Message Passing Interface (MPI)

Arash Bakhtiari

2013-01-13 Sun

Distributed Memory

- Processors have their own local memory

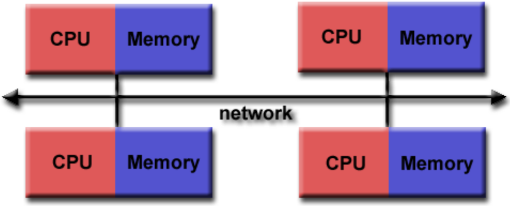

Figure : Distributed Memory [1]

Advantages and Disadvantages

Advantages:

- Memory is scalable with the number of processors
- Each processor can rapidly access its own memory without interference

Disadvantages:

- programmer is responsible:
  - to provide data in another processor
  - to explicitly define how and when data is communicated
  - to synchronize between tasks

What is MPI?

- MPI is a specification for the developers and users of message passing libraries
- Provide a standard for writing message passing programs
- Specifications is available for C/C++ and Fortran

Advantages of MPI

- **Standardization**: MPI is the only message passing library which can be considered a standard
- **Portability**: no need to modify your source code when you port your application to a different platform
- **Functionality**: Many routines available to use
- **Availability**: A variety of implementations are available

MPI Program Structure

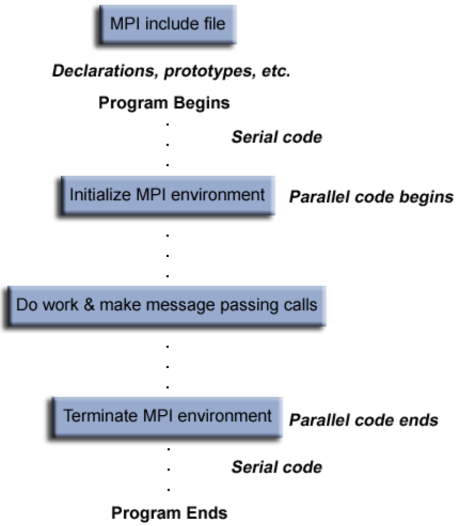

Figure : MPI Program Structure [1]

Core Routines

- **MPI_Init**: Initializes the MPI execution environment

```
int MPI_Init( int *argc, char ***argv )
```

- **MPI_Finalize**: Terminates the MPI execution environment

```
MPI_Finalize ()
```

- **MPI_Comm_size**: Returns the total number of MPI processes in the specified communicator

```
int MPI_Comm_size( MPI_Comm comm, int *size )
```

- **MPI_Comm_rank**: Returns the rank of the calling MPI process within the specified communicator.

```
int MPI_Comm_rank( MPI_Comm comm, int *rank )
```

DEMO

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

int main(int argc, char **argv) {

  int my_rank;
  int size;

  MPI_Init(&argc, &argv); /*START MPI */

  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  printf("Hello world! I'm rank %d.\n",my_rank);

  MPI_Finalize();  /* EXIT MPI */

}
```

Communicaiton Routines

Point to Point Communication:

- Involve message passing between two, and only two, different MPI tasks
- One task performe a send operation and the other task performe a matching receive operation

Collective Communication:

- Collective communication must involve all processes in the scope of a communicator

Communicaiton Routines: Point-to-Point

- **MPI_Send**:

```
int MPI_Send(void *buf, int count, MPI_Datatype datatype
             MPI_Comm comm)
```

- **MPI_Recv**:

```
int MPI_Recv(void *buf, int count, MPI_Datatype datatype
             MPI_Comm comm, MPI_Status *status)
```

Communicaiton Routines: Collective

- **MPI_Bcast**: Broadcasts (sends) a message from the process with rank "root" to all other processes in the group

```
int MPI_Bcast( void *buffer, int count,
               MPI_Datatype datatype,
               int root, MPI_Comm comm )
```

- **MPI_Barrier**: Creates a barrier synchronization in a group

```
int MPI_Barrier( MPI_Comm comm )
```

DEMO

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

int main(int argc, char *argv[])
{
    int rank, size, i;
    int buffer[ 10 ];
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (rank == 0)
    {
        for (i=0; i<10; i++)
            buffer[i] = i;
        MPI_Send(buffer, 10, MPI_INT,
                 1, 123, MPI_COMM_WORLD);
    }
```

DEMO

```
    if (rank == 1)
    {
        for (i=0; i<10; i++)
            buffer[i] = -1;
        MPI_Recv(buffer, 10, MPI_INT,
                 0, 123, MPI_COMM_WORLD,
                 &status);
        for (i=0; i<10; i++)
        {
            printf("buffer[%d] = %d\n", i, buffer[i]);
        }
        fflush(stdout);
    }
    MPI_Finalize();
    return 0;
}
```

References

- Blaise Barney, Lawrence Livermore National Laboratory, [[https://computing.llnl.gov/tutorials/mpi/][https://computing.llnl.gov/tutorials/mpi/]]
- DeinoMPI [[http://mpi.deino.net/][http://mpi.deino.net/]]