Point to point communication: whom

Name space
- Rank
  - 0..n-1
- Within Group
- Communicator
  - Group + communication context
  - Predefined
    - MPI_COMM_WORLD
    - MPI_COMM_SELF

Designation of source/sink of communication
- (Communicator, rank)
Point to point communication: what

- **What is communicated**
  - Set of objects of a given type

- **Data types**
  - Predefined
    - Fortran: MPI_INTEGER, MPI_REAL, MPI_CHARACTER, etc.
    - C: MPI_CHAR, MPI_SHORT, MPI_INT, MPI_LONG, MPI_FLOAT, etc.
    - MPI_BYTE, MPI_PACKED
  - Derived
    - Mechanism to define new types

- **Send and receive types must match for the program to be correct**
- No type conversion
- **Type representation conversion**

---

Point to point communication: synch.

- **End to end (communication modes)**
  - Buffered
  - Synchronous
  - Standard
  - Ready

- **Local**
  - Blocking call
  - Non-blocking call:
    - Immediate calls
Point to point communication: semantics

- Communication semantics
  - Order: no overtaking
    - between messages that match the same receive, receives that match the same send
    - Between the same pair of processes
  - Progress
  - Fairness: not guaranteed

MPI-1 Interface: communication model

- Point to point sends
  - MPI_Send (buf, count, datatype, dest, tag, comm)
  - MPI_Bsend(buf, count, datatype, dest, tag, comm)
  - MPI_Rsend(buf, count, datatype, dest, tag, comm)
  - MPI_Ssend(buf, count, datatype, dest, tag, comm)
  - MPI_Isend (buf, count, datatype, dest, tag, comm, request)
  - MPI_Ibsend(buf, count, datatype, dest, tag, comm, request)
  - MPI_Issend(buf, count, datatype, dest, tag, comm, request)
  - MPI_Irsend(buf, count, datatype, dest, tag, comm, request)
MPI-1 Interface: communication model

Point to point receive

- MPI_Recv (buf, count, datatype, source, tag, comm, status)
- MPI_Get_count(status, datatype, count)

Example

Do isize=1,4
  msgsize = sizes(isize)
  if (rank .eq. 0) then
    call MPI_send(sndmsg, msgsize, MPI_INTEGER1, dest, rank,
    1           MPI_COMM_WORLD, error)
  endif
  do i=1, NITERS
    call Compute(delay_time1)
    call MPI_Recv(rcvmsg, msgsize, MPI_INTEGER1, MPI_ANY_SOURCE,
    1          MPI_ANY_TAG, MPI_COMM_WORLD, status, error)
    call Compute(delay_time2)
    call MPI_send(sndmsg, msgsize, MPI_INTEGER1, dest, rank,
    1           MPI_COMM_WORLD, error)
  enddo
  if (rank .gt. 0) then
    call MPI_Recv(rcvmsg, msgsize, MPI_INTEGER1, MPI_ANY_SOURCE,
    1          MPI_ANY_TAG, MPI_COMM_WORLD, status, error)
  endif
  enddo

Sizes: 100, 1000, 10000, 100000
All sizes

Do isize=1,4
  msgsize = sizes(isize)
  if (rank .eq. 0) then
    call MPI_send(...)
  endif
  do i=1, NITERS
    call Compute(delay_time1)
    call MPI_Recv(...)
    call Compute(delay_time2)
    call MPI_send(...)
  enddo
  if (rank .gt. 0) then
    call MPI_Recv(...)
  endif
enddo

Short messages

Do isize=1,4
  msgsize = sizes(isize)
  if (rank .eq. 0) then
    call MPI_send(...)
  endif
  do i=1, NITERS
    call Compute(delay_time1)
    call MPI_Recv(...)
    call Compute(delay_time2)
    call MPI_send(...)
  enddo
  if (rank .gt. 0) then
    call MPI_Recv(...)
  endif
enddo
Long messages

Do isize=1,4
  msgsize = sizes(isize)
  if (rank .eq. 0) then
    call MPI_send(...)
  endif
  do i=1, NITERS
    call Compute(delay_time1)
    call MPI_Recv(...)
    call Compute(delay_time2)
    call MPI_send(...)
  enddo
  if (rank .gt. 0) then
    call MPI_Recv(...)
  endif
endo

Message size = 100000

MPI-1 Interface: communication model

- Point to point receive
  - MPI_Probe (source, tag, comm, status)
  - Check for arrival of matching message
**MPI-1 Interface: communication model**

### Point to point receive

- MPI_Irecv (buf, count, datatype, dest, tag, comm, request)
- MPI_Wait(request, status)
- MPI_Test(request, flag, status)
- MPI_Get_count(status, datatype, count)

### Wait/check finalization of Immediate call

- MPI_Waitany (count, array_of_requests, index, status)
- MPI_Waitall (count, array_of_requests, array_of_statuses)
- MPI_Waitsome (incount, array_of_requests, outcount, array_of_indices, array_of_statuses)

- MPI_Testany (count, array_of_requests, index, flag, status)
- MPI_Testall (count, array_of_requests, flag, array_of_statuses)
- MPI_Testsome (incount, array_of_requests, outcount, array_of_indices, array_of_statuses)
MPI-1 Interface: communication model

- Other issues
  - Cancellation
  - Persistent communication requests
  - Send-receive
  - MPI_PROC_NULL
  - Derived data types
    - MPI_TYPE_CONTIGUOUS, MPI_TYPE_VECTOR, MPI_TYPE_INDEXED, MPI_TYPE_STRUCT, ...
    - MPI_TYPE_COMMIT
  - MPI_PACK, MPI_UNPACK

MPI Run Time Library
Implementation issues

- Buffering
  - At sender
    - Fast local response
    - 3 messages protocol
  - At receiver
    - Advance transmission

- Buffer management
  - Static reservation
    - Inefficient
  - Dynamic allocation
    - Overhead

- Protocol
  - Small/large messages

  Data
  Request
  Ack
  Data

- Flow control
  - Avoid overflow of incoming messages
  - Tokens/credit: ~ 2 messages protocol

- Matching
  - Receive to sends
  - Efficiency

```c
MPI_Recv(...)
{
  if (found in library buffer) return

  do {
    get from socket/link
    if (matches user request) ? user buffer
    else ? library buffer
  } until matched request
}
```
Implementation issues

- Number of copies
- Injection
  - system call
    - sockets
  - memory mapped devices
    - Efficient user mode access
    - Resource consumption

Incoming arrival detection
- Interrupt
  - Signal
  - Threads:
    - Influence on OS scheduling
- Poll
Environment variables

- **Mechanism for the user to control the implementation mechanisms**
- **Machine specific: IBM**
  - MP_SHARED_MEMORY  yes/no
  - MP_EAGER_LIMIT  value
  - MP_CSS_INTERRUPT  yes/no
  - MP_EUILIB  us/ip

MPI-1 Interface: communication model

- **Collective synchronization**
  - Called by all processes within a group
    - MPI_Barrier (comm)
Collective communication

- MPI_Bcast (buffer, count, datatype, root, comm)

- MPI_Gather (sendbuf, sendcount, sendtype, recvbuf, recvtype, root, comm)

- MPI_Scatter (sendbuf, sendcount, sendtype, recvbuf, recvtype, root, comm)
Collective communication

- MPI_Allgather (sendbuf, sendcount, sendtype, recvbuf, recvtype, comm)

![Diagram of MPI_Allgather](image)

Collective communication

- MPI_Alltoall (sendbuf, sendcount, sendtype, recvbuf, recvtype, comm)

![Diagram of MPI_Alltoall](image)
**MPI-1 Interface: communication model**

**Collective reductions**

- `MPI_Reduce` (sendbuf, recvbuf, count, datatype, op, root, comm)
  - `MPI_MAX`, `MPI_MIN`, `MPI_SUM`, `MPI_PROD`, ...
- `MPI_Allreduce` (sendbuf, recvbuf, count, datatype, op, comm)
- `MPI_Reduce_scatter` (sendbuf, recvbuf, recvcounts, datatype, op, comm)
- `MPI_Scan` (sendbuf, recvbuf, count, datatype, op, comm)

**Collective implementation issues**

**Based on point to point**

- Tree based communications to minimize
  - # communications
  - Dependence chain
- On SMP
  - First local then remote
Other issues

- Initialization termination
- Time management
- Group and communicator management
  - Definition of sub groups
- Topologies
  - Ease “typical” communication patterns
  - Potential to better match platform
- PMPI: MPI Profiler Interface
  - Ease portable tools development

MPI-1 Interface: process model

- Initialization: declare as part of parallel application
  - MPI_init (argc, argv)
  - MPI_Finalize()
- Time management
  - MPI_Wtime()
  - MPI_Wtick()
  - MPI_Wtime_is_global()
### Name space management

- **Self identification**
  - `MPI_Comm_rank(comm, rank)`
  - `MPI_Group_rank (group, rank)`

- **Name space query**
  - `MPI_Comm_size(comm, size)`
  - `MPI_Group_size (group, size)`

- **Name translation**
  - `MPI_Group_translate (group1, n, ranks1, group2, ranks2)`
  - `MPI_Comm_group (comm, group)`

### Group management

- `MPI_Group_union (group1, group2, newgroup)`
- `MPI_Group_intersection (group1, group2, newgroup)`
- `MPI_Group_incl (group, n, ranks, newgroup)`
- `MPI_Range_incl (group, n, ranks, newgroup)`
- `MPI_Range_excl (group, n, ranks, newgroup)`
- `MPI_Group_free (group)`
 MPI-1 Interface: process model

- **Name space management**
  - **Communicator management**
    - MPI_Comm_dup (comm, newcomm)
    - MPI_Comm_create (comm, group, newcomm)
    - MPI_Group_split (comm, color, key, newgroup)
    - MPI_Group_free (group)

- **Name translation: Topologies**
  - MPI_Cart_create (comm_old, ndims, dims, periods, reorder, newcomm)
  - MPI_Cartdim_get (comm, ndims)
  - MPI_Cart_rank (comm, coords, rank)
  - MPI_Cart_coords (comm, rank, maxdims, coords)
  - MPI_Cart_shift (comm, direction, disp, rank_source, rank_dest)
  - MPI_Cart_sub (comm, remain_dims, newcomm)
  - MPI_Cart_map (comm, ndims, dims, periods, newrank)
  - MPI_Dims_create (nnodes, ndims, dims)
MPI-1 Interface: process model

- Name space management
  
  - Name translation: Topologies (cont.)
    
    - MPI_Graph_create (comm_old, nnodes, index, edges, reorder, comm_graph)
    - MPI_Topo_test (comm, status)
    - MPI_Graphdims_get (comm, nnodes, edges)
    - MPI_Graph_get (comm, maxindex, maxedges, index, edges)
    - MPI_Graph_neibors_count (comm, rank, maxneibor, neibors)
    - MPI_Graph_map (comm, nnodes, index, edges, new_rank)

- and more
  
  - MPI_Group_compare (group1, group2, result)
  - MPI_Comm_compare (comm1, comm2, result)
**MPI-1 Interface: type model**

- **Type definitions**
  - MPI_Type_contiguous (count, oldtype, newtype)
  - MPI_Type_vector (count, blocklength, stride, oldtype, newtype)
  - MPI_Type_hvector (count, blocklength, stride, oldtype, newtype)
  - MPI_Type_indexed (count, array_of_blocklengths, array_of_displacements, oldtype, newtype)
  - MPI_Type_hindexed
  - MPI_Type_struct (count, array_of_blocklengths, array_of_displacements, array_of_types, newtype)

- **Creation / destruction**
  - MPI_Type_commit (datatype)
  - MPI_Type_free (datatype)
**MPI-1 Interface: type model**

- **Type queries**
  - `MPI_Type_extent (datatype, extent)`
  - `MPI_Type_size (datatype, size)`
    - Data bytes
  - `MPI_Type_count (datatype, count)`

---

**Nested MPI + OpenMP**

- MPI specified as thread safe, not all implementations are
- Typical combination
  - OpenMP used for loops between two communications
  - MPI called by the master thread
Nested MPI + OpenMP

msgsize = 10000
...
do i=1,NITERS
  do j=1,n
    if (rank.eq.0) then
      delay_time1=(N-j)/100
    else
      delay_time1=j/100
    endif
    call Compute(delay_time1)
  enddo
  call MPI_Recv(...)
  call Compute(delay_time2)
  call MPI_send(...)
  enddo
  ...
  }

Jesús Labarta, SC, 2002
MPI-1 Interface: comments

- New concept ⇔ lot of new calls
  - # functions / concepts used ??
  - Design by committee ⇒ ∪ proposals

- Implementation cost = f (# man pages)
  - Overhead if just sending array in memory

- Conclusion: MPI-1 was not enough ⇒ MPI-2

Batallitas

- Sweep3D
- Environment variables
- Metacomputing
- Blue Gene
- Bet
  - OpenMP as popular as OpenMP
  - Causa perdida?
MPI-2

- **Functionalities**
  - Clean-up and clarifications
    - handful of small problems
  - Process model
    - Dynamic process creation
  - One-sided communication
    - shared memory
  - MPI-I/O

- **Status**
  - Delayed implementations
  - Performance =

---

MPI-2: Process model

- **Dynamic process creation**

  Binary file
  String of args
  Granularity

  - MPI_Comm_spawn(command, argv, maxprocs, info, root, comm, intecomm, array_of_errcodes)

  Parents + children
  MPI_Get_parent
  +

  Children local

  Collective over
  Wait MPI_INIT

  Resquested
  if not possible ....

  AOB
MPI-2: Process model

- Dynamic process creation

- Client-server
  - `MPI_Open_port(info, port_name)`
  - `MPI_Close_port(port_name)`
  - `MPI_Comm_accept(port_name, info, root, comm, newcomm)`
  - `MPI_Comm_connect(port_name, info, root, comm, newcomm)`

- Name service
  - `MPI_Publish_name(service_name, info, port_name)`
  - `MPI_Unpublish_name(service_name, info, port_name)`
  - `MPI_Lookup_name(service_name, info, port_name)`

  ....familiar
MPI-2: One-Sided Communications

- Influence of non-coherent shared memory programming models
  - CRAY shmem
  - Sympathy for Load/store

- Provide a shared memory interface on distributed machines
  - Separate communication and synchronization
  - Needs:
    ✓ Declare accessible address space
    ✓ Access primitives
    ✓ Consistency management

MPI-2: One-Sided Communications

- Declaration of accessible space
  - Window: region of memory
  - MPI_win_create (base, size, disp_unit, info, comm, win)
  - MPI_win_free (win)
**MPI-2: One-Sided Communications**

### Info on accessible space

- `MPI_Win_get_attr ( win, queried_atrib, value, flag )`
- `MPI_Win_get_group (win, group)`

### Data access

- `MPI_Put (orig_addr, orig_count, orig_datatype, target_rank, target_disp, target_count,target_datatype, win)`
- `MPI_Get (orig_addr, orig_count, orig_datatype, target_rank, target_disp, target_count,target_datatype, win)`

\[
\text{From/to region} \quad \text{*disp_unit+ Window_base} = \text{From/to address}
\]
MPI-2: One-Sided Communications

- Data access (and ... )
  - MPI_Accumulate(orig_addr, orig_count, orig_datatype, target_rank, target_disp, target_count, target_datatype, op, win)

- Consistency
  - Epoch: Time between synchronization calls
    - Memory state within epoch: who knows
    - Memory state consistent at end of epoch

- Epoch synchronization: Two ways
  - Fence
  - Post/start
  - Locks
MPI-2: One-Sided Communications

- Fence synchronization
  - Collective start of new epoch (and end of previous)
    - \texttt{MPI\_Win\_fence ( assert, win)}

MPI-2: One-Sided Communications

- General synchronization
  - What I want to access
    - \texttt{MPI\_Win\_start (group, assert, win)}
  - To whom I offer
    - \texttt{MPI\_Win\_post (group, assert, win)}
    - \texttt{MPI\_Win\_complete (win)}
    - \texttt{MPI\_Win\_wait (win)}
    - \texttt{MPI\_Win\_Test (win)}
MPI-2: One-Sided Communications

Process 0  Process 1  Process 2  Process 3

Post(0)  Post(0,3)  Start(2)

Start(1,2)  Put (1)  Put (2)

Put (2)  Complete  Complete

Wait  Wait

Locks synchronization
- Access epoch with mutual exclusion

- MPI_Win_lock (lock_type, rank, assert, win)
- MPI_Win_unlock (rank, win)
Mixed mode programming

- **Programming model:** MPI + OpenMP
  - Main thread does all communication while in sequential OpenMP part

- **Parallelization:**
  - MPI coarse grain
  - OpenMP fine grain
  - Complexity
    - Seems simple
    - Two programming models to deal with a problem of two granularities
  - Performance
    - Possibly conflicting approaches