COS786: Parallel and Distributed Computing

MPI: Groups, Contexts and Communicators
Why Libraries?

- Shield programmers from the implementations of key algorithms.
- Obviate the need to re-invent key data structures and algorithms.
- Hide variations that are designed for various problems and situations.
Required Support for Libraries

• Safe communication space (akin to a type namespace).

• Group scope for collective processes.

• Abstract process naming.

• The ability to adorn a set of communicating processes with additional, user-defined attributes. For example, adding extra collective operations.
Support Provided by MPI

- Contexts of communication.
- Groups of processes.
- Virtual topologies.
- Attribute caching.

All of the above are encapsulated by Communicators.
Communicators

- Divided into two types:
  - Intra-communicators: for operations within a group.
  - Inter-communicators: for point-to-point communication between two groups.
Groups

• A group defines an ordered collection of processes and each process is assigned a rank.

• Groups define the scope for process names in point-to-point communication.

• Groups define the scope for collective operations.

• While groups may be manipulated separately, only communicators may be used in communication operations.
Intra-communicators

- Intra-communicators contain:
  - An instance of a group
  - Communication contexts for point-to-point and collective communication.
  - The ability to include a virtual topology and other attributes.
Intra-communicators

- A context provides a unique universe in which messages are transmitted.
  - This allows for libraries to insulate their internal communication from external communication.
  - Distinct libraries and multiple instances of the same library therefore do not interfere with one another.
  - It also implies that a library may be invoked without synchronizing on the entry/exit code or waiting for pending communications.
Intra-communicators

- Groups define the participants in a communication.
- A virtual topology defines the mappings of ranks in a group to and from a topology.
- Attributes define the local information that a user or library may add to a communicator for later reference.
Intra-communicators

- A predefined communicator, MPI_COMM_WORLD, is provided to include all processes in a communication.
- Processes are assigned ranks in consecutive order.
- Point-to-point communication identifies participants by rank.
- Collective communication (broadcasts) involves all of the processes.
Inter-communicators

- Provide for communication between universes (groups).
- When building an application from multiple, parallel modules (libraries) it is convenient to be able to use local ranks when addressing processes in the second module.
- Inter-communication support provides a means of creating processes dynamically.
Inter-communicators

- The context establishes a safe universe in which the two participating groups may communicate.
  - A send in one group is a receive in the other and vice versa.
  - However, there is no general purpose collective communication provided by inter-communicators, which facilitate only the isolation of point-to-point communications.
Inter-communicators

- The local and remote group specify the recipients and destinations for an inter-communicator.
- Virtual topologies are undefined.
- As with intra-communicators, the attributes cache defines local information added by the user or library for later reference.
Inter-communicators

• MPI support:
  – MPI provides support for creating and manipulating inter-communicators.
  – Inter-communicators are used similarly to intra-communicators in point-to-point communications.
  – Collective operations via inter-communicators must be layered on top of MPI by the user.
  – Inter-communicator support for overlapping groups must be layered on top of MPI by the user.
Groups

- A group is an ordered set of process identifiers.
- Each process in a group is associated with a rank.
  - Ranks are contiguous and are allocated from zero.
- Groups cannot be transferred among processes.
- Group objects are used inside communicators to describe the participants inside a communication universe.
Groups

• MPI provides two group constants:
  – MPI_GROUP_EMPTY
  – MPI_GROUP_NULL
Contexts

- A context is a property of communicators.
- A context partitions the communication space.
- A message sent in one context can not be received in another context.
- Where applicable, collective operations are independent of point-to-point operations.
Intra-communicators

- Combine the concepts of group and context.
- MPI communication operations reference communicators in order to determine the scope and the communication universe in which point-to-point or collective operations should operate.
- Each communicator contains a group of valid participants.
  - This group always includes the local process.
  - Source and destination of a message are determined by the process rank within that group.
Intra-communicators

• The communicator restricts the spatial scope of communication and;
• provides machine-independent process addressing through ranks.
• Intra-communicator objects cannot be directly transferred from one process to another.
Intra-communicators

- MPI provides the following constants:
  - MPI_COMM_WORLD
  - MPI_COMM_SELF
  - MPI_COMM_NULL
Group Management

- Operations that access groups are local and do not require interprocess communication.
Group Management

MPI_GROUP_RANK(group, rank)

• group – group handle
• (OUT) rank – the rank of the calling process or MPI_UNDEFINEd
Group Management

MPI_GROUP_TRANSLATE_RANKS(group1, n, ranks1, group2, ranks2)

• group1 – handle of first group
• n – number of ranks in ranks1 and ranks2
• ranks1 – array of valid ranks in group1
• group2 – handle of second group
• (OUT) ranks2 – array of corresponding ranks in group2
Group Construction

- MPI does not provide a facility for creating groups from scratch.
- The base group is associated with MPI_COMM_WORLD.
- Group construction is a local operation.
- Distinct groups may be created on different processes.
- A process may create a group that does not include itself.
Group Construction

MPI_COMM_GROUP(comm, group)

- comm – handle of communicator
- (OUT) group – handle of corresponding group
Group Construction

- MPI_GROUP_UNION
- MPI_GROUP_INTERSECTION
- MPI_GROUP_DIFFERENCE
- MPI_GROUP_INCL
- MPI_GROUP_EXCL
- MPI_GROUP_RANGE_INCL
- MPI_GROUP_RANGE_EXCL
Group Destruction

MPI_GROUP_FREE(group)

- (IN/OUT) group – handle of group to free; on return contains MPI_GROUP_NULL
Communicator Management

• Operations that access communicators are local and do not require interprocess communication.
Communicator Management

- **MPI_COMM_RANK(comm, rank)**
- `comm` – handle to communicator
- `(OUT)` `rank` – rank of calling process in the group of communicator.
Communicator Construction

- Construction operations are collective operations and involve all of the processes in the group of the source communicator.

- A source communicator is always required to create a new communicator. MPI_COMM_WORLD is used as the base communicator for all other communicators.
Communicator Construction

- MPI_COMM_DUP(comm, newcomm)
- MPI_COMM_CREATE(comm, group, newcomm)
- MPI_COMM_SPLIT(comm, color, key, newcomm)
Communicator Destruction

MPI_COMM_FREE(comm)

- (IN/OUT) comm – handle of communicator to free; result is MPI_COMM_NULL.
- This is a collective operation.
- Pending calls are completed before the communicator is freed.
Inter-communicators

- Process groups are evident in:
  - modular and multi-discipline systems
  - user-level server designs.
- In such cases, communication between groups is necessary.
- In so doing, it is most natural to identify processes by their rank, in a target group.
Semantics

- The group containing the process that initiates the inter-communication is called the **local group**.
- The group containing the target process is called the **remote group**.
- The target process is specified using a *(communicator, rank)* pair.
- The rank of the specified process is relative to the remote group.
Semantics

• Inter-communicator constructors are blocking operations.

• The groups specified in the constructor must be disjoint:
  – For instance, the user controls the ranking of processes when groups are merged into a single communicator so it makes sense to start with disjoint groups.
Semantics

• Summary:
  – The operations for inter-communication are the same as for intra-communication.
  – The same communicator is used for both send and receive.
  – A target process is addressed by its rank in the remote group (for both send and receive).
  – MPI guarantees that communications utilising different communicators do not conflict.
Semantics

• Summary (continued):
  – Inter-communicators can not be used for collective operations.
  – A communicator may provide either inter-communication or intra-communication but not both.
  – An inter-communicator can not be used as the source for the creation of an intra-communicator (e.g. by using MPI_COMM_CREATE).
  – Some communicator access routines may be used to access inter-communicators as well as intra-communicators.
Inter-communicator Accessors

MPI_COMM_TEST_INTER(comm, flag)

- (IN) comm – the communicator to test
- (OUT) flag – the result
Inter-communicator Accessors

- The following intra-communicator accessors may be used with inter-communicators:
  - MPI_COMM_SIZE – returns the size of the local group
  - MPI_COMM_GROUP – returns the local group
  - MPI_COMM_RANK – returns the rank in the local group
Inter-communicator Accessors

• To access information about remote groups:
  – MPI_COMMREMOTE_SIZE(comm, size)
  – MPI_COMMREMOTE_GROUP(comm, group)

• To compare communicators:
  – MPI_COMM_COMPARE may also be used (both communicators must be either inter-communicators or intra-communicators).
  – MPI_IDENT, MPI_CONGRUENT, MPI_SIMILAR, MPI_UNEQUAL
Inter-communicator Constructors

MPI_INTERCOMM_CREATE(local_comm, local_leader, peer_comm, remote_leader, tag, newintercomm)

- (IN)local_comm – local intra-communicator
- (IN)local_leader – rank of leader in local_comm
- (IN)peer_comm – significant at local leader only
- (IN)remote_leader – rank of remote group leader in peer_comm; significant at local leader only
- (IN)tag – a safe tag
- (OUT)newintercomm
Inter-communicator Constructors

MPI_INTERCOMM_CREATE (continued)

• Is collective over the union of the local and remote groups of processes.

• Processes should provide identical local_comm and local_leader arguments.

• This call employs point-to-point communication using peer_comm and tag between the leaders.

• Ensure that no pending communication on peer_comm can interfere with this call.
Inter-communicator Constructors

MPI_INTERCOMM_CREATE (continued)

- Recommend using a duplicate of MPI_COMM_WORLD to avoid communication problems that may arise with other peer communicators.
Inter-communicator Constructors

MPI_INTERCOMM_MERGE(intercomm, high, newintracomm)

- (IN)intercomm – inter-communicator
- (IN)high – boolean; order of groups
- (OUT)newintracomm – new intra-communicator that is a union of the groups of intercomm.

This operation is blocking and collective with respect to the union of the groups of intercomm.
Inter-communicator Constructors

MPI_COMM_DUP(comm, newcomm)

- (IN)comm – existing inter-communicator
- (OUT)newcomm – new inter-communicator
Inter-communicator Destruction

MPI_COMM_FREE(comm)

- (IN/OUT)comm – communicator handle; set to MPI_COMM_NULL
- This is a collective operation.
- Pending calls are completed before the communicator is freed.
Example: Three Group Pipeline
Example: Three Group Pipeline

MPI_Comm local_comm;

MPI_Comm inter_comm_1;
MPI_Comm inter_comm_2;

int membership_key;
int rank;
Example: Three Group Pipeline

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

membership_key = rank % 3;

MPI_Comm_split(MPI_COMM_WORLD, membership_key, rank, local_comm);
Example: Three Group Pipeline

```c
if (membership_key == 0)
{
    MPI_Intercomm_create(
        local_comm,
        0,
        MPI_COMM_WORLD,
        1,
        1,
        &inter_comm_1);
}
```
Example: Three Group Pipeline

```c
else if (membership_key == 1)
{
    MPI_Intercomm_create(local_comm, 0,
                          MPI_COMM_WORLD, 0, 1,
                          &inter_comm_1);

    MPI_Intercomm_create(local_comm, 0,
                          MPI_COMM_WORLD, 2, 2,
                          &inter_comm_2);
}
```
Example: Three Group Pipeline

```c
else if (membership_key == 2)
{
    MPI_Intercomm_create(local_comm, 0,
                         MPI_COMM_WORLD, 1, 2,
                         &inter_comm_1);
}
```
Example: Three Group Pipeline

// Perform task...

// Clean up
switch (membership_key)
{
    case 1:
        MPI_Comm_free(&inter_comm_2);
    case 0:
    case 2:
        MPI_Comm_free(&inter_comm_1);
}

MPI_Finalize();
Example: Three Group Ring

Group 1

Group 2

Group 3
Example: Three Group Ring

```c
if (membership_key == 0)
{
    MPI_Intercomm_create(local_comm, 0,
             MPI_COMM_WORLD, 1, 1,
             &inter_comm_1);

    MPI_Intercomm_create(local_comm, 0,
             MPI_COMM_WORLD, 2, 2,
             &inter_comm_2);
}
```
Example: Three Group Ring

```c
if (membership_key == 1) {
    MPI_Intercomm_create(local_comm, 0,
                         MPI_COMM_WORLD, 0, 1,
                         &inter_comm_1);

    MPI_Intercomm_create(local_comm, 0,
                         MPI_COMM_WORLD, 2, 2,
                         &inter_comm_2);
}
```
Example: Three Group Ring

```c
if (membership_key == 2) {
    MPI_Intercomm_create(local_comm, 0,
                         MPI_COMM_WORLD, 0, 2,
                         &inter_comm_1);

    MPI_Intercomm_create(local_comm, 0,
                         MPI_COMM_WORLD, 1, 2,
                         &inter_comm_2);
}
```
Example: Three Group Ring

// Perform task...

// Clean up
MPI_Comm_free(&inter_comm_1);
MPI_Comm_free(&inter_comm_2);
MPI_Comm_free(&local_comm);

MPI_Finalize();