MPI_Send:

Syntax: MPI_Send(void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

Function: Sends a message from the specified buffer to a process with the given destination rank.

Arguments:

buf: Pointer to the send buffer.

count: Number of elements in the send buffer.

datatype: MPI datatype of the elements being sent.

dest: Rank of the destination process.

tag: Message tag for the communication.

comm: Communicator that defines the group of processes involved in the communication.

MPI_Ssend:

Syntax: MPI_Ssend(void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

Function: Similar to MPI_Send, but it uses synchronous send mode, ensuring the receive operation starts before the send call completes.

Arguments: Same as MPI_Send.

MPI_Isend:

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

Function: Initiates a non-blocking send operation, allowing the program to continue execution without waiting for the send to complete.

Arguments:

buf: Pointer to the send buffer.

count: Number of elements in the send buffer.

datatype: MPI datatype of the elements being sent.

dest: Rank of the destination process.

tag: Message tag for the communication.

comm: Communicator that defines the group of processes involved in the communication.

request: Pointer to an MPI_Request object that can be used to query the status of the send operation.

MPI_Issend:

Syntax: MPI_Issend(void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request* request)

Function: Similar to MPI_Isend, but uses synchronous send mode, ensuring the receive operation starts before the send call completes.

Arguments: Same as MPI_Isend.

MPI_Bsend:

Syntax: MPI_Bsend(void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

Function: Performs a buffered send operation, which uses an intermediate buffer to store the message until it is received by the destination process.

Arguments: Same as MPI_Send.

MPI_Recv:

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

Function: Receives a message from a specific source process and stores it in the specified receive buffer.

Arguments:

buf: Pointer to the receive buffer.

count: Maximum number of elements that can be received.

datatype: MPI datatype of the elements being received.

source: Rank of the source process.

tag: Message tag for the communication.

comm: Communicator that defines the group of processes involved in the communication.

status: Pointer to an MPI_Status object that provides information about the received message.

MPI_Sendrecv:

Syntax: 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)

Function: Combines both sending and receiving into a single call, allowing simultaneous communication between two processes.

Arguments:

sendbuf: Pointer to the send buffer.

sendcount: Number of elements in the send buffer.

sendtype: MPI datatype of the elements being sent.

dest: Rank of the destination process.

sendtag: Message tag for the send operation.

recvbuf: Pointer to the receive buffer.

recvcount: Maximum number of elements that can be received.

recvtype: MPI datatype of the elements being received.

source: Rank of the source process.

recvtag: Message tag for the receive operation.

comm: Communicator that defines the group of processes involved in the communication.

status: Pointer to an MPI_Status object that provides information about the received message.

MPI_Scatter:

Syntax: MPI_Scatter(void* sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

Function: Distributes data from the root process to all processes in the communicator in a scatter-like fashion.

Arguments:

sendbuf: Pointer to the send buffer (used by the root process).

sendcount: Number of elements sent to each process from the send buffer.

sendtype: MPI datatype of the elements being sent.

recvbuf: Pointer to the receive buffer (used by each process).

recvcount: Number of elements received by each process into the receive buffer.

recvtype: MPI datatype of the elements being received.

root: Rank of the root process (the process that scatters the data).

comm: Communicator that defines the group of processes involved in the communication.

MPI_Gather:

Syntax: MPI_Gather(void* sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

Function: Gathers data from all processes in the communicator to the root process.

Arguments: Same as MPI_Scatter, but with reversed roles for sendbuf and recvbuf.

MPI_Allgather:

Syntax: MPI_Allgather(void* sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm)

Function: Gathers data from all processes and distributes it to all processes in the communicator, creating a copy of the data on each process.

Arguments: Same as MPI_Gather, but with recvbuf used by all processes.

MPI_Allgatherv:

Syntax: MPI_Allgatherv(void* sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf, int* recvcounts, int* displs, MPI_Datatype recvtype, MPI_Comm comm)

Function: Gathers data from all processes and distributes it to all processes, with varying amounts of data received by each process.

Arguments:

sendbuf: Pointer to the send buffer (used by each process).

sendcount: Number of elements sent by each process from the send buffer.

sendtype: MPI datatype of the elements being sent.

recvbuf: Pointer to the receive buffer (used by each process).

recvcounts: Array specifying the number of elements received from each process.

displs: Array specifying the displacement of the data for each process in the receive buffer.

recvtype: MPI datatype of the elements being received.

comm: Communicator that defines the group of processes involved in the communication.

MPI_Barrier:

Syntax: MPI_Barrier(MPI_Comm comm)

Function: Synchronizes all processes in the communicator, ensuring that no process proceeds past the barrier until all processes have reached it.

Arguments:

comm: Communicator that defines the group of processes involved in the barrier synchronization.

MPI_Bcast:

Syntax: MPI_Bcast(void* buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)

Function: Broadcasts data from the root process to all other processes in the communicator.

Arguments:

buffer: Pointer to the send/receive buffer used by each process.

count: Number of elements sent/received by each process.

datatype: MPI datatype of the elements being sent/received.

root: Rank of the root process (the process that broadcasts the data).

comm: Communicator that defines the group of processes involved in the communication.

MPI_Reduce:

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

Function: Reduces values from all processes in the communicator to a single result on the root process, using a specified reduction operation.

Arguments:

sendbuf: Pointer to the send buffer (used by each process).

recvbuf: Pointer to the receive buffer (used by the root process).

count: Number of elements sent/received by each process.

datatype: MPI datatype of the elements being sent/received.

op: MPI reduction operation to be applied during the reduction.

root: Rank of the root process (the process that receives the reduced result).

comm: Communicator that defines the group of processes involved in the communication.

MPI_Cart_Shift:

Syntax: MPI_Cart_Shift(MPI_Comm comm, int direction, int displ, int* source, int* dest)

Function: Computes the source and destination ranks for shifting data in a Cartesian topology.

Arguments:

comm: Communicator that defines the Cartesian topology.

direction: Coordinate direction of the shift.

displ: Number of positions to shift.

source: Pointer to the variable that will receive the rank of the source process.

dest: Pointer to the variable that will receive the rank of the destination process.

MPI_Cart_Create:

Syntax: MPI_Cart_Create(MPI_Comm comm_old, int ndims, const int* dims, const int* periods, int reorder, MPI_Comm* comm_cart)

Function: Creates a new communicator with a Cartesian topology based on an existing communicator.

Arguments:

comm_old: The original communicator.

ndims: Number of dimensions in the Cartesian grid.

dims: Array specifying the size of each dimension.

periods: Array specifying whether each dimension is periodic or not.

reorder: Flag indicating whether ranks can be reordered.

comm_cart: Pointer to the new communicator with the Cartesian topology.

MPI_Comm_rank:

Syntax: MPI_Comm_rank(MPI_Comm comm, int* rank)

Function: Retrieves the rank of the calling process in the specified communicator.

Arguments:

comm: The communicator.

rank: Pointer to the variable that will receive the rank of the calling process.

MPI_Comm_size:

Syntax: MPI_Comm_size(MPI_Comm comm, int* size)

Function: Retrieves the size (number of processes) in the specified communicator.

Arguments:

comm: The communicator.

size: Pointer to the variable that will receive the size of the communicator.

MPI_Type_Vector:

Syntax: MPI_Type_Vector(int count, int blocklength, int stride, MPI_Datatype oldtype, MPI_Datatype* newtype)

Function: Creates a new datatype representing a strided vector derived from an existing datatype.

Arguments:

count: Number of blocks in the vector.

blocklength: Number of elements in each block.

stride: Distance between the start of each block (in multiples of the old datatype size).

oldtype: The old datatype to be used as the base type.

newtype: Pointer to the variable that will receive the newly created datatype.

MPI_Type_create_subarray:

Syntax: MPI_Type_create_subarray(int ndims, const int* sizes, const int* subsizes, const int* starts, int order, MPI_Datatype oldtype, MPI_Datatype* newtype)

Function: Creates a new datatype representing a subarray derived from an existing datatype.

Arguments:

ndims: Number of dimensions in the full array.

sizes: Array specifying the size of each dimension in the full array.

subsizes: Array specifying the size of each dimension in the subarray.

starts: Array specifying the starting position of the subarray in each dimension of the full array.

order: Ordering of the dimensions (either MPI_ORDER_C or MPI_ORDER_FORTRAN).

oldtype: The old datatype to be used as the base type.

newtype: Pointer to the variable that will receive the newly created datatype.