0 for false, 1 for true)
- `new_COMM` {`MPI_Comm`} (output) - New communicator created by the function

Example:
```
int num_dims = 2;
int dims[num_dims] = {2,3};
int pdc[num_dims] = {0,0};
int reorder = 1;
MPI_Comm new_COMM;

MPI_Cart_create(num_dims, dims, pdc, reorder, new_COMM)
```

`MPI_Cart_shift(COMM, dim, dir, &nbr_pre, &nbr_post)`

Finds the neighboring processors to the current processor in the specified dimension. The neighboring processors are used in the communication routines

- `COMM` {`MPI_Comm`} (input) - Communicator
- `dim` {`int`} (input) - The desired dimension
- `dir` {`int`} (input) - > 0 shift forward, < 0 shift backward (use > 0)
- `nbr_pre` {`int`} (output) - Rank of processor previous to current processor
- `nbr_post` {`int`} (output) - Rank of processor proceeding the current processor

Example:
```
MPI_Comm GRID_COMM;
int LEFT = 0, RIGHT = 1;
int NBR[2];

MPI_Cart_shift(GRID_COMM, 1, 1, &NBR[LEFT], &NBR[RIGHT])
```

MPI_Cart_coords(COMM, rank, num_dims, coords)

- COMM {MPI_Comm} (input) - Communicator
- rank {int} (input) - Rank of current processor
- num_dims {int} (input) - Number of dimensions
- coords {int} (output) - Output array of size [num_dims] specifying the processor coordinates of the current processor

Example:
```
int rank;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
int coords;

MPI_Cart_coords(MPI_COMM_WORLD, rank, 2, coords)
```

MPI_Dims_create(num_tasks, num_dims, dims)

- num_tasks {int} (input) - Total number of processors
- num_dims {int} (input) - Number of dimensions (Generally 2 or 3)
- dims {int} (in/out) - Array of size [num_dims] specifying the number of processors in each dimension

Example:
```
int size;
MPI_Comm_size(MPI_COMM_WORLD, &size);
int num_dims = 2;
int dims[num_dims];

MPI_Dims_create(size, num_dims, dims)
```

---

6 Communication Routines

MPI_Allreduce(&src, &dest, tag, datatype, op, COMM)

Makes the value of a variable the same across all processors, using an

- src {int/double/float} (input) - Variable to be reduced
- dest {int/double/float} (output) - Destination of result
- tag {int} (input) - Generally 0 or 1
- datatype {MPI_Datatype} (input) - The corresponding MPI datatype of src
- op {MPI_Op} (input) - The desired operation such as MPI_SUM
- COMM {MPI_Comm} (input) - Name of communicator the operation is to take place on

Example:
```
MPI_Allreduce()
```

MPI_Isend(&src, size, datatype, dest, tag, COMM, &req)

Non-blocking pass of data from one processor to another (point-to-point communication).

- src {int/double/float} (input) - Data desired to be sent (source)
- size {int} (input) - Number of Data points to send
- datatype {MPI_Datatype} (input) - Type of data being sent

- `dest` {`int`} (input) - Processor number that the data is being sent to
- `tag` {`int`} (output) - Generally 0 or 1
- `COMM` {`MPI_Comm`} (input) - Communicator on which the data is being sent
- `req` {`MPI_Request`} (output) - Communication request

Example:
`MPI_Isend()`

`MPI_Irecv(&dest, size, datatype, src, tag, COMM, &req)`

Non-Blocking receive function.

- `dest` {`int/double/float`} (output) - Destination of the data being sent
- `size` {`int`} (input) - Number of Data points to send
- `datatype` {`MPI_Datatype`} (input) - Type of Data being sent
- `src` {`int/double/float`} (input) - Processor number that the data is being sent from
- `tag` {`int`} (input) - Generally 0 or 1
- `COMM` {`MPI_Comm`} (input) - Communicator on which the data is being sent
- `req` {`MPI_Request`} (output) - Communication Request

Example:
`MPI_Irecv()`

`MPI_Waitall(size, req, stat)`

Forces all processes to wait until all requests are filled. Basically makes the non-blocking send a blocking send.

- `size` {`int`} (input) - Communicator
- `req` {`MPI_Request`} (input) - Communication Request
- `stat` {`MPI_Status`} (output) - Communication Status

Example:
`MPI_Waitall()`

7 Misc Routines

`MPI_Barrier(COMM)`

Creates a barrier that must be reached by all processors before the program can proceed.

-

Example:
`MPI_Barrier(MPI_COMM_WORLD)`