MPI

Jesús Labarta

- MPI 1
  - Point to point
  - Collectives
  - Group management
  - Topologies

- MPI 2
  - Process management
  - One-sided
  - I/O

- Some references
  - http://www-unix.mcs.anl.gov/mpi/
Distributed memory: @ space

```c
#include 

int main ()
{
    my_part = f( who am I)
    Compute my part
    Communicate if needed
}
```

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, ...
    - C: MPI_CHAR, MPI_SHORT, MPI_INT, MPI_LONG, MPI_FLOAT, ...
    - 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

Jesús Labarta, MP, 2008
**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)

- **Rank within group**
- **Identifier for later enquiry**
- **Attribute for receive selection**
- **Context**
- **Mode**
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

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

Sizes: 100, 1000, 10000, 100000
All sizes

\begin{verbatim}
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
\end{verbatim}

State

MPI calls

Computation phase

Message size

Jesús Labarta, MP, 2008

Short messages

\begin{verbatim}
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
\end{verbatim}

State

MPI calls

Computation phase

Message size = 100

Jesús Labarta, MP, 2008
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
enddo

Message size = 100000

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, status)
  - MPI_Get_count(status, datatype, count)

Finished?
MPI-1 Interface: communication model

- **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)

- **Check receive machings**
  - MPI_Iprobe (source, tag, comm, flag, status)
    - Non blocking check for arrival of matching message

  - MPI_Probe (source, tag, comm, status)
    - Blocking check for arrival of matching message
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

User() {
  send() {
    Socket
  }
}
recv() {
  Socket
}
User() {
  send() {
    Socket
  }
}
recv() {
  Socket
}
Implementation issues

- Buffering
  - At sender
    - Fast local response
  - At receiver
    - Advance transmission

- Protocol
  - Small/large messages

- Buffer management
  - Amount
  - Static reservation/Dynamic allocation
    - Efficiency/Overhead

Flow control
- Avoid overflow of incoming messages
- Tokens/credit

Matching
- Search overhead

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

- Start of request (isend)
  - immediate
  - At the next blocking call
    ✓ i.e. allow for sequence of requests

Implementation issues

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

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

![Diagram](image1)

Jesús Labarta, MP, 2008
Implementation issues

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

Collective communication
- MPI_Bcast (buffer, count, datatype, root, comm)
**MPI-1 Interface: communication model**

- **Collective communication**
  - MPI\_Gather (sendbuf, sendcount, sendtype, recvbuf, recvtype, root, comm)
  - MPI\_Scatter (sendbuf, sendcount, sendtype, recvbuf, recvtype, root, comm)

---

**MPI-1 Interface: communication model**

- **Collective communication**
  - MPI\_Allgather (sendbuf, sendcount, sendtype, recvbuf, recvtype, comm)
MPI-1 Interface: communication model

- **Collective communication**
  - `MPI_Alltoall (sendbuf, sendcount, sendtype, recvbuf, recvtype, comm)`

![Diagram of data distribution and processor connections involving collective communication]

- **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
  - Partitioned message
    - # concurrent network injection links
    - Pipelining
  - On SMP
    - Local and remote

MPI collectives: Broadcast

- Drain/sink bandwidth

- 2 broadcasts, 4MB
  - Heavy cost of initial phase
  - Bubbles!!
  - Fraction of available bandwidth

- Possibility of local network
  - Within Bladecenter
  - Low cost
MPI collectives: Alltoall

- 1 AlltoAll, 4MB
  - Contention?
  - Send to self!!
  - Fraction of available bandwidth

MPI collectives: AllReduce

- 2 allreduces, 4MB
  - Send to self!!
  - Contention!!
MPI collectsives: AllReduce

- 1 AllReduce
  - Reality vs Dimemas?

Causes
- Contention?

Contentation @ collectsives

- P2p transfers within alltoall
- MPI call cost in ns/byte
  - Real vs
  - Dimemas vs
  - Dimemas + Venus

Jesús Labarta, MP, 2008
Contention @ collectives

- **P2p transfers within alltoall**
  ![Diagram](image)

- **MPI call cost in ns/byte**
  - Real vs
  - Dimemas vs
  - Dimemas + Venus

  ![Diagram](image)

**Impact of eager limit**

- 24 concurrent transfers
- 16 concurrent transfers
- 12 concurrent transfers
- 11 concurrent transfers
- 10 concurrent transfers
- 8 concurrent transfers

*Jesús Labarta, MP, 2008*
Progress issues in point to point

- Unexpected behavior
  - IFS on E10000 (MPICH)*
  - IBM SP (@KTH)

* Courtesy FECIT

Progress issues in point to point

- MPICH – GM
  - Early irecv
  - Isend followed by computation
  - Periodic tests for end of recv
  - No actual transfer till end of sender computation 😐

End of computation
Periodic tests for end of recv
Continuous tests when no more computation

Jesús Labarta, MP, 2008
Progress issue in all2all

- Some processes get stuck in the All2all while others go into the next computation phase.

- When the ones that got out of the all2all call MPI again the stalled ones can proceed.... but they now have to perform the computation.

- Actual cause:

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()

---

MPI-1 Interface: process model

- **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)
MPI-1 Interface: process model

- Name space management
  - Group management
    - MPI_Group_union (group1, group2, newgroup)
    - MPI_Group_intersection (group1, group2, newgroup)
    - MPI_Group_incl (group, n, ranks, newgroup)
    - MPI_Group_free (group)

- Communicator management
  - MPI_Comm_dup (comm, newcomm)
  - MPI_Comm_create (comm, group, newcomm)
  - MPI_Group_split (comm, color, key, newgroup)
  - MPI_Group_free (group)
MPI-1 Interface: process model

- Name space management

  - 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_neighbors_count (comm, rank, maxneighbor, neighbors)
    - MPI_Graph_map (comm, nnodes, index, edges, new_rank)
MPI-1 Interface: process model

- Name space management
  - 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_hindexed
  - MPI_Type_struct (count, array_of_blocklengths, array_of_displacements, array_of_types, newtype)
MPI-1 Interface: type model

- **Creation / destruction**
  - `MPI_Type_commit (datatype)`
  - `MPI_Type_free (datatype)`

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

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

C$OMP PARALLEL DO SCHEDULE(GUIDED)
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
...

State
MPI calls
Computation phase
Parallel functions
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
  - Process model
  - One-sided communication
  - 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)
  
  Requested
  
  if not possible ....

  AOB

  Parents + children
  
  MPI_Get_parent
  
  Children local

Collective over
Wait MPI_INIT
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
    - Memory segment bytes
    - With some structure
    - 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)`

From/to region

\[*disp\_unit+ Window\_base = 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+1 ways
  - Fence
  - Post/start
  - Locks
MPI-2: One-Sided Communications

- Fence synchronization
  - Collective start of new epoch (and end of previous)
  - `MPI_Win_fence (assert, win)`

- General synchronization
  - What I want to access
  - `MPI_Win_start (group, assert, win)`
  - `MPI_Win_complete (win)`
  - To whom I offer
  - `MPI_Win_post (group, assert, win)`
  - `MPI_Win_wait (win)`
  - `MPI_Win_Test (win)`
MPI-2: One-Sided Communications

- **Locks synchronization**
  - Access epoch with mutual exclusion
  - 
    ```
    MPI_LOCK_EXCLUSIVE
    MPI_LOCK_SHARED
    ```
  - 
    ```
    MPI_Win_lock (lock_type, rank, assert, win)
    ```
  - 
    ```
    MPI_Win_unlock (rank, win)
    ```

---

Jesús Labarta, MP, 2008