MPI – History and Basic Concepts

Erwin Laure
Director PDC-HPC

1

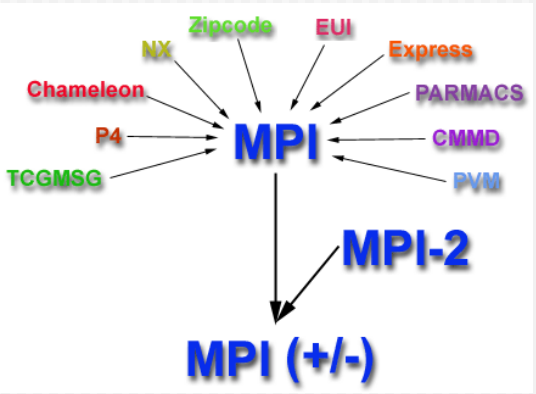
What is MPI

- **M P I** = **M**essage **P**assing **Interface**
- MPI is not an **implementation** – it is a **specification**
  - Specifies the interface of the library
- Interface specifications have been defined for C/C++ and Fortran programs.
- Commonly used implementations of MPI:
  - MPICH (Argonne)
  - MVAPICH
  - OpenMPI
  - Vendor specific
    - Cray
    - Platform
    - IBM

2

MPI History

- Many different message passing implementations in the late 80s
  - Very difficult to port an application to another platform, sometimes even between two generations of the same platform
- 1992-1994: community process to standardize MPI
- 1996: MPI-2
- 201?: MPI-3



3

Reasons for MPI

- **Standardization** - MPI is the only message passing library which can be considered a standard. It is supported on virtually all HPC platforms. Practically, it has replaced all previous message passing libraries.
- **Portability** - There is no need to modify your source code when you port your application to a different platform that supports (and is compliant with) the MPI standard.
- **Performance Opportunities** - Vendor implementations should be able to exploit native hardware features to optimize performance.
- **Functionality** - 128 routines are defined in MPI-1 alone – some 333 in MPI-2
- **Availability** - A variety of implementations are available, both vendor and public domain.

4

Main MPI Concepts

5

A basic MP library

send(address, length, destination, tag)

- **address**: memory location signifying the beginning of the buffer containing the data to be sent,
- **length**: is the length in bytes of the message,
- **destination**: is the receiving process identifier
- **tag**: arbitrary integer to restrict receipt of message

recv (address, maxlen, source, tag, actlen)

6

Message Buffers

- (**address, length**) is insufficient in case of non-contiguous data and the need of data conversion
- MPI introduces datatypes
  - Basic datatypes predefined (MPI_INT, MPI_DOUBLE, …)
  - User can define own (non-contiguous) data types
- A message buffer in MPI is described as

```
(buf, count, datatype)
```

7

MPI Basic Datatypes (Fortran)

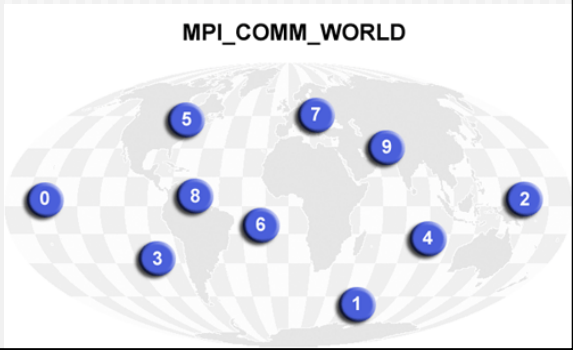
MPI Datatype	Fortran Datatype
MPI_INTEGER	INTEGER
MPI_REAL	REAL
MPI_DOUBLE_PRECISION	DOUBLE_PRECISION
MPI_COMPLEX	COMPLEX
MPI_LOGICAL	LOGICAL
MPI_CHARACTER	CHARACTER(1)
MPI_BYTE	
MPI_PACKED	

Note: the names of the MPI C datatypes are slightly different

8

Processes and Communicators

- Processes belong to **groups**
- Processes within a group are identified with their **rank**
  - A group of n processes has ranks 0 … n-1
- MPI uses objects called **communicators** and groups to define which collection of processes may communicate with each other
  - `MPI_COMM_WORLD` is the default communicator covering all of the original MPI processes



Why Communicators?

- How to chose safe (unique) tags when writing a library? I.e. how to avoid a message being picked up by the wrong receiver?
- Collective operations (broadcast, reductions) can be easily defined over subgroups by using communicators

10

Note: Processes vs. Processors

- MPI defines **processes**, it does not specify how these processes are mapped to physical **processors/cores**
- The mapping of processes to processors/cores is done at program start and dependent on the startup mechanism available on a certain resource – more about that later on.

11

Send/Receive in MPI

```
MPI_Send (buf, count, datatype, dest, tag, comm)
```

- `(buf, count, datatype)` describes the data to be sent
- `Dest` is the rank of the destination in the group associated with communicator `comm`
- `tag` is an identifier of the message
- `comm` identifies a group of processes

```
MPI_Recv (buf, count, datatype, source, tag,
          comm, status)
```

- `status` provides information on the message received, including source, tag, and count

12

Recap: Basic MPI Concepts

- Message **buffers** described by address, data type, and count
- Processes identified by their **ranks**
- **Communicators** identifying communication contexts/ groups

13

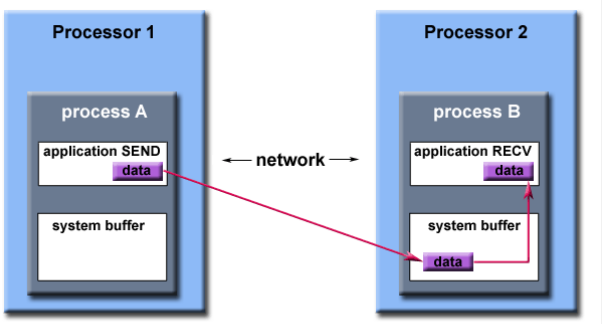
MPI has over 300 functions …

- How many years do I have to study before I can use it?
- In fact, you will hardly ever use most of the MPI functions
- 6 functions are sufficient for simple programs:
  - `MPI_Init` – to initialize the MPI environment
  - `MPI_Comm_Size` – to know the number of processes
  - `MPI_Comm_Rank` – to know the rank of the calling process
  - `MPI_Send` – to send a message
  - `MPI_Recv` – to receive a message
  - `MPI_Finalize` – to exit in a clean way

14

What is not specified

- Certain aspects are not specified in the MPI standard but left as implementation detail:
  - Process startup (how to start an MPI program)
    - All what happens before `MPI_Init` is executed
  - Richer error codes are allowed
  - Message buffering

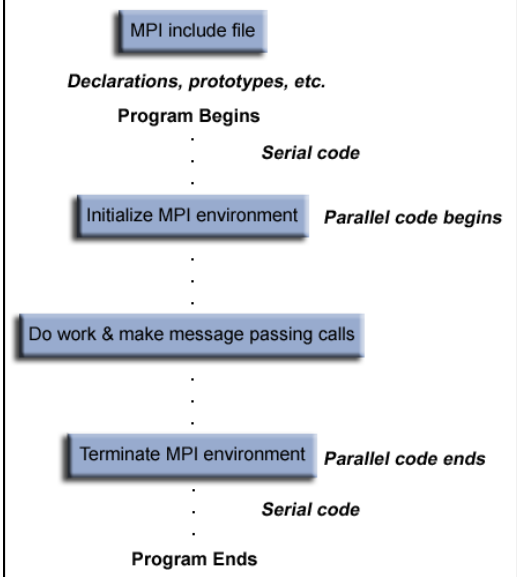


Path of a message buffered at the receiving process

A first MPI Program

16

MPI Program Structure




```
#include "mpi.h"

rc = MPI_Init(&argc,&argv);

MPI_Comm_size(MPI_COMM_WORLD,&
numtasks);

MPI_Comm_rank(MPI_COMM_WORLD,&
rank);

MPI_Finalize();
```

17

Format of MPI Routines

- C Binding:
  - `rc = MPI_Xxxxx`(parameter, ...)
  - Example:`rc = MPI_Send(&buf,count,type,dest,tag,comm)`
  - Error code: Returned as "`rc`". `MPI_SUCCESS` if successful
- Fortran Binding
  - `call mpi_xxxxx(parameter,..., ierr)`
  - Example: `CALL MPI_SEND(buf,count,type,dest,tag,comm,ierr)`
  - Error code: Returned as "`ierr`" parameter. `MPI_SUCCESS` if successful

18

Example: Hello, World (C)

```
#include "mpi.h"
#include <stdio.h>

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

rc = MPI_Init(&argc,&argv);
if (rc != MPI_SUCCESS) {
  printf ("Error starting MPI program. Terminating.\n");
  MPI_Abort(MPI_COMM_WORLD, rc);
  }

MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
printf ("Hello, World from rank %d out of %d\n", rank, numtasks);
MPI_Finalize();
}
```

19

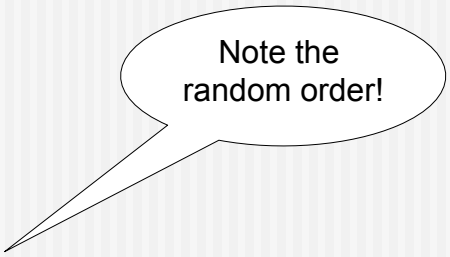
Example: Hello, World (Fortran)

```
program simple
include 'mpif.h'

integer numtasks, rank, ierr, rc

call MPI_INIT(ierr)
if (ierr .ne. MPI_SUCCESS) then
   print *,'Error starting MPI program. Terminating.'
   call MPI_ABORT(MPI_COMM_WORLD, rc, ierr)
end if

call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, numtasks, ierr)
print *, 'Hello, World from rank ',rank, ' out of=',numtasks

call MPI_FINALIZE(ierr)

end
```

20

Sample Output (24 processes)

```
Hello, World from rank 9 out of 24
Hello, World from rank 17 out of 24
Hello, World from rank 13 out of 24
Hello, World from rank 7 out of 24
Hello, World from rank 11 out of 24
Hello, World from rank 14 out of 24
Hello, World from rank 16 out of 24
Hello, World from rank 4 out of 24
Hello, World from rank 15 out of 24
Hello, World from rank 3 out of 24
Hello, World from rank 23 out of 24
Hello, World from rank 10 out of 24
Hello, World from rank 5 out of 24
Hello, World from rank 12 out of 24
Hello, World from rank 2 out of 24
Hello, World from rank 19 out of 24
Hello, World from rank 21 out of 24
Hello, World from rank 8 out of 24
Hello, World from rank 18 out of 24
Hello, World from rank 1 out of 24
Hello, World from rank 6 out of 24
Hello, World from rank 22 out of 24
Hello, World from rank 20 out of 24
Hello, World from rank 0 out of 24
```



21

How to launch MPI Programs?

- Not specified by MPI standard
- Many implementations use `mpirun –np X`
  - Hostfile used to specify processes/hardware mapping
- MPI standard proposes, but does not mandate, a common `mpiexec` syntax/semantics, similar to `mpirun`
- Cray uses `aprun –n x`

22

Summary

- MPI Basics
  - Message buffers
  - Processes and communicators
  - Structure of MPI programs
  - Implementation specific features
- To find out the exact syntax of certain commands:
  - On Lindgren use `> man MPI_xxx`
  - Look up Web resources

23