Message Passing Interface

Message Passing Interface (MPI)

- Message Passing Interface (MPI) is a specification designed for parallel applications.
- The goal of the Message Passing Interface is to provide a widely used standard for writing message passing programs. The interface attempts to be
  - practical
  - portable
  - efficient
  - flexible
- A message-passing library specification is:
  - message-passing model
  - not a compiler specification
  - not a specific product

Message Passing Interface

- MPI is designed to provide access to advanced parallel hardware for
  - end users
  - library writers
  - tool developers
- Message Passing is now mature as programming paradigm
  - well understood
  - efficient match to hardware
  - many applications
- Interface specifications have been defined for C and Fortran programs.

History of MPI

- April, 1992:
  - Workshop on Standards for Message Passing in a Distributed Memory Environment, sponsored by the Center for Research on Parallel Computing, Williamsburg, Virginia.
  - The basic features essential to a standard message passing interface were discussed, and a working group established to continue the standardization process. Preliminary draft proposal developed subsequently.
- November 1992:
  - Working group meets in Minneapolis. Oak Ridge National Laboratory (ORNL) presented a MPI draft proposal (MPI1).
  - Group adopts procedures and organization to form the MPI Forum. MPIF eventually comprised of about 175 individuals from 40 organizations including parallel computer vendors, software writers, academia and application scientists.

History of MPI

- November 1993: Supercomputing 93 conference - draft MPI standard presented.
- Final version of draft released in May, 1994 - available on the WWW at: http://www.mcs.anl.gov/Projects/mpi/standard.html
- MPI-2 picked up where the first MPI specification left off, and addresses topics which go beyond the first MPI specification.

Reasons for Using MPI

- **Standardization** - MPI is the only message passing library which can be considered a standard. It is supported on virtually all HPC platforms.
- **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** - Over 115 routines are defined.
- **Availability** - A variety of implementations are available, both vendor and public domain.

Types of Parallel Programming

- Flynn's taxonomy (hardware oriented)
  - SISD : Single Instruction, Single Data
  - SIMD : Single Instruction, Multiple Data
  - MISD : Multiple Instruction, Single Data
  - MIMD : Multiple Instruction, Multiple Data
- A programmer-oriented taxonomy
  - Data-parallel : Same operations on different data (SIMD)
  - Task-parallel : Different programs, different data
  - MIMD : Different programs, different data
  - SPMD : Same program, different data
  - Dataflow : Pipelined parallelism
- MPI is for SPMD/MIMD.

MPI Programming

- Writing MPI programs
- Compiling and linking
- Running MPI programs
- More information
  - *Using MPI* by William Gropp, Ewing Lusk, and Anthony Skjellum,
  - *Designing and Building Parallel Programs* by Ian Foster.
  - A Tutorial/User's Guide for MPI by Peter Pacheco (ftp://math.usfca.edu/pub/MPI/mpi.guide.ps)
  - The MPI standard and other information is available at http://www.mcs.anl.gov/mpi. Also the source for several implementations.

Simple MPI C Program (hello.c)

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char **argv )
{
  MPI_Init(&argc, &argv);
  printf("Hello world\n");
  MPI_Finalize();
  return 0;
}
```

- Complie hello.c: mpicc –o hello helllo.c
- #include <mpi.h> provides basic MPI definitions and types
- MPI_Init starts MPI
- MPI_Finalize exits MPI
- Note that all non-MPI routines are local; thus the printf run on each process

Simple MPI C++ Program (hello.cc)

```
#include <iostream.h>
#include <mpi.h>
Int main(int argc, char *argv[])
{
  int rank, size;
  MPI::Init(argc, argv);
  cout << "Hello world" << endl;
  MPI::Finalize();
  return 0;
}
```

- Compile hello.cc: mpiCC –o hello hello.cc

Starting and Exiting MPI

- Starting MPI

```
int MPI_Init(int *argc, char **argv)
void MPI::Init(int& argc, char**& argv)
```

- Exiting MPI

```
int MPI_Finalize(void)
void MPI::Finalize()
```

Running MPI Programs

- On many platforms MPI programs can be started with ‘ mpirun’.

mpirun -np <np> hello

- ‘mpirun’ is not part of the standard, but some version of it is common with several MPI implementations.
- Two of the first questions asked in a parallel program are as follows:
  1. “How many processes are there?”
  2. “Who am I?”
- “How many” is answered with MPI_COMM_SIZE;

  “Who am I” is answered with MPI_COMM_RANK.
- The rank is a number between zero and (SIZE -1).

Second MPI C Program

```
#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;
}
```

Second MPI C++ Program

```
#include <iostream.h>
#include <mpi.h>

int main(int argc, char **argv)
{
  MPI::Init(argc, argv);
  int rank = MPI::COMM_WORLD.Get_rank();
  int size = MPI::COMM_WORLD.Get_size();
  cout << "Hello world! I am " << rank << " of  "<< size << endl;
  MPI::Finalize();

  return 0;
}
```

Second MPI C Program

```
#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;
}
```

MPI_COMM_WORLD

- Communication in MPI takes place with respect to communicators (more about communicators later).
- The MPI_COMM_WORLD communicator is created when MPI is started and contains all MPI processes.
- MPI_COMM_WORLD is a useful default communicator – many applications do not need to use any other.
- mpirun -np <number of processes> <program name and arguments>

MPI C Data Type

MPI C Data Type	C Data Type
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_UNSIGNED_INT	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

MPI C++ Data Types

MPI C++ data type	C++ data type
MPI::CHAR	signed char
MPI::SHORT	signed short int
MPI::INT	signed int
MPI::LONG	signed long int
MPI::UNSIGNEDCHAR	unsigned char
MPI::UNSIGNEDSHORT	unsigned short int
MPI::UNSIGNED	unsigned int
MPI::UNSIGNEDLONG	unsigned long int
MPI::FLOAT	float
MPI::DOUBLE	double
MPI::LONGDOUBLE	long double
MPI::BYTE	
MPI::PACKED	

MPI Basic Functions

- **MPI_Init** – set up an MPI program.
- **MPI_Comm_size** – get the number of processes participating in the program.
- **MPI_Comm_rank** – determine which of those processes corresponds to the one calling the function.
- **MPI_Send** – send messages.
- **MPI_Recv** – receive messages.
- **MPI_Finalize** – stop participating in a parallel program.

MPI Basic Functions

- **MPI_Init(int *argc, char ***argv)**
  - Takes the command line arguments to a program,
  - checks for any MPI options, and
  - passes remaining command line arguments to the main program.
- **MPI_Comm_size(MPI_Comm comm, int *size)**
  - Determines the size of a given MPI Communicator.
  - A communicator is a set of processes that work together.
  - For typical programs this is the default MPI_COMM_WORLD, which is the communicator for all processes available to an MPI program.

MPI Basic Functions

- **MPI_Comm_rank(MPI_Comm comm, int *rank)**
  - Determine the rank of the current process within a communicator.
  - Typically, if a MPI program is being run on N processes, the communicator would be MPI_COMM_WORLD, and the rank would be an integer from 0 to N-1.
- **MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)**
  - Send the contents of **buf**, which contains count elements of type **datatype** to a process of rank **dest** in the communicator **comm**, flagged with the message **tag**. Typically, the communicator is MPI_COMM_WORLD.

MPI Basic Functions

- **MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)**
  - Read into **buf**, which contains **count** elements of type **datatype** from process **source** in communicator **comm** if a message is sent flagged with **tag**. Also receive information about the transfer into **status**.
- **MPI_Finalize()**
  - Handles anything that the current MPI protocol will need to do before exiting a program.
  - Typically should be the final or near final line of a program.

MPI Functions

- **MPI_Bcast –** Broadcasts a message from the process with rank "root" to all other processes of the group.
  - **int MPI_Bcast (void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)**
  - Input/output Parameters
    - **buffer** starting address of buffer (choice)
    - **count** number of entries in buffer (integer)
    - **datatype** data type of buffer (handle)
    - **root** rank of broadcast root (integer)
    - **comm** communicator (handle)

MPI Functions

- **MPI_Reduce –** Reduces values on all processes to a single value
  - int MPI_Reduce (void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
  - Input/output Parameters
    - **sendbuf** address of send buffer (choice)
    - **count** number of elements in send buffer (integer)
    - **datatype** data type of elements of send buffer (handle)
    - **op** reduce operation (handle)
    - **root** rank of root process (integer)
    - **comm** communicator (handle)

MPI Reduce Operation

MPI Name	Operation
MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_PROD	Product
MPI_SUM	Sum
MPI_LAND	Logical and
MPI_LOR	Logical or
MPI_LXOR	Logical exclusive or (xor)
MPI_BAND	Bitwise and
MPI_BOR	Bitwise or
MPI_BXOR	Bitwise xor
MPI_MAXLOC	Maximum value and location
MPI_MINLOC	Minimum value and location

MPI Example Programs

- **Master/Worker** (greeting.c, master-worker.c, ring.c) – A (master) process waits for the message sent by other (worker) processes.
- **Integration** (integrate.c): Evaluates the trapezoidal rule estimate for an integral of F(x).
- **π Calculation** (pi.c) – Each process calculates part of π and summarize in the master process.
- **Matrix Multiplication** (matrix-multiply.c, matrix.c) – Each worker process calculates some rows of matrix multiplication and sends the result back to the master process.

MPI Example Programs

- **Exponential** (exponent.c): Evaluates *e* for a series of cases. This is essentially the sum of products of (1 - exp()).
- **Statistics** (statistics.c) : Tabulates statistics of a set of test scores. :
- **Tridiagnonal system** (tridiagonal.c) : Solves the tridiagonal system Tx = y.