Message Passing Interface

Part - II

Dheeraj Bhardwaj
Department of Computer Science & Engineering
Indian Institute of Technology, Delhi – 110016 India
http://www.cse.iitd.ac.in/~dheerajb

Message Passing Interface

Outlines

- Basics of MPI
- How to compile and execute MPI programs?
- MPI library calls used in Example program
- MPI point-to-point communication
- MPI advanced point-to-point communication
- MPI Collective Communication and Computations
- MPI Datatypes
- MPI Communication Modes
- MPI special features

Is MPI Large or Small?

<u>The MPI Message Passing Interface Small or Large</u>

MPI can be small.

One can begin programming with 6 MPI function calls

MPI_INIT	*Initializes MPI*
MPI_COMM_SIZE	*Determines number of processors*
MPI_COMM_RANK	*Determines the label of the calling process*
MPI_SEND	*Sends a message*
MPI_RECV	*Receives a message*
MPI_FINALIZE	*Terminates MPI*

MPI can be large

One can utilize any of 125 functions in MPI.

MPI Blocking Send and Receive

<u>Blocking Send</u>

A typical blocking send looks like

send (*dest, type, address, length*)

<u>Where</u>

- ❖ **<u>dest</u>** is an integer identifier representing the process to receive the message
- ❖ **<u>type</u>** is nonnegative integer that the destination can use to selectively screen messages
- ❖ **(<u>address</u>, <u>length</u>)** describes a contiguous area in memory containing the message to be sent

MPI Blocking Send and Receive

Point-to-Point Communications

The sending and receiving of messages between pairs of processors.

- **BLOCKING SEND:** returns only after the corresponding RECEIVE operation has been issued and the message has been transferred.

  MPI_Send

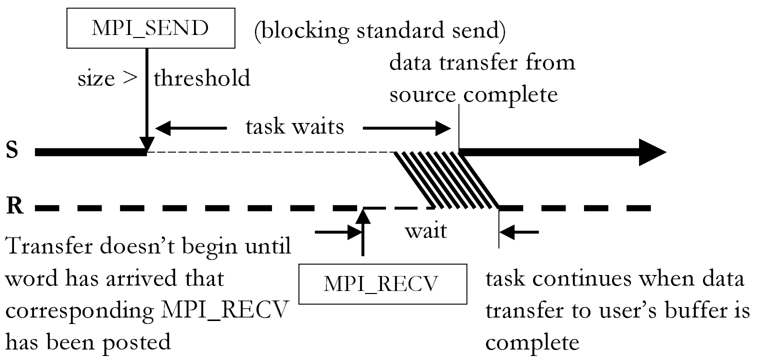
- **BLOCKING RECEIVE:** returns only after the corresponding SEND has been issued and the message has been received.

  MPI_Recv

MPI Blocking Send and Receive

If we are sending a large message, most implementations of blocking send and receive use the following procedure.

S = Sender R = Receiver


MPI_SEND
(blocking standard send)
size > threshold
data transfer from source complete
task waits
S
R
wait
Transfer doesn't begin until word has arrived that corresponding MPI_RECV has been posted
MPI_RECV
task continues when data transfer to user's buffer is complete

MPI Non- Blocking Send and Receive

Non-blocking Receive: does not wait for the message transfer to complete, but immediate returns control back to the calling processor.

MPI_IRecv

C

```
MPI_Isend (buf, count, dtype, dest, tag, comm, request);
MPI_Irecv (buf, count, dtype, dest, tag, comm, request);
```

Fortran

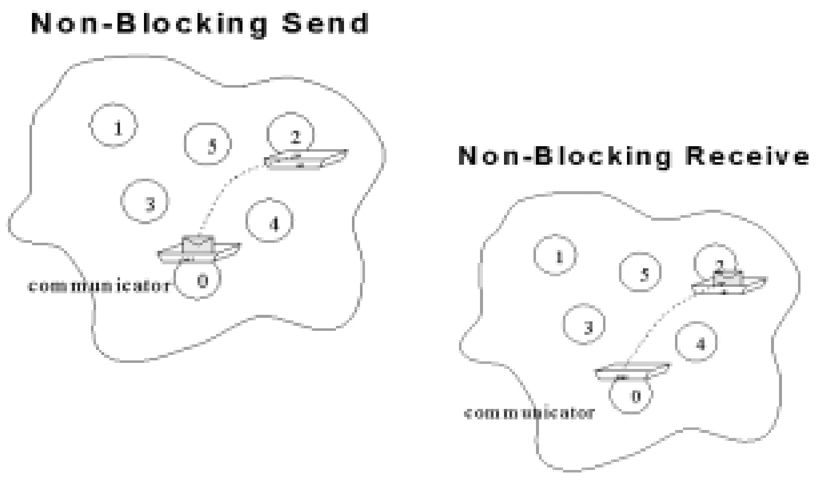
```
MPI_Isend (buf, count, dtype, tag, comm, request, ierror)
MPI_Irecv (buf, count, dtype, source, tag, comm, request, ierror)
```

Non- Blocking Communications

- Separate communication into three phases:
  - Initiate non-blocking communication.
  - Do some work (perhaps involving other communications ?)
  - Wait for non-blocking communication to complete.

Message Passing Interface

MPI Non- Blocking Send and Receive



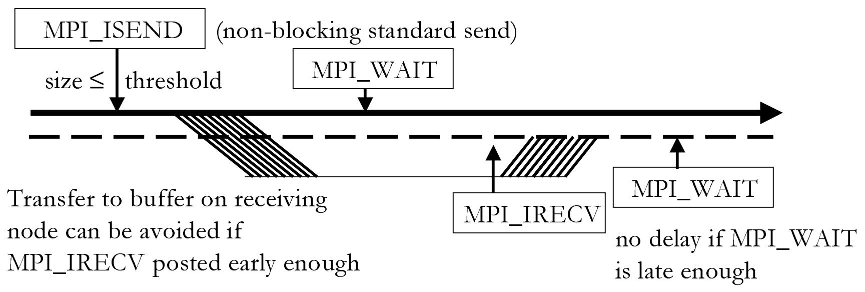
Message Passing Interface

MPI Non- Blocking Send and Receive

If we are sending a small message, most implementations of non-blocking sends and receive use the following procedure. The message can be sent immediately and stored in a buffer on the receiving side.

S = Sender R = Receiver

An MPI-Wait checks to see it a non-blocking operation has completed. In this case, the MPI_Wait on the sending side believes the message has already been received.



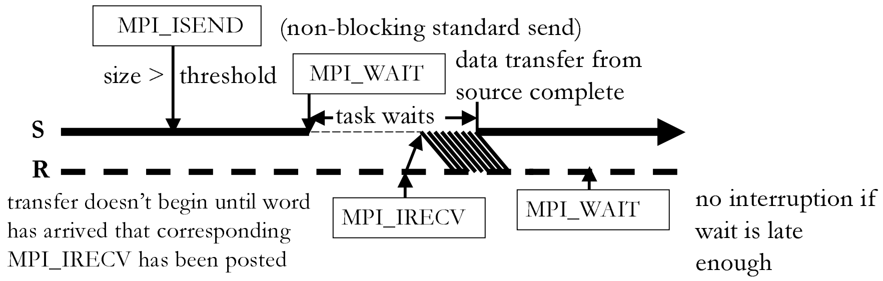
MPI_ISEND
(non-blocking standard send)
size ≤ threshold
MPI_WAIT
Transfer to buffer on receiving node can be avoided if MPI_IRECV posted early enough
MPI_IRECV
MPI_WAIT
no delay if MPI_WAIT is late enough

MPI Non- Blocking Send and Receive

If we are sending a large message, most implementations of non-blocking sends and receive use the following procedure. The send is issued, but the data is not immediately sent. Computation is resumed after the send, but later halted by an MPI_Wait.

S = Sender R = Receiver

An MPI_Wait checks to see it a non-blocking operation has completed. In this case, the MPI_Wait on the sending side sees that the message has not been sent yet.


MPI_ISEND
(non-blocking standard send)
size > threshold
MPI_WAIT
data transfer from source complete
task waits
S
R
transfer doesn't begin until word has arrived that corresponding MPI_IRECV has been posted
MPI_IRECV
MPI_WAIT
no interruption if wait is late enough

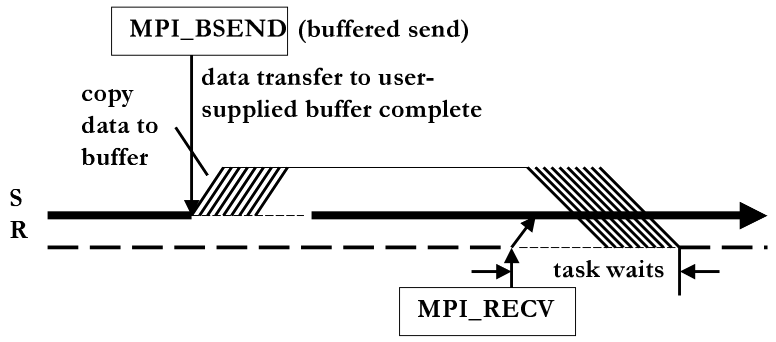
MPI Communication Modes

- **Synchronous mode**
  - The same as standard mode, except the send will not complete until message delivery is guaranteed
- **Buffered mode**
  - Similar to standard mode, but completion is always independent of matching receive, and message may be buffered to ensure this

MPI Buffered Send and Receive

If we the programmer allocate some memory (buffer space) for temporary storage on the sending processor, we can perform a type of non-blocking send.

S = Sender R = Receiver



MPI Communication Modes

Sender mode	Notes
Synchronous send	Only completes when the receive has completed
Buffered send	Always completes (unless an error occurs), irrespective of receiver.
Standard send	Either synchronous or buffered.
Ready send	Always completes (unless an error occurs), irrespective of whether the receive has completed.
Receive	Completes when a message has arrived.

MPI Communication Modes

MPI Sender Modes

OPERATION	MPI CALL
Standard send	MPI_SEND
Synchronous send	MPI_SSEND
Buffered send	MPI_BSEND
Ready send	MPI_RSEND
Receive	MPI_RECV

MPI Datatypes

Message type

- A message contains a number of elements of some particular datatype
- MPI datatypes:
  - Basic types
  - Derived types - Vector, Struct, Others
- Derived types can be built up from basic types
- C types are different from Fortran types

MPI Derived Datatypes

Contiguous Data

- The simplest derived datatype consists of a number of contiguous items of the same datatype
- **C :**

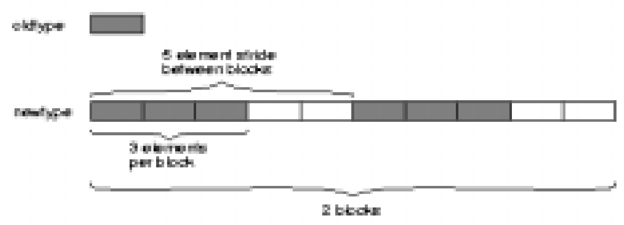
  int MPI_Type_contiguous (int count, MPI_Datatype oldtype,MPI_Datatype *newtype);
- **Fortran :**

  MPI_Type_contiguous (count, oldtype, newtype)

  integer count, oldtype, newtype

MPI Derived Datatypes

Vector Datatype Example



- ❑ `count = 2`
- ❑ `stride = 5`
- ❑ `blocklength = 3`

MPI Derived Datatypes

Constructing a Vector Datatype

- **C** int MPI_Type_vector (int count, int blocklength, int stride,
  MPI_Datatype oldtype, MPI_Datatype *newtype);

- **Fortran**
  MPI_Type_vector (count, blocklength, stride, oldtype, newtype, ierror)

Extent of a Datatype

- **C** int MPI_Type_extent (MPI_Datatype datatype, int *extent);

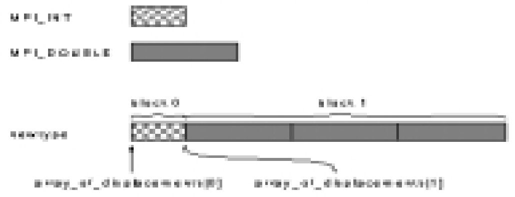
- **Fortran**

  MPI_Type_extent(datatype, extent, ierror)

  integer datatype, extent, ierror

MPI Derived Datatypes

Struct Datatype Example



- ❑ count = 2
- ❑ array_of_blocklengths[0] = 1
- ❑ array_of_types[0] = MPI_INT
- ❑ array_of_blocklengths[1] = 3
- ❑ array_of_types[1] = MPI_DOUBLE

MPI Derived Datatypes

Constructing a Struct Datatype

- C :
  int MPI_Type_struct (int count, int array_of_blocklengths,
      MPI_Aint *array_of_displacements,
      MPI_Datatype *array_of_types,
      MPI_Datatype *newtype);

- Fortran :
  MPI_Type_Struct (count, array_of_blocklengths,
      array_of_displacements, array_of_types, newtype, ierror)

MPI Derived Datatypes

Committing a datatype

- Once a datatype has been constructed, it needs to be committed before it is used.

- This is done using MPI_TYPE_COMMIT

- **C**
  int MPI_Type_Commit (MPI_Datatype *datatype);

- **Fortran**
  MPI_Type_Commit (datatype, ierror)
  integer datatype, ierror

MPI - Using topology

MPI : Support for Regular Decompositions

- Using topology routines

  “MPI_Cart_Create “

  User can define virtual topology

- Why you use the topology routines

  “Simple to use (why not?)

  “Allow MPI implementation to provide low expected contention layout of processes (contention can matter)

  “Remember,contention still matters; a good mapping can reduce contention effects

MPI Persistent Communication

MPI : Nonblocking operations, overlap effective

- Isend, Irecv, Waitall

MPI : Persistent Operations

- Potential saving

  “ Allocation of MPI_Request

- Variation of example

  “ sendinit, recvinit, startall, waitall

  “ startall(recvs), sendrecv/barrier, startall(rsends), waitall

- Vendor implementations are buggy

MPI Collective Communications

<u>Collective Communications Collective Communications</u>

The sending and/or receiving of messages to/from groups of processors. A collective communication implies that all processors need participate in the communication.

- Involves coordinated communication within a group of processes
- No message tags used
- All collective routines block until they are locally complete
- Two broad classes :
  - Data movement routines
  - Global computation routines

MPI Collective Communications

<u>Collective Communication</u>

- Communications involving a group of processes.
- Called by all processes in a communicator.
- Examples:
  - Barrier synchronization.
  - Broadcast, scatter, gather.
  - Global sum, global maximum, etc.jj

MPI Collective Communications

Characteristics of Collective Communication

- Collective action over a communicator
- All processes must communicate
- Synchronization may or may not occur
- All collective operations are blocking.
- No tags.
- Receive buffers must be exactly the right size

MPI Collective Communications

Communication is coordinated among a group of processes

- Group can be constructed "**by hand**" with MPI group-manipulation routines or by using MPI topology-definition routines
- Different communicators are used instead
- No non-blocking collective operations

Collective Communication routines - Three classes

- Synchronization
- Data movement
- Collective computation

MPI Collective Communications

Barrier

A barrier insures that all processor reach a specified location within the code before continuing.

- C:

```
int MPI_Barrier (MPI_Comm comm);
```

- Fortran:

```
MPI_barrier (comm, ierror)
integer comm, ierror
```

MPI Collective Communications

Broadcast

A broadcast sends data from one processor to all other processors.

- **C**:

```
int MPI_Bcast ( void *buffer, int count, MPI_Datatype
                datatype, int root, MPI_Comm comm);
```

- **Fortran**:

```
MPI_bcast (buffer, count, datatype, root, comm,
           ierror)
<type> buffer(*)
integer count, datatype, root, comm, ierror
```

MPI Collective Computations

Global Reduction Operations

- Used to compute a result involving data distributed over a group of processes.
- Examples:
  - Global sum or product
  - Global maximum or minimum
  - Global user-defined operation

MPI Collective Computations

Fortran

```
MPI_Reduce (sendbuf, recvbuf, count, datatype, op,
                root, comm, ierror)
<type> sendbuf (*), recvbuf (*)
integer count, datatype, op, root, comm,
integer ierror
```

C

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

MPI Collective Computations

Collective Computation Operations

MPI_Name	**Operation**
MPI_LAND	Logical and
MPI_LOR	Logical or
MPI_LXOR	Logical exclusive or (xor)
MPI_BAND	Bitwise AND
MPI_BOR	Bitwise OR
MPI_BXOR	Bitwise exclusive OR

MPI Collective Computations

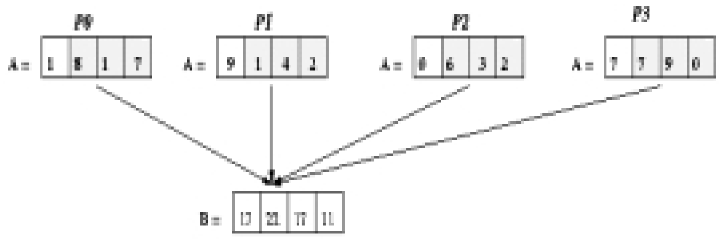
Collective Computation Operation

MPI Name	**Operation**
MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_PROD	Product
MPI_SUM	Sum
MPI_MAXLOC	Maximum and location
MPI_MAXLOC	Maximum and location

MPI Collective Computations

Reduction

A reduction compares or computes using a set of data stored on all processors and saves the final result on one specified processor.

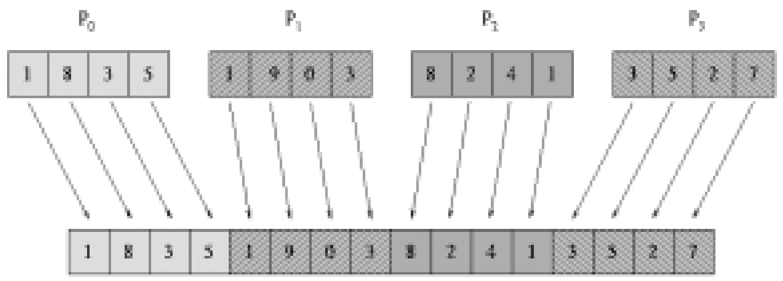


Global Reduction (sum) of an integer array of size 4 on each processor and accumulate the same on processor P1

MPI Collective Communication

Gather

Accumulate onto a single processor, the data that resides on all processors

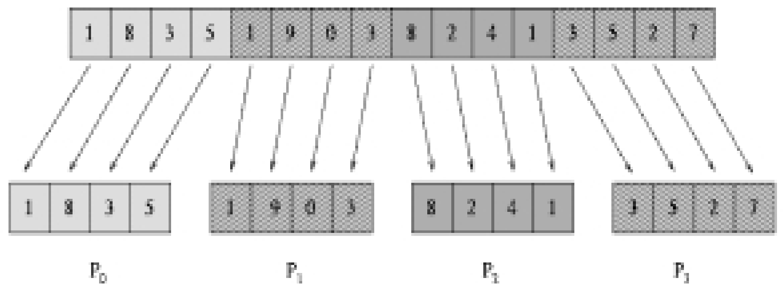


Gather an integer array of size of 4 from each processor

MPI Collective Communication

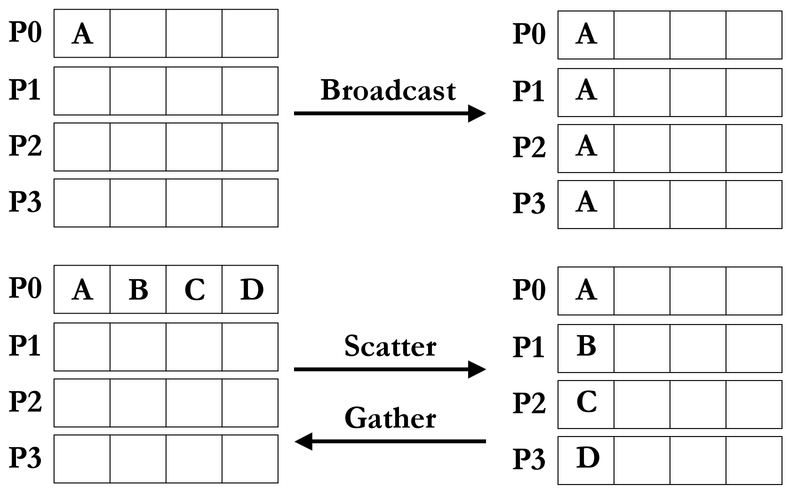
Scatter

Distribute a set of data from one processor to all other processors.



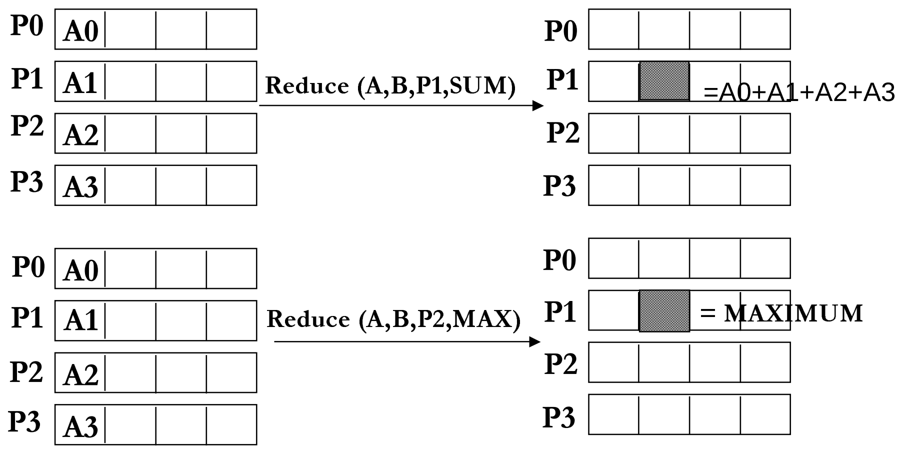
Scatter an integer array of size 16 on 4 processors

MPI Collective Communication



Representation of collective data movement in MPI

MPI Collective Communication



Representation of collective data movement in MPI

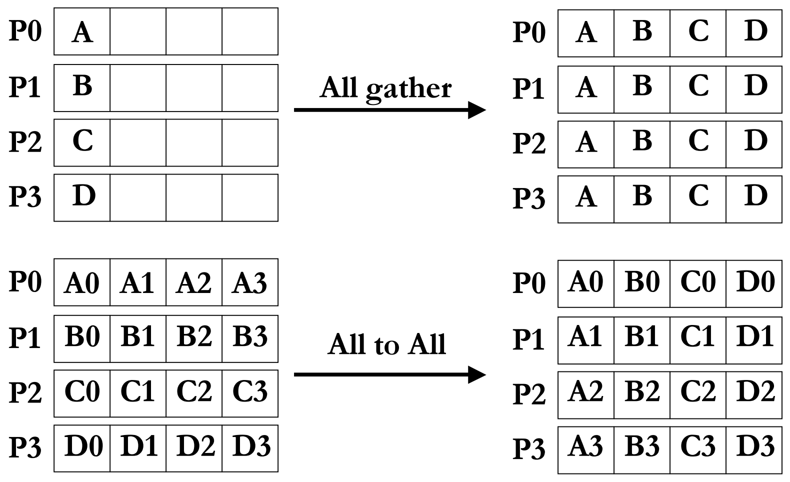
MPI Collective Communications & Computations

Allgather	Allgatherv	Allreduce
Alltoall	Alltoallv	Bcast
Gather	Gatherv	Reduce
Reduce Scatter	Scan	Scatter
Scatterv		

- ❖ All versions deliver results to all participating processes
- ❖ *V* -version allow the chunks to have different non-uniform data sizes (Scatterv, Allgatherv, Gatherv)
- ❖ All reduce, Reduce , ReduceScatter, and Scan take both built-in and user-defined combination functions

Message Passing Interface

MPI Collective Communication



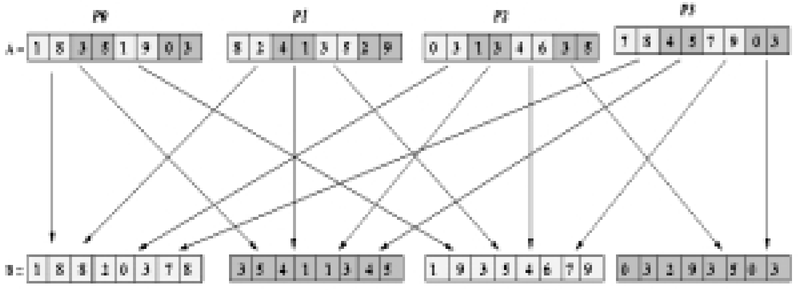
Representation of collective data movement in MPI

Message Passing Interface

MPI Collective Communication

<u>All-to-All</u>

Performs a <u>scatter</u> and <u>gather</u> from all four processors to all other four processors. every processor accumulates the final values



All-to-All operation for an integer array of size 8 on 4 processors

Features of MPI

- **Profiling -** Hooks allow users to intercept MPI calls to install their own tools
- **Environmental**
  - Inquiry
  - Error control
- **Collective**
  - Both built-in and user-defined collective operations
  - Large number of data movements routines
  - Subgroups defined directly or by topology
- **Application-oriented process topologies**
  - Built-in support for grids and graphs (uses groups)

Features of MPI

- **General**
  - Communicators combine context and group for message security
  - Thread safety
- **Point-to-Point communication**
  - Structured buffers and derived datatypes, heterogeneity
  - Modes: normal (blocking and non-blocking), synchronous, ready (to allow access to fast protocols), buffered

Features of MPI

- **Non-message-passing concepts included:**
  - Active messages
  - Threads
- **Non-message-passing concepts not included:**
  - Process management
  - Remote memory transfers
  - Virtual shared memory

Features of MPI

- **<u>Positives</u>**
  - MPI is De-facto standard for message-passing in a box
  - Performance was a high-priority in the design
  - Simplified sending message
  - Rich set of collective functions
  - Do not require any daemon to start application
  - No language binding issues

Features of MPI

Pros

- Best scaling seen in practice
- Simple memory model
- Simple to understand conceptually
- Can send messages using any kind of data
- Not limited to “shared -data”

Features of MPI

Cons

- Debugging is not easy
- Development of production codes is much difficult and time consuming
- Codes may be indeterministic in nature, using asynchronous communication
- Non-contiguous data handling either use derived data types which are error prone or use lots of messages, which is expensive

Features of MPI-2

MPI-2 Techniques – Positives

- Non-blocking collective communications
- One-sided communication
- " put/get/barrier to reduce synchronization points
- Generalized requests (interrupt receive)
- Dynamic Process spawning/deletion operations
- MPI-I/O
- Thread Safety is ensured
- Additional language bindings for Fortran90 /95 and C++

MPI - Performance

Tuning Performance (General techniques)

- Aggregation
- Decomposition
- Load Balancing
- Changing the Algorithm

Tuning Performance

- Performance Techniques
- MPI -Specific tuning
- Pitfalls

References

1. Gropp, W., Lusk, E. and Skjellum, A., Using MPI: Portable Parallel Programming with Message-Passing Interface, The MIT Press, 1999.

1. Pacheco, P. S., Parallel Programming with MPI, Morgan Kaufmann Publishers, Inc, California (1997).
2. Vipin Kumar, Ananth Grama, Anshul Gupta, George Karypis, Introduction to Parallel Computing, Design and Analysis of Algorithms, Redwood City, CA, Benjmann/Cummings (1994).
3. William Gropp, Rusty Lusk, Tuning MPI Applications for Peak Performance, Pittsburgh (1996)