Collective communication

- Global communication
  - reduction, scan
  - scatter
  - Data movement
    - broadcast, gather

- Synchronization
  - explicit timing coordination

Three broad classes:

- Complete
  - All collective routines block until they are locally
  - No message tags used since all processes must

- Participate
  - Involves coordinated communication within a group
  - of processes
do not interact.

computation to ensure that messages in two phases

A barrier is a simple way to separate two phases of

All processes block until all reach barrier

MPI-BARRIER(COMM)

Used to synchronize execution of a group of processes

Barrier Routine
(op, root, comm)
    MPI_ALLREDUCE(imbpt, output, count, type, root, comm)
    MPI_REDUCE(imbpt, output, count, type, root, comm)
processes.

buffer of single root process or output buffer of all
operation OP, and return combined value into output
the input buffer of each process using a specified
Reduction operations combine the values provided in

Reduction Operations
ATTR:REDUCE sends answer to every process, i.e.

communication context (handle)

comm identifies a group of processes and a root is the process id of root process (integer)

MPI-\text{SUM} etc.

\text{op} is the operation: MPI-\text{MIN}, MPI-\text{MAX}, \text{MPI-\text{SUM}}

\text{type} is the datatype of buffer elements (handle)

and the output buffer on root (integer)

\text{count} is the number of elements in each input buffer

\text{output} is the address of the output buffer

\text{input} is the address of the input buffer
output and input buffers are used on all processes
Initial data

MPI-REDUCE with MPI-SUM root = 1

MPI-ALLREDUCE with MPI-MIN

MPI-REDUCE with MPI-MIN root = 0

A

BC

D
We can compute the dot product $X_Y^T$ using the following code.

We are in the address space of process 0.

- For example, elements $x_{0}, \ldots, x_{n-1}$ are contained in each address space.
- For each vector $p$ of processes ranked $0, \ldots, I-1$, $d/u_{i}$, $x_{j}$ are partitioned across the two vectors.

Example of Reduction
{
    return dot;
    MPI-SUM, 0, MPI.COMM_WORLD)
    MPI-Reduce(LOCAL-dot, dot, 1, MPI-FLOAT,
    local-dot = SERIAL-dot(local-x, local-y, local-n);
    float SERIAL-dot(float x[], float y[], int m) {
        float dot = 0.0;
        float local-dot;
    }
    int local-n
    int i
    int j
    float local-x[]
    float local-y[]

    Serial PARALLEL-dot( )
}
MPI_BCAST(input, count, type, root, comm)

the broadcast.

Processes must know what process is the source of

Note a wild card for the root is not allowed.

data to all other processes.
The same

Broadcast routine implements a one-to-all broadcast

Data Movement Routines
comm identifies a communication context

root is the process id of root process

intype is the datatype of buffer elements

incount is the number of elements in input buffer

receive buffer on other processes

input is the address of the sending buffer on root and
of the processes

• The data is placed in the root's buffer in rank order

(including the root) to the root process.

• This sends a fixed amount of data from all processes

MPI-GATHER(input, inputtype, inputcount, output, outputtype, outputcount)
(handle, used only on root)

- output is the datatype of output buffer elements
- output is the number of elements to be placed in output buffer by each process
- output is the address of the output buffer (used only on root)
- input is the datatype of input buffer elements
- input is the number of elements in input buffer
- input is the address of the input buffer
comm identifies a communication context (handle)

• for all processes

(root is the process id of root process (integer), same)
processors

- Allows aggregate data to be collected on all processors

- The data is placed in the each receive buffer in rank order of the processes

- This sends a fixed amount of data from all processes (including the root) to the all_processes

MPI_ALLGATHER(\text{input}, \text{input_count}, \text{input_type}, \text{output}, \text{output_count}, \text{output_type}, \text{output})
order of the processes

• The data is removed in the root's buffer in rank

• This sends a fixed equal fraction of the data on the root to all processes (including the root).

MPI-SHATTER(inbuf, incount, inype, output, outbuf, count, outputtype, comm)
- `inbuf` is the address of the input buffer on root
- `incount` is the number of elements in input buffer to be sent to each process (integer)
- `intype` is the datatype of input buffer elements (handle)
- `outbuf` is the address of the output buffer
- `outcount` is the number of elements to be received from the root in output buffer by each process (integer)
- `outtype` is the datatype of output buffer elements (handle)
comm identifies a communication context (handle)

(for all processes)

root is the process id of root process (integer, same)
one-to-all broadcast
MPI_BCAST

one-to-all scatter
MPI_SCATTER

one-to-all gather
MPI_GATHER

processes
(tself)

sequence of $d$ sends, one to each process (including

This is the equivalent of each process executing a

buffer of process $j$.

process $i$ and placed in the $i$-th block of the receive

The $j$-th block sent from process $i$ is received by

blocks.

Each process has a buffer containing $d$ equally sized

blocks.

There is also an MPI_ALLOC function.
MPI_Comm free: Releases a communicator.

MPIComm create: An intercommunicator.

MPIInterComm create: Create a new communicator.

MPIComm split: Create a new communicator comprised just a subset of a given group of processes.

MPIComm merge: Create a new communicator comprised just a subset of a given group of processes.

MPIComm world: Create a new communicator comprised just a subset of a given group of processes.

MPI Communicators
the same processes as `comm` but with a new context.

- Creates a new communicator comprising
  `MPI_COMM_DUP(comm, newcomm)`

- A call of the form library uses.

  User of a library routine may not know the tags the
  user of a library routine.

- It is important to ensure that the message tags used
  in the library are distinct from those used in the rest
  of the application.

- Consider an application that calls a library routine
  implementing an array transpose operation.

Creating Communicators
Timing could cause misinterpretation

Composed tasks use same message tags

Errors can occur if two parallel programs are

Sequential Composition of Two Programs
call MPI-Comm-FREE(newcomm, ierr)
| Free new context

call transpose(newcomm, A)
| Pass to Library

call MPI-Comm-DUP(comm, newcomm, ierr)
| Create new context

... 

integer comm, newcomm, ierr

Example of Use of New Communicator
cannot be confused with communications outside

Ensures that communications within this routine

operations

communicator newcomm in all communication
The transpose routine itself will be defined to use the
Processes in the group that is split

Note that this implies global synchronization among the
all groups.

Note that the new communicator has the same name in
increasing communicators ordered by key.

These processes are assigned identities with
creates a new communicator for each unique value of

MPI.COMM_SPLIT(comm, color, key, newcomm)

Available processes can be partitioned into disjoint sets

Partitioning Processes
Example of Partitioned Calls
MPI-Comm-split(comm, color, myid, newcomm);

color = myid % 3;

MPI-Comm-rank(comm, myid);

MPI-Comm-comm, newcomm;

{ 2,5 }

and {0,3,6}, {1,4,7}, and

If comm contains 8 processes, form three

•