* MPI\_INIT (ierr): *Initialize MPI*
* MPI\_COMM\_SIZE (comm, nProc, ierr): *Find out how many processes there are*
* MPI\_COMM\_RANK (comm, myid, ierr): *Find out which process I am*
* MPI\_BARRIER (comm, ierr): *Ensure every process has completed the computation*
* MPI\_FINALIZE (ierr): *Terminate MPI*
* MPI\_REDUCE (sendbuf, recvbuf, count, datatype, operation, root, comm, ierr)

*It performs a reduction of data from each process onto the specified root process. The root process must wait for all other process to enter the call.*

* MPI\_ALLREDUCE (sendbuf, recvbuf, count, datatype, operation, comm, ierr)

*There is no root, all processes receive the results.*

* MPI\_SCATTERV (sendbuf, sendcounts, displs, sendtype, recvbuf, recvcount, recytype, root, comm)

*Scatters a buffer in parts to all processes in a communicator.*

sendbuf: *address of send buffer;* sendcounts: *integer array specifying the number of elements to send to each processor;* displs: *integer array and entry i specifies the displacement, relative to sendbuf from which to take the outgoing data to process i.* recvbuf: *address of receive buffer (output);* recvcount: *number of elements in receive buffer.*

* MPI\_GATHERV (sendbuf, sendcount, sendtype, recvbuf, recvcounts, displs, rectytpe, root, comm)

*Gathers into specified locations from all processes in a group.*

sendbuf: *starting address of send buffer;* sendcount: *integer number of elements in send buffer;*recvbuf: *address of receive buffer (output);* recvcounts: *integer array containing the number of elements that are received from each process;*displs: *integer array and entry i specifies the displacement relative to recvbuf at which to place the incoming data from process i.*

* MPI\_SEND(buf, count, datatype, dest, tag, comm)

*Performs a* ***blocking*** *send.*

buf: *initial address of send buffer;* count: *integer number of elements in send buffer;* dest: *integer rank of destination.*

* MPI\_RECV(buf, count, datatype, source, tag, comm, status)

***Blocking*** *receives for a message.*

buf: *initial address of receive buffer;* count: *maximum integer number of elements in receive buffer;* source: *rank of source*

* MPI\_SENDRECV (sendbuf, sendcount, sendtype, dest, sendtag, recvbuf, recvcount, recvtype, source, recvtag, comm, status)

*Sends and receives a message*

sendbuf: *initial address of send buffer;* sendcount: *integer number of elements in send buffer;* dest: *integer rank of destination;* recvbuf: *initial address of receive buffer (output);* recvcount: *integer number of elements in receive buffer;* source: *integer rank of source*

* MPI\_BCAST (buffer, count, datatype, root, comm, ierr)

*It sends data from one process in one group to all processes in the other group.*

* MPI\_DIMS\_CREATE (nProc, ndims, dims)

*It creates a division of processors in a cartesian grid. MPI attempts to balance the distribution by minimising the difference in the number of processes assigned to each dimension.*

ndims: *integer number of cartesian dimensions;* dims: *integer array of size ndims specifying the number of nodes in each dimension (output).*

* MPI\_CART\_CREATE (comm\_old, ndims, dims, periods, reorder, comm\_cart)

*It creates a Cartesian decomposition of the processes.*

comm\_old: *input communicator;* ndims: *integer number of cartesian dimensions;* dims: *integer array of size ndims specifying the number of nodes in each dimension;* periods: *logical array of size ndims specifying whether the grid is periodic (true) or not (****false****) in each dimension;* reorder: *ranking may be reordered (****true****) or not (false);* comm\_cart: *communicator with new cartesian topology*

* MPI\_CART\_SHIFT (comm, direction, disp, source, dest)

*It returns the shifted source and destination ranks, given a shift direction and amount.*

comm: *communicator with cartesian structure;* direction: *coordinate dimension of shift, in the range [0, n-1] for an n-dimensional Cartesian mesh;* disp: *displacement (> 0: upwards shift, < 0: downwards shift);* source: *rank of source process (output);* dest: *rank of destination process (output)*

* MPI\_CART\_GET (comm, maxdims, dims, periods, coords)

*It retrieves Cartesian topology information associated with a communicator.*

maxdims: *length of vectors dims;* dims: *integer array for processes number in each cartesian dimension;* periods: *logical array of periodicity (true/false) for each cartesian dimension;* coords: *integer array of coordinates for calling process in cartesian structure (output).*

* MPI\_TYPE\_VECTOR (count, blocklength, stride, oldtype, newtype)

*It creates a vector (strided) datatype.*

count: *number of blocks;* blocklength: *number of elements in each block;* stride: *number of elements between start of each block.*