

Chapter 9

Message Passing Interface (MPI)

9. Message Passing Interface

- Message Passing Interface (MPI) provides a standard library of routines for writing portable and efficient message passing programs.
- MPI is not a language.
- It is a specification of a library of routines that can be called from programs.
 - Has a rich collection of point-to-point communication routines for data movement, global computation, and synchronization.
- An MPI application is a collection of concurrent communicating tasks.

9. Message Passing Interface

- A program includes code written by the application programmer that is linked with a function library provided by the MPI software implementation.
- Each task is assigned a unique rank within certain context (an integer number between 0 and $n-1$) for an MPI application of n tasks.
- These ranks are used by MPI tasks to identify each other
 - In sending/receiving messages.
 - To execute collective operations.
 - To cooperate.
- MPI tasks can be run on the same processor or on different processors concurrently.

9.1 Communicators

- In MPI there is no virtual machine. Therefore, using just a message tag is not enough to safely distinguish library messages from user messages.
- The concept of communicator is introduced in MPI to achieve this safe communication requirement.
- A communicator is a binding of a communication context to a group of tasks.

9.1 Communicators

- A communicator is an object that can be accessed via a handle of type `MPI_COMM`.
- Communicators are classified into:
 - Intracommunicators: for operations within a single group of tasks.
 - Intercommunicators: for operations between different groups of tasks.
- When an MPI application starts, all tasks are associated to a “world” communicator.

9.1 Communicators

- Task groups
 - A group in MPI is an object that can be accessed via a handle of the predefined type `MPI_Group`.
 - Members of a group are assigned unique identifiers within the group called ranks.
 - A group is an ordered set of ranks that are contiguous and start from 0.
 - MPI provides a number of functions to create new groups from existing ones.

9.1 Communicators

- Task groups
 - No functions to create groups from scratch are provided.
 - At the beginning, all tasks belong to one base group from which others can be formed.
 - Members of a new group can be solicited from one or more groups.
 - Also, given an existing group, a new group can be formed by either excluding a set of tasks or by only including a set of tasks from the existing group.
 - A new group can also be formed using 2 existing groups using set operations: union, intersection, or difference.

9.1 Communicators

- Default communicator
 - `MPI_COMM_WORLD`: default communicator.
 - Once `MPI_Init ()` is called, the default communicator defines a single context including the set of all MPI tasks available for the computation.
 - `MPI_COMM_SELF` is a predefined communicator which includes only the calling process itself.

9.1 Communicators

- Task Rank
 - Tasks involved in a communicator are assigned consecutive integer identifiers between 0 and size of communicator's group -1.
 - A task can find its rank within a communicator by calling the function `MPI_Comm_rank ()` that takes an existing communicator and returns the rank of the calling task.

9.1 Communicators

- Communicator's group
 - The group associated with a communicator can be retrieved using the function `MPI_Comm_group ()`.
 - This function takes an existing communicator and returns its corresponding group of tasks.
 - `MPI_Comm_group ()` is important to create a base group from which other groups can be formed (since MPI doesn't provide functions to create groups from scratch)
 - The size of the group associated with a communicator can be determined by calling the function `MPI_Comm_Size()`.
 - This function takes an existing communicator and returns the size of the corresponding group.

9.1 Communicators

- Creating new Communicators
 - 3 collective functions can be used to create new communicators.
 - It is required that these functions be called by all tasks that belong to the existing communicator even if some tasks will not belong to the new one.

9.1 Communicators

- Creating new Communicators
 - `MPI_Comm_dup (oldcomm, &newcomm)`:
 - This function duplicates the existing `oldcomm`.
 - It returns in `newcomm` a new communicator with the same group of tasks but in a new context.
 - `MPI_Comm_create (oldcomm, group, &newcomm)`:
 - This function creates a new communicator `newcomm` with a corresponding group of tasks `group`.
 - `Group` must be a subset of the set of tasks associated with `oldcomm`.

9.1 Communicators

- Creating new Communicators
 - `MPI_Comm_split (oldcomm,split_key,rank_key,&newcomm)`:
 - This function partitions the group of `oldcomm` into disjoint subgroups, one for each value in the argument `split_key`.
 - » Within each subgroup are all the tasks of the same `split_key`.
 - Within each subgroup the tasks are ranked in the order defined by the value of the argument `rank_key`.
 - » Ties are broken according to the rank in the group associated with the old communicator.
 - A new communicator is created for each subgroup and returned in `newcomm`.
 - This function must be called by all tasks in `oldcomm` even if user doesn't wish to assign every task to a new communicator.
 - » If a task provides `MPI_UNDEFINED` as the argument `split_key`, it will get the value `MPI_COMM_NULL` as `newcomm`.

9.1 Communicators

- Intercommunicator
 - A general type of communicator targeted towards group-to-group communication.
 - Each intercommunicator contains a local group and a remote group.
 - The owner of the intercommunicator is always a member of the local group.

9.1 Communicators

- Intercommunicator
 - The local group is accessible using functions such as `MPI_Comm_group ()` and `MPI_Comm_size ()`
 - Information about the remote group can be accessed using `MPI_Comm_remote_group ()` and `MPI_Comm_remote_size()`
 - Intercommunicator can be used in creating tasks dynamically.
 - New tasks in MPI can be started using spawn functions to start new tasks and establish communication with them.

9.2 Virtual Topologies

- Cartesian Topology

- MPI_Cart_create() can be used to create Cartesian structures of arbitrary dimensions.
- It takes the following as input and returns a handle to a communicator with the new Cartesian structure:
 - An existing communicator that defines the set of tasks on which the topology to be mapped.
 - Number of dimensions in Cartesian structure.
 - Size of each dimension.
 - Whether or not the structure is periodic at each dimension.
 - Whether or not the system is allowed to optimize the mapping of the virtual topology on the underlying physical processors.
- MPI_Cart_create (oldcomm, ndims, sizeofdims, periods, mapping, newcomm).

9.2 Virtual Topologies

- Retrieval of tasks coordinates and ranks
 - A task is allowed to inquire about its rank and coordinates in a Cartesian structure using the function: `MPI_Cart_rank` (communicator, coords, &rank)
 - The function `MPI_Cart_coords` (communicator, rank, ndims, &coords) returns the coordinates of the task with rank in the Cartesian structure communicator of dimensions ndims.

9.2 Virtual Topologies

- Graph topology
 - `MPI_Graph_create` (`oldcomm`, `nnodes`, `index`, `edges`, `mapping`, `newcomm`) can be used to create a new communicator to which a graph topology information is attached.

9.3 Task Communication

- Communication modes
 - Standard send
 - The sender blocks until its message has been safely copied into either a matching receive buffer or a temporary system buffer.
 - MPI implementation decides whether or not a message is to be buffered or not.
 - MPI_Send (buf, count, data_type, to_whom, tag, communicator)
 - The function will send the message stored starting at address buf to the task whose rank is given as to_whom.
 - The message consists of count elements, each of which of type data_type.
 - Both sender and receiver must be part of the same communicator.
 - If the buffer is used to store the outgoing message, the sender will continue without having to wait for a matching receive to be posted.

9.3 Task Communication

- Communication modes
 - Blocking receive
 - Is the standard receive function in MPI: a call to this function will not return until it receives the message it is expecting in its buffer.
 - `MPI_Recv(buf, count, data_type, from_whom, tag, communicator, &status)`
 - This receive will select a message with a matching sender (`from_whom`) and a matching message tag (`tag`) for receipt into the buffer (`buf`).
 - Buffered Send (`MPI_Bsend`)
 - Message buffering is guaranteed.
 - Once the message information is buffered, the send call will return and the send buffer becomes reusable.

9.3 Task Communication

- Communication modes
 - Synchronous Send (MPI_Ssend)
 - Both the sender and receiver block until the send and receive calls are posted and the communication is complete.
 - Since standard receive is already blocking, then we just need a blocking send to accomplish synchronous communication.
 - MPI_Ssend () can start without having to wait for a matching receive call to be posted. However, it will not complete until a matching receive has been posted and the receiver has started to receive the message.
 - Ready Send (MPI_Rsend)
 - Can be started only after a matching receive has been posted.
 - The completion of the ready send does not depend on the status at the receiving end.

9.3 Task Communication

- Nonblocking communication
 - Can be accomplished in 3 steps for both send/receive:
 - Initiate a send/receive using `MPI_Isend()/MPI_Irecv()`.
 - Do some computation during communication time.
 - Complete communication using `MPI_Wait()` and `MPI_Test()`.

9.3 Task Communication

- Nonblocking communication
 - Initiating nonblocking communication
 - `MPI_Isend(buf, count, data_type, to_whom, tag, communicator, &request)`
 - » A call to `MPI_Isend ()` will return before the message is copied out of the send buffer.
 - » The request argument is a system object returned when the function is called. It is used to identify communication operations and match the operations that initiate and complete a nonblocking communication.

9.3 Task Communication

- Nonblocking communication
 - Completing nonblocking communication
 - MPI provides functions that test for the completion of nonblocking communication operations that have been initiated.
 - The completion of a nonblocking send operation implies that the data has been copied out of the send buffer and the sender can reuse the buffer for other purposes.
 - The completion of a nonblocking receive operation implies that the data has already been placed in the receive buffer and is ready to be accessed by the receiver.

9.3 Task Communication

- Nonblocking communication
 - Completing nonblocking communication
 - MPI_Test (request, &flag, &status) returns the current standing of the communication operation identified by the request. Flag will be set to true if the communication operation is complete and false otherwise.
 - MPI_wait (request, &status) will return after the completion of the communication operation identified by request.

9.3 Task Communication

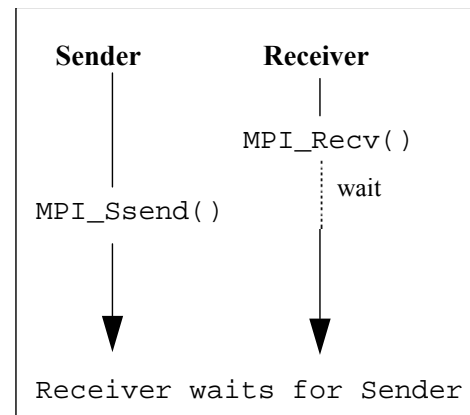
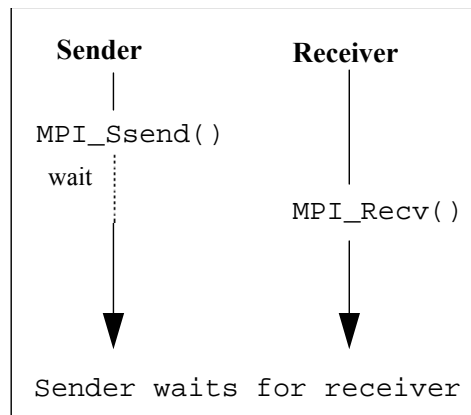
- Persistent Communication
 - MPI provides persistent communication requests constructs to help reduce the communication overhead between a task and the communication controller.
 - A persistent communication is created in MPI using one of the following functions:
 - MPI_Send_init ()
 - MPI_Bsend_init ()
 - MPI_Ssend_init ()
 - MPI_Rsend_init ()

9.4 Synchronization

- Precedence synchronization
 - Using blocking receive `MPI_Recv()` forces the receiving task to wait until a matching message is received.
 - The sender of this matching message may hold its message as long as it wants the receiver to wait.

9.4 Synchronization

- Communication Rendezvous
 - Synchronous mode of communication.
 - 2 tasks can rendezvous at a synchronization point.
 - If both send and receive are blocking, the communication will not complete at either end until both sender and receiver meet.

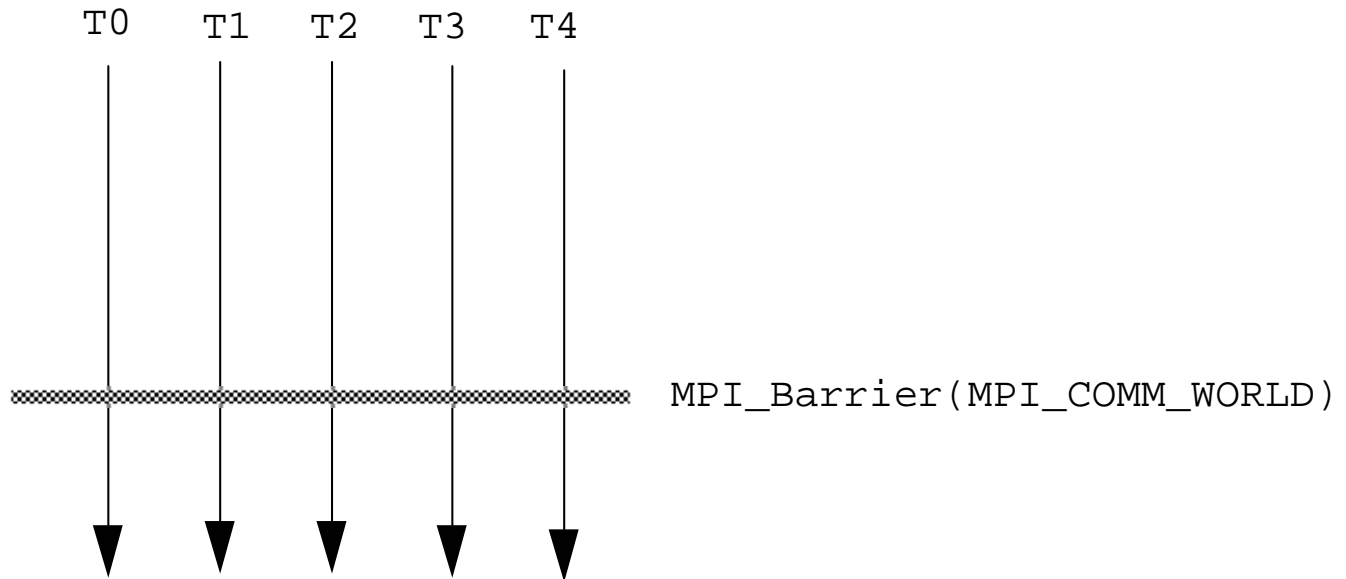


9.4 Synchronization

- Barriers
 - Tasks in a group can synchronize at a synchronization point using a barrier.
 - No task proceeds beyond the barrier until all tasks have checked in at that barrier.
 - Barrier synchronization is achieved by having all tasks in the communicator's group call the function: `MPI_Barrier (communicator)`.
 - A call to `MPI_Barrier ()` returns only after all the communicator's group members have executed their calls to this function.

9.4 Synchronization

- Barriers



9.5 Collective Operations

- Global Computation

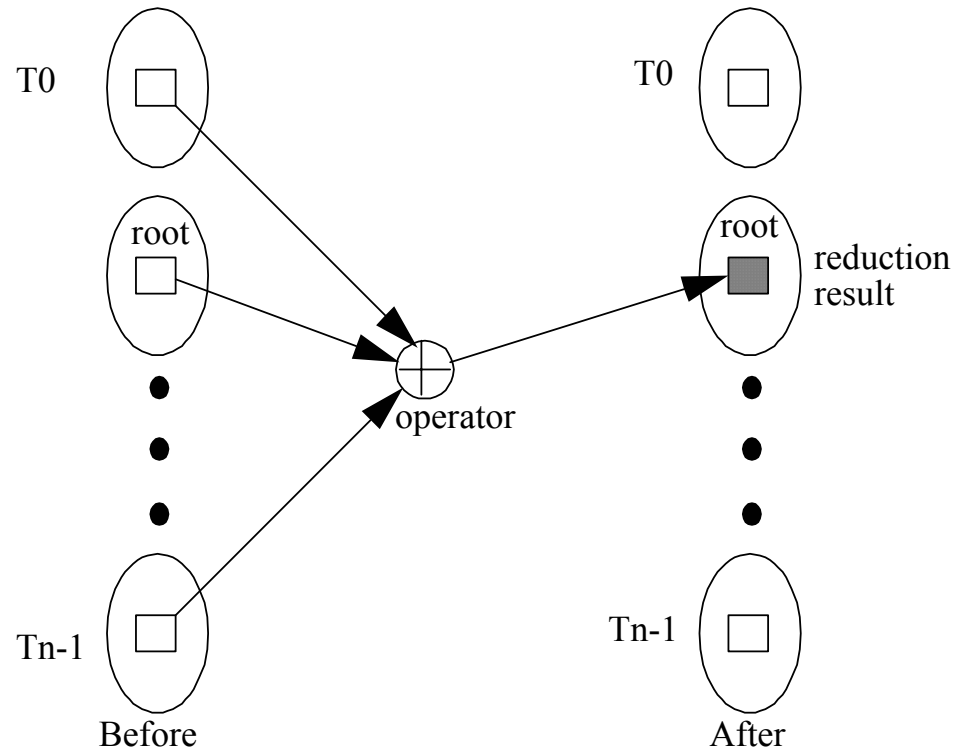
- In reduction, an associative, commutative operation is applied across data items provided by members of a group of tasks.
- The reduction operation can be user-defined function or MPI predefined operation such as sum, minimum, maximum.
- The result of applying a reduction operation may be returned to a single task called the root or maybe sent to every task in the group.
- `MPI_Reduce (sbuf, rbuf, n, data_type, op, rt, communicator)` returns the result only to the receive buffer at the root (global combine)
- `MPI_Allreduce (sbuf, rbuf, n, data_type, op, communicator)` returns the result to the receive buffers of all the members of the group (many-to-many).

9.5 Collective Operations

- Global Computation
 - Scan operations
 - 2 types prefix and postfix.
 - The result of a scan operation is different at each task based on the rank of the task.
 - `MPI_Scan (sbuf, rbuf, n, data_type, op, communicator)` performs a prefix reduction on data associated with group members.
 - After the execution of this function, the receive buffer of the task of rank i will have the reduction of the values in the send buffer of the tasks of ranks $0, 1, \dots, i$.

9.5 Collective Operations

- Global Computation



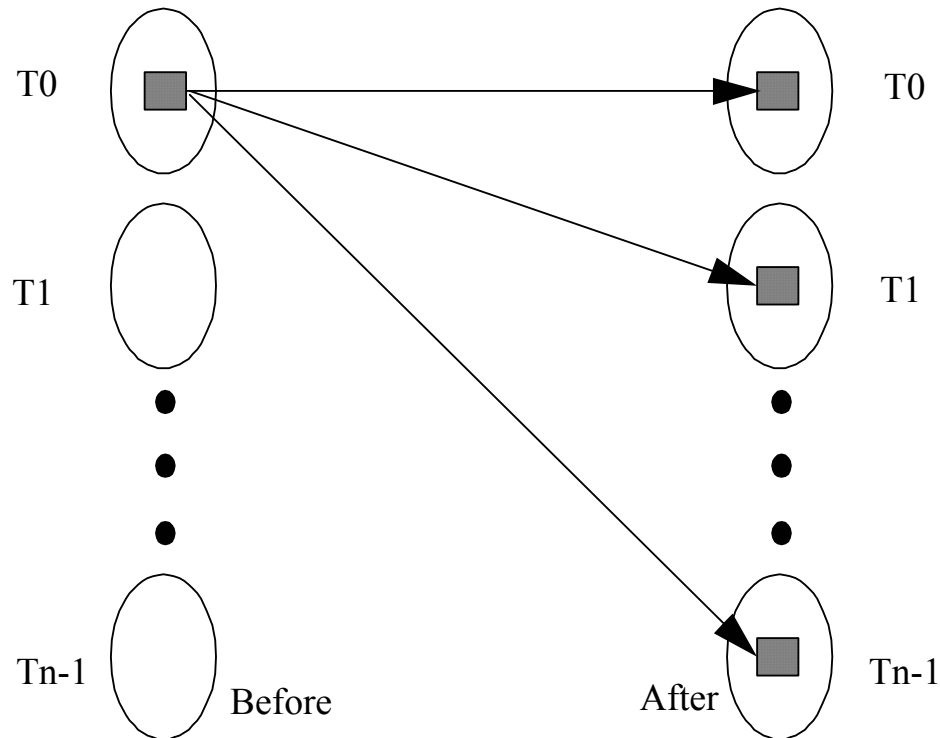
Reduce Operation

9.5 Collective Operations

- Data Movement Operation
 - MPI supports a broad variety of data movements collective functions.
 - The basic operations supported are broadcast (same message to every member), scatter (different message to each member) and gather (a dual scatter: one process will receive a message from each member in the group).
 - Broadcasting: `MPI_Bcast` (`buffer`, `n`, `data_type`, `root`, `communicator`).
 - The contents of the root's buffer will be copied to the buffers of all tasks.

9.5 Collective Operations

- Data Movement Operation



A Broadcast from Task T0 (root)

9.5 Collective Operations

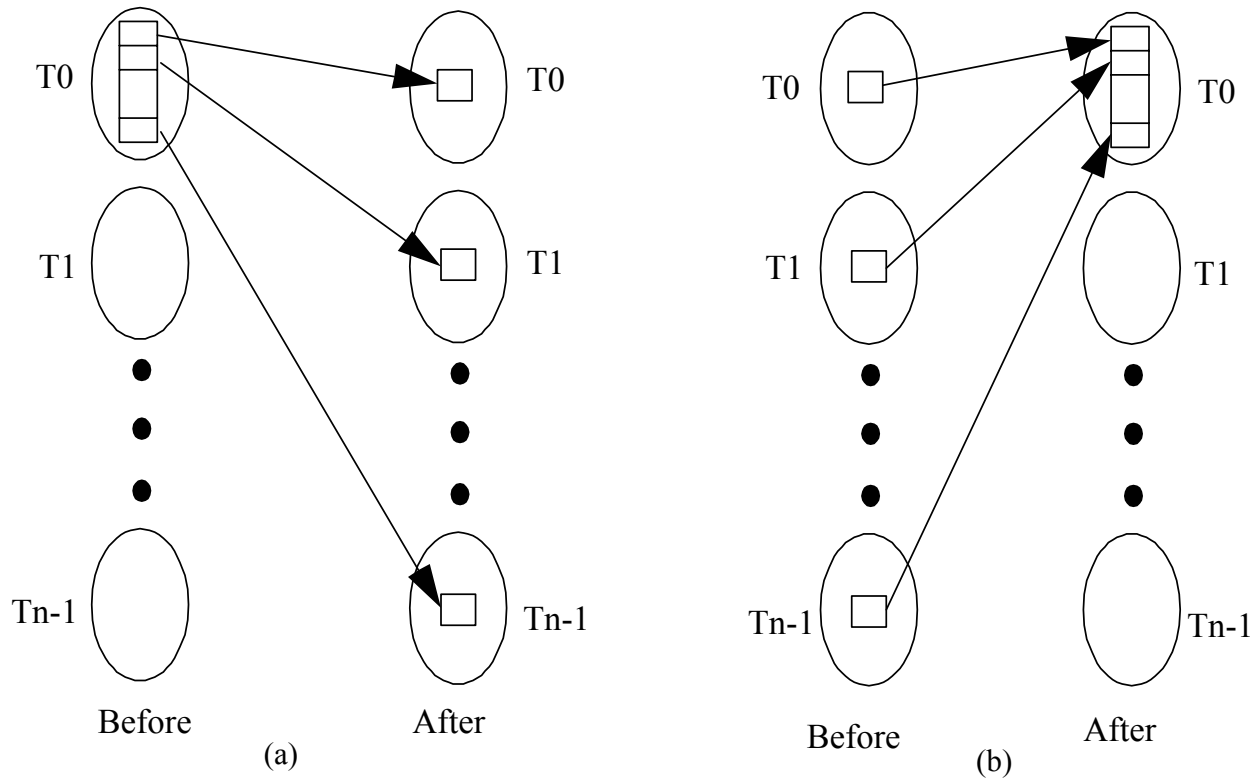
- Data Movement Operation
 - Scatter function allows one task to distribute its buffer to each member of the group.
 - Gather function allows a task to build its buffer from pieces of data collected from other members in a group.
 - MPI_Scatter (sbuf, n, stype, rbuf, rtype, rt, communicator).
 - MPI_Gather(sbuf, n, stype, rbuf, rtype, rt, communicator).

9.5 Collective Operations

- Data Movement Operation
 - In scatter, the send buffer at the root is divided into a number of segments each of size n . The first n elements in the root's send buffer are copied in the receive buffer of the first member of the group, the second n elements are copied into the buffer of the 2nd element, and so on.
 - In gather, each task including the root sends the contents of its send buffer to root task. The root stores them in rank order after receiving them.
 - These functions must be called by all members of a group using the same communicator and root.

9.5 Collective Operations

- Data Movement Operation



a) Scatter from the root task T0, b) Gather at the root task T0

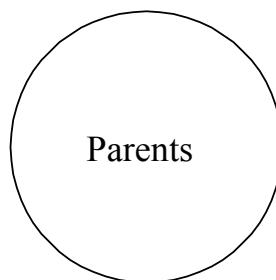
9.6 Task Creation

- Bridging between spawned and spawning tasks
 - Allowing dynamic tasks imply creation of new tasks.
 - But also requires establishment of communication between new and previously existing tasks.
 - We need to provide a bridging communicator between the old and new tasks: use intercommunicators.
 - On the spawning side, the local group of the intercommunicator is the group that did the spawning and the remote group is that was spawned.
 - On the spawned side, the local group is set to the `MPI_COMM_WORLD` of the new tasks, and the remote group is set to the group that did the spawning.

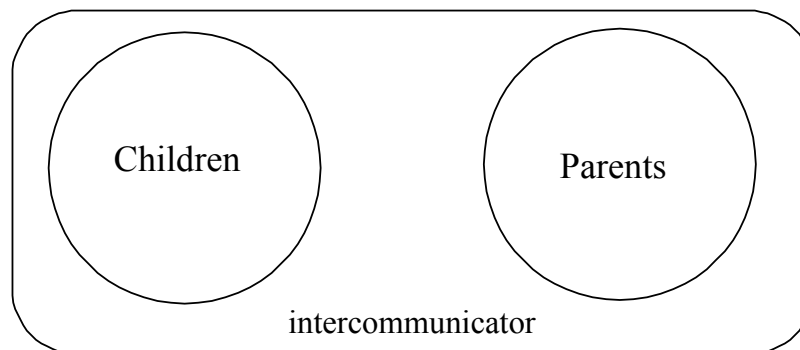
9.6 Task Creation

- Bridging between spawned and spawning tasks

Before Spawning



After Spawning



Creating children and establishing communication with them via intercommunicator

9.6 Task Creation

- Starting identical tasks
 - `MPI_Comm_Spawn ()` is used to create a number of identical copies of a given program and establish communication with them.
 - `MPI_Comm_Spawn (command, argv, maxprocs, info, root, comm., &inter_comm, &array_error_codes)`
- Starting multiple executables
 - `MPI_Comm_spawn_multiple ()` makes it possible to start multiple programs or the same program with multiple sets of arguments.

9.7 One-Sided Communication

- MPI has constructs for Remote Memory Access (RMA).
 - A task is allowed to access the remote memory of another task.
- These constructs are useful in applications with dynamically changing data access patterns but where the data distribution is fixed or slowly changing.
- In RMA, only one task is supposed to issue data transfer operation (origin).
- RMA is supported via 2 main operations, put and get.
 - Put is equivalent to send/receive.
 - Get is equivalent to receive/send.

9.7 One-Sided Communication

- Specifying Target Windows
 - Each task must specify a memory window made available for access by remote tasks using a collective operation that returns a window object.
 - The window object represents the group of tasks that own and access a set of windows and the attributes of each window as specified by each task.
 - `MPI_Win_init` (`base`, `size`, `disp_unit`, `info`, `comm`, `&win`).

9.7 One-Sided Communication

- Specifying Target Windows
 - This function is to be executed by all the tasks sharing the same communicator.
 - Each task specifies a window of existing memory for remote access by other tasks.
 - Different tasks can specify completely different target windows with different starting locations, sizes and displacement units.
 - If a task decides not to make any memory available for remote access, size is set to 0.

9.7 One-Sided Communication

- Put and Get Operations
 - MPI_Put (origin_addr, origin_count, origin_datatype, target_rank, target_disp, target_count, target_data_type, win).
 - Calling this function transfers data from origin task with rank target_rank in the group associated with the window object win.
 - MPI_Get(origin_addr, origin_count, origin_datatype, target_rank, target_disp, target_count, target_data_type, win).

9.7 One-Sided Communication

- Put and Get Operations
 - It is also possible to combine in a put operation the data transferred to the target task with the data already there.
 - An accumulate operation can use a reduction operation to combine the contents of the target buffer with the data transferred.
 - `MPI_Accumulate(origin_addr, origin_count, origin_datatype, target_rank, target_disp, target_count, target_datatype, op, win)`.

9.8 Summary

- MPI is a standard for writing message passing programs.
- It was developed in 1993 and 1994 by an international group called the MPI Forum.
- The Forum expanded the scope of MPI to encompass a wider set of distributed and parallel programming constructs.
- Currently, MPI is used in clusters and other message passing systems due to its rich functionality.