Introduction to MPI

What is Message Passing Interface (MPI)?

- Portable standard for communication
- Processes can communicate through messages.
- Each process is a separable program
- All data is private

What is Message Passing Interface (MPI)?

- This is a library, not a language!!
- Different compilers, but all must use the same libraries, i.e. MPICH, LAM, etc.
- There are two versions now, MPI-1 and MPI-2
- Use standard sequential language. Fortran, C, C++, etc.

Basic Idea of Message Passing Interface (MPI)

- MPI Environment
  - Initialize, manage, and terminate communication among processes
- Communication between processes
  - Point to point communication, i.e. send, receive, etc.
  - Collective communication, i.e. broadcast, gather, etc.
- Complicated data structures
  - Communicate the data effectively
  - i.e. matrices and memory

Is MPI Large or Small?

- MPI is large
  - More than one hundred functions
  - But not necessarily a measure of complexity
- MPI is small
  - Many parallel programs can be written with just 6 basic functions
- MPI is just right
  - One can access flexibility when it is required
  - One need not master all MPI functions

When Use MPI?

- You need a portable parallel program
- You are writing a parallel library
- You care about performance
- You have a problem that can be solved in parallel ways

F77/F90, C/C++ MPI library calls

- Fortran 77/90 uses subroutines
  - CALL is used to invoke the library call
  - Nothing is returned, the error code variable is the last argument
  - All variables are passed by reference
- C/C++ uses functions
  - Just the name is used to invoke the library call
  - The function returns an integer value (an error code)
  - Variables are passed by value, unless otherwise specified

Types of Communication

- Point to Point Communication
  - communication involving only two processes.
- Collective Communication
  - communication that involves a group of processes.

Implementation of MPI

Getting started with LAM

- Create a file called “lamhosts”
- The content of “lamhosts” (8 notes):

```
cp0-1 cpu=2
cp0-2 cpu=2
cp0-3 cpu=2
…
cp0-8 cpu=2
frontend-0 cpu=2
```

Getting started with LAM

- starts LAM on the specified cluster
  - lamboot -v lamhosts
- removes all traces of the LAM session on the network
  - lamhalt
- In the case of a catastrophic failure (e.g., one or more LAM nodes crash), the lamhalt utility will hang
  - wipe -v lamhosts

MPI Commands

- mpicc - compiles an mpi program

```
mpicc -o foo foo.c
mpif77 -o foo foo.f
mpif90 -o foo foo.f90
```

- mpirun - start the execution of mpi programs

```
mpirun -v -np 2 foo
```

Basic MPI Functions

MPI Environment

- Initialize
  - initialize environment
- Finalize
  - terminate environment
- Communicator
  - create default communication group for all processes
- Version
  - establish version of MPI

MPI Environment

- Total processes
  - spawn total processes
- Rank/Process ID
  - assign identifier to each process
- Timing Functions
  - MPI_Wtime, MPI_Wtick

MPI_INIT

- Initializes the MPI environment
- Assigns all spawned processes to MPI_COMM_WORLD, default comm.
- C
  - int MPI_Init(argc,argv)
    - int *argc;
    - char ***argv;
  - Input Parameters
    - argc - Pointer to the number of arguments
    - argv - Pointer to the argument vector
- Fortran
  - CALL MPI_INIT(error_code)
  - int error_code – variable that gets set to an error code

MPI_FINALIZE

- Terminates the MPI environment
- C
  - int MPI_Finalize()
- Fortran
  - CALL MPI_FINALIZE(error_code)
  - int error_code – variable that gets set to an error code

MPI_ABORT

- This routine makes a “best attempt” to abort all tasks in the group of comm.
- Usually used in error handling.
- C
  - int MPI_Abort(comm, errorcode)
    - MPI_Comm comm
    - int errorcode
  - Input Parameters
    - comm - communicator of tasks to abort
    - errorcode - error code to return to invoking environment
- Fortran
  - CALL MPI_ABORT(COMM, ERRORCODE, IERROR)
  - INTEGER COMM, ERRORCODE, IERROR

MPI_GET_VERSION

- Get the version of currently used MPI
- C
  - int MPI_Get_version(int *version, int *subversion)
  - Input Parameters
    - version – version of MPI
    - subversion – subversion of MPI
- Fortran
  - CALL MPI_GET_VERSION(version, subversion, error_code)
  - int error_code – variable that gets set to an error code

MPI_COMM_SIZE

- This finds the number of processes in a communication group
- C
  - int MPI_Comm_size (comm, size)
    - MPI_Comm comm – MPI communication group;
    - int *size;
  - Input Parameter
    - comm - communicator (handle)
  - Output Parameter
    - size - number of processes in the group of comm (integer)
- Fortran
  - CALL MPI_COMM_SIZE(comm, size, error_code)
  - int error_code – variable that gets set to an error code
- Using MPI_COMM_WORLD will return the total number of processes started

MPI_COMM_RANK

- This gives the rank/identification number of a process in a communication group
- C
  - int MPI_Comm_rank (comm, rank)
    - MPI_Comm comm;
    - int *rank;
  - Input Parameter
    - comm - communicator (handle)
  - Output Parameter
    - rank – rank/id number of the process who made the call (integer)
- Fortran
  - CALL MPI_COMM_RANK(comm, rank, error_code)
  - int error_code – variable that gets set to an error code
- Using MPI_COMM_WORLD will return the rank of the process in relation to all processes that were started

Timing Functions – MPI_WTIME

- MPI_Wtime() - returns a floating point number of seconds, representing elapsed wall-clock time.
- C
  - double MPI_Wtime(void)
- Fortran
  - DOUBLE PRECISION MPI_WTIME()
- The times returned are local to the node/process that made the call.

Timing Functions – MPI_WTICK

- MPI_Wtick() - returns a double precision number of seconds between successive clock ticks.
- C
  - double MPI_Wtick(void)
- Fortran
  - DOUBLE PRECISION MPI_WTICK()
- The times returned are local to the node/process that made the call.

Hello World 1

- Echo the MPI version
- MPI Functions Used
  - MPI_Init
  - MPI_Get_version
  - MPI_Finalize

Hello World 1 (C)

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char *argv[])
{
    int version, subversion;
    MPI_Init(&argc, &argv);
    MPI_Get_version(&version, &subversion);
    printf("Hello world!\n");
    printf("Your MPI Version is: %d.%d\n", version, subversion);
    MPI_Finalize();
    return(0);
}
```

Hello World 1 (Fortran)

```
program main
include 'mpif.h'
integer ierr, version, subversion
call MPI_INIT(ierr)
call MPI_GET_VERSION(version,
subversion, ierr)
print *, 'Hello world!'
print *, 'Your MPI Version is: ', version, '.',
subversion
call MPI_FINALIZE(ierr)
end
```

Hello World 2

- Echo the process rank and the total number of process in the group
- MPI Functions Used
  - MPI_Init
  - MPI_Comm_rank
  - MPI_Comm_size
  - MPI_Finalize

Hello World 2 (C)

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char *argv[])
{
    int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf(”Hello world! I am %d of %d\n”, rank, size);
    MPI_Finalize();
    return(0);
}
```

Hello World 2 (Fortran)

```
program main
include 'mpif.h'
integer rank, size, ierr
call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierr)
print *, 'Hello world! I am ', rank, ' of ', size
call MPI_FINALIZE(ierr)
end
```

MPI C Datatypes

MPI Datatype	**C Datatype**
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int

MPI C Datatypes

MPI Datatype	C Datatype
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

MPI Fortran Datatypes

MPI Datatype	Fortran Datatype
MPI_INTEGER	INTEGER
MPI_REAL	REAL
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_LOGICAL	LOGICAL
MPI_CHARACTER	CHARACTER
MPI_BYTE	
MPI_PACKED	

The End