MPI: a message passing parallel programming model

Eduard Ayguadé
Computer Sciences Department Associate Director (BSC)
Professor of the Computer Architecture Department (UPC)

- Environment
  - MPI_INIT
  - MPI_COMM_SIZE
  - MPI_COMM_RANK
  - MPI_FINALIZE
  - MPI_ABORT

- Point-to-point communication

- Collective communication

- Group management
**MPI_INIT**

**Usage**
- `int MPI_Init( int* argc_ptr, /* in */ char** argv_ptr[] ); /* in */`

**Description**
- Initialize MPI
- All MPI programs must call this routine once and only once before any other MPI routines

**MPI_FINALIZE**

**Usage**
- `int MPI_Finalize (void);`

**Description**
- Terminates all MPI processing
- Make sure this routine is the last MPI call
- All pending communications involving a process have completed before the process calls MPI_FINALIZE
**MPI_COMM_SIZE**

**Usage**
- `int MPI_Comm_size( MPI_Comm comm, /* in */
  int* size ); /* out */`

**Description**
- Return the number of processes (`size`) in the group associated with a communicator `comm`.
- **Communicator**
  - Context for a communication operation
  - Messages are always received within the context they were sent
  - Messages sent in different contexts do not interfere
  - `MPI_COMM_WORLD`
- **Process group**
  - Set of processes that share a communication context

**MPI_COMM_RANK**

**Usage**
- `int MPI_Comm_rank ( MPI_Comm comm, /* in */
  int* rank ); /* out */`

**Description**
- Returns the identifier of the local process in the group associated with a communicator `comm`.
- The identifier (`rank`) of the process is in the range from 0 … `size - 1`
Usage

- int MPI_Abort( MPI_Comm comm, /* in */
  int errorcode ); /* in */

Description

- Forces all processes of an MPI job to terminate

#include "mpi.h"

int rank;
int nproc;

int main( int argc, char* argv[] ) {
  MPI_Init( &argc, &argv );
  MPI_Comm_size( MPI_COMM_WORLD, &nproc );
  MPI_Comm_rank( MPI_COMM_WORLD, &rank);

  /* Nothing to do */
  MPI_Finalize();
}
**MPI: Point-to-point communication**

**Blocking**
- Return from the procedure indicates the user is allowed to reuse resources specified in the call

**Non-blocking**
- The procedure may return before the operation completes, and before the user is allowed to reuse resources specified in the call

**List of routines:**
- MPI_SEND
- MPI_RECV
- MPI_ISEND
- MPI_IRECV
- MPI_WAIT
- MPI_GET_COUNT

---

### MPI_SEND

**Usage**
- `int MPI_Send( void* buf, /* in */
  int count, /* in */
  MPI_Datatype datatype, /* in */
  int destination, /* in */
  int tag, /* in */
  MPI_Comm comm ); /* in */`

**Description**
- Performs a blocking send operation
- The message can be received by either MPI_RECV or MPI_IRECV
- Message envelope
  - Information used to distinguish messages and selectively receive them
  - `<destination, tag, comm>`
**Usage**
- int MPI_Recv( void* buf, /* out */
  int count, /* in */
  MPI_Datatype datatype, /* in */
  int source, /* in */
  int tag, /* in */
  MPI_Comm comm, /* in */
  MPI_Status* status ); /* out */

**Description**
- Performs a blocking receive operation
- The message received must be less than or equal to the length of the receive buffer buf
- MPI_RECV can receive a message sent by either MPI_SEND or MPI_ISEND
- Message envelope: <source, tag, comm>
**MPI data types**

- **MPI_Datatype** can be one of the following:
  - MPI_CHAR
  - MPI_SHORT
  - MPI_INT
  - MPI_LONG
  - MPI_UNSIGNED_CHAR
  - MPI_UNSIGNED_SHORT
  - MPI_UNSIGNED
  - MPI_UNSIGNED_LONG
  - MPI_FLOAT
  - MPI_DOUBLE
  - MPI_LONG_DOUBLE
  - MPI_BYTE
  - MPI_PACKED

**MPI_ISEND**

- **Usage**
  ```
  int MPI_Isend( void* buf, /* in */
  int count, /* in */
  MPI_Datatype datatype, /* in */
  int dest, /* in */
  int tag, /* in */
  MPI_Comm comm, /* in */
  MPI_Request* request ); /* out */
  ```

- **Description**
  - Performs a non-blocking send operation
  - `request` is an identifier for later enquiry with `MPI_WAIT` or `MPI_TEST`
  - The send buffer `buf` may not be modified until the request has been completed by `MPI_WAIT` or `MPI_TEST`
  - The message can be received by either `MPI_RECV` or `MPI_IRECV`
### MPI_Irecv

**Usage**

```c
int MPI_Irecv( void* buf, /* out */
    int count, /* in */
    MPI_Datatype datatype, /* in */
    int source, /* in */
    int tag, /* in */
    MPI_Comm comm, /* in */
    MPI_Request* request ); /* out */
```

**Description**

- Performs a non-blocking receive operation
- Do not access any part of the receive buffer `buf` until the receive is completed by `MPI_WAIT` or `MPI_TEST`
- The message received must be less than or equal to the length of the receive buffer `buf`
- `MPI_Irecv` can receive a message sent by either `MPI_SEND` or `MPI_Isend`

### MPI_Wait

**Usage**

```c
int MPI_Wait( MPI_Request* request, /* inout */
    MPI_Status* status ); /* out */
```

**Description**

- Waits for a non-blocking operation to complete, with identifier stored in `request`
- Information on the completed operation is found in `status`
- If wildcards (`MPI_ANY_SOURCE`, `MPI_ANY_TAG`) were used by the receive for either the source or tag, the actual source and tag can be retrieved from `status`:
  - `MPI_SOURCE`
  - `MPI_TAG`
MPI: Non-blocking operations

**Usage**

- int MPI_Get_count( MPI_Status status, /* in */
  MPI_Datatype datatype, /* in */
  int* count ); /* out */

**Description**

- Returns the number of elements in a message (indicated by status)
- The datatype argument and the argument provided by the call that set the status variable should match
**MPI: Collective communication**

- **Collective**
  - If all processes in a process group need to invoke the procedure

- **List of routines:**
  - MPI_BCAST
  - MPI_SCATTER
  - MPI_SCATTERV
  - MPI_GATHER
  - MPI_GATHERV
  - MPI_ALLGATHER
  - MPI_ALLGATHERV
  - MPI_ALLTOALL

---

**MPI_BCAST**

- **Usage**
  - int MPI_Bcast( void* buffer, /* inout */
    int count, /* in */
    MPI_Datatype datatype, /* in */
    int root, /* in */
    MPI_Comm comm); /* in */

- **Description**
  - Broadcasts a message from root to all processes in communicator comm
  - The type signature of count and datatype on any process must be equal to the type signature of count and datatype at the root
```c
#include "mpi.h"

int main(int argc, char* argv[]) {
    int rank;
    int ibuf;

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

    if(rank == 0) ibuf = 12345;
    else ibuf = 0;
    MPI_Bcast(&ibuf, 1, MPI_INTEGER, 0, MPI_COMM_WORLD);
    printf("ibuf = %d\n", ibuf);

    MPI_Finalize();
}
```

---

**MPI_SCATTER**

- **Usage**
  - `int MPI_Scatter( void* sendbuf, /* in */
    int sendcount, /* in */
    MPI_Datatype sendtype, /* in */
    void* recvbuf, /* out */
    int recvcount, /* in */
    MPI_Datatype recvtype, /* in */
    int root, /* in */
    MPI_Comm comm); /* in */`

- **Description**
  - Distribute individual messages from root to each process in communicator
  - Inverse operation to `MPI_GATHER`
**MPI_SCATTER**

- **Usage**
  - `int MPI_Scatter( void* sendbuf, /* in */
    int sendcount, /* in */
    MPI_Datatype sendtype, /* in */
    void* recvbuf, /* out */
    int recvcount, /* in */
    MPI_Datatype recvtype, /* in */
    int root, /* in */
    MPI_Comm comm ); /* in */

- **Description**
  - Collects individual messages from each process in communicator `comm` to the `root` process and store them in rank order.

---

**MPI_GATHER**

- **Usage**
  - `int MPI_Gather( void* sendbuf, /* in */
    int sendcount, /* in */
    MPI_Datatype sendtype, /* in */
    void* recvbuf, /* out */
    int recvcount, /* in */
    MPI_Datatype recvtype, /* in */
    int root, /* in */
    MPI_Comm comm ); /* in */

- **Description**
  - Collects individual messages from each process in communicator `comm` to the root process and store them in rank order.
**Usage**

- ```c
   int MPI_Allgather( void* sendbuf, /* in */
   int sendcount, /* in */
   MPI_Datatype sendtype, /* in */
   void* recvbuf, /* out */
   int recvcount, /* in */
   MPI_Datatype recvtype, /* in */
   MPI_Comm comm); /* in */
```

**Description**

- Gathers individual messages from each process in communicator `comm` and distributes the resulting message to each process.
- Similar to `MPI_GATHER` except that all processes receive the result.
**Usage**

- `int MPI_Alltoall( void* sendbuf, /* in */
  int sendcount, /* in */
  MPI_Datatype sendtype, /* in */
  void* recvbuf, /* out */
  int recvcount, /* in */
  MPI_Datatype recvtype, /* in */
  MPI_Comm comm); /* in */`

**Description**

- Sends a distinct message from each process to every other process.
- The j-th block of data sent from process i is received by process j and placed in the i-th block of the buffer recvbuf.
Usage

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

Description

Applies a reduction operation to the vector `sendbuf` over the set of processes specified by communicator `comm` and places the result in `recvbuf` on `root`.
MPI_REDUCE (Cont’d)

- **Description (Cont’d)**
  - Both the input and output buffers have the same number of elements with the same type
  - Users may define their own operations or use the predefined operations provided by MPI

- **Predefined operations**
  - MPI_SUM, MPI_PROD
  - MPI_MAX, MPI_MIN
  - MPI_MAXLOC, MPI_MINLOC
  - MPI_LAND, MPI_LOR, MPI_LXOR
  - MPI_BAND, MPI_BOR, MPI_BXOR

**MPI_REDUCE for scalars**
Exercise: PI program

Sequential version:

```c
static long num_steps = 100000;
double step;

void main ()
{
    int i;
    double x, pi, sum = 0.0;

    step = 1.0/(double) num_steps;

    for (i=1;i<= num_steps; i++)
    {
        x = (i-0.5)*step;
        sum = sum + 4.0/(1.0+x*x);
    }

    pi = step * sum;
}
```
#include <mpi.h>

void main (int argc, char *argv[]) {
    int i, my_id, numprocs;
    double x, pi, step, sum = 0.0;

    step = 1.0/(double) num_steps;
    MPI_Init(&argc, &argv);
    MPI_Comm_Rank(MPI_COMM_WORLD, &my_id);
    MPI_Comm_Size(MPI_COMM_WORLD, &numprocs);
    my_steps = num_steps/numprocs;

    for (i=my_id*my_steps; i<(my_id+1)*my_steps; i++) {
        x = (i+0.5)*step;
        sum += 4.0/(1.0+x*x);
    }
    sum *= step;
    MPI_Reduce(&sum, &pi, 1, MPI_DOUBLE,
                MPI_SUM, 0, MPI_COMM_WORLD);
    MPI_Finalize();
}

- **Usage**
  - int MPI_Barrier(MPI_Comm comm);/* in */

- **Description**
  - Blocks each process in communicator comm until all processes have called it