Case Study: MPI

-Spurthy.S

Introduction:

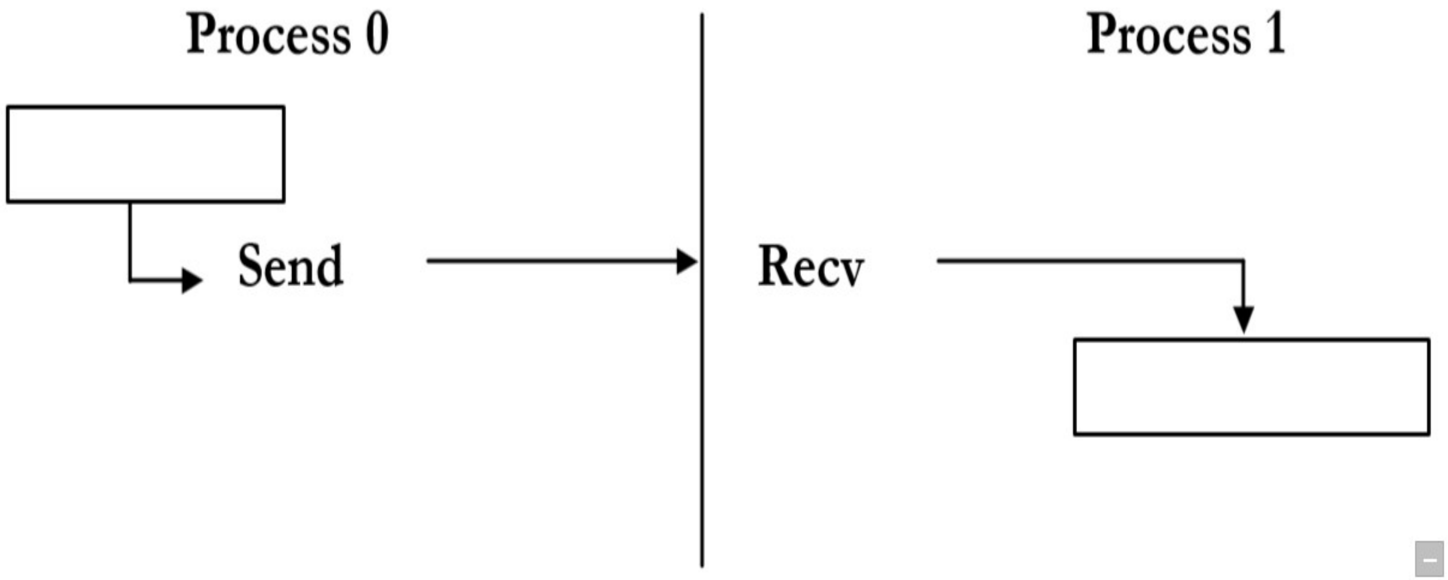
- An MPI is an Application Programming Interface(API) for communication between separate processes.
- It specifies library routines needed for writing message passing programs.
- It is used for distributed parallel computing.
- It is portable, scalable, flexible.
- It is platform independent.

A message-passing library specification is:

- Message-passing model
- Not a compiler specification
- Not a specific product
- Used for parallel computers, clusters, and heterogeneous networks as a message passing library.
- Designed to permit the development of parallel software libraries

MPI Send and Receive

Sending and Receiving messages



MPI Routines:

- **MPI_Send()** : sends a message from the current process to another process (the destination).
- **MPI_Recv()** : receives a message on the current process from another process (the source).
- **MPI_Bcast()** : broadcasts a message from one process to all of the others.
- **MPI_Reduce()** : performs a reduction of a variable in all processes, with the result ending up in a single process.
- **MPI_Allreduce()** : performs a reduction of a variable in all processes, with the result ending up in all processes.

Other functions:

- MPI_Init()
- MPI_Finalize()
- MPI_Comm_size()
- MPI_Comm_rank()

Parameters of MPI_SEND():

- int MPI_Send (void *buf,

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

Parameters-Explanation

- buf - initial address of send buffer
- count - number of elements in send buffer
- datatype - datatype of each send buffer element
- dest - rank of destination (integer)
- tag - message tag (integer)
- comm - communicator

Parameters of MPI_RECV():

- int MPI_Recv (void *buf,

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

Parameters-Explanation

- buf - initial address of receive buffer
- count - number of elements in receive buffer (integer)
- datatype - datatype of each receive buffer element
- source - rank of source (integer)
- tag - message tag (integer)
- comm - communicator
- status - status object (Status)

Send operations	*Blocking*	*Non-blocking*
Generic	*MPI_Send*: the sender blocks until it is safe to return – that is, until the message is in transit or delivered and the sender's application buffer can therefore be reused.	*MPI_Isend*: the call returns immediately and the programmer is given a communication request handle, which can then be used to check the progress of the call via *MPI_Wait* or *MPI_Test*.
Synchronous	*MPI_Ssend*: the sender and receiver synchronize and the call only returns when the message has been delivered at the receiving end.	*MPI_Issend*: as with *MPI_Isend*, but with *MPI_Wait* and *MPI_Test* indicating whether the message has been delivered at the receive end.
Buffered	*MPI_Bsend*: the sender explicitly allocates an MPI buffer library (using a separate *MPI_Buffer_attach* call) and the call returns when the data is successfully copied into this buffer.	*MPI_Ibsend*: as with *MPI_Isend* but with *MPI_Wait* and *MPI_Test* indicating whether the message has been copied into the sender's MPI buffer and hence is in transit.
Ready	*MPI_Rsend*: the call returns when the sender's application buffer can be reused (as with *MPI_Send*), but the programmer is also indicating to the library that the receiver is ready to receive the message, resulting in potential optimization of the underlying implementation.	*MPI_Irsend*: the effect is as with *MPI_Isend*, but as with *MPI_Rsend*, the programmer is indicating to the underlying implementation that the receiver is guaranteed to be ready to receive (resulting in the same optimizations),

MPI 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	

MPI Datatypes – C:

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_UNSIGNED	Unsigned int
MPI_UNSIGNED_LONG	Unsigned long int
MPI_FLOAT	Float
MPI_DOUBLE	Double
MPI_LONG_DOUBLE	Long double
MPI_BYTE	
MPI_PACKED	

Writing MPI Program:

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

```c
#include "mpi.h"
#include <stdio.h>
int main( int argc, char **argv)
{
 int rank;
 MPI_Init( &argc, &argv );
 MPI_Comm_rank( MPI_COMM_WORLD, &rank );
 printf( "Hello world! I'm Process:%d\n", rank );
 MPI_Finalize();
 return 0;
 }
```

To send 'x' from P0 to P1:

- MPI_Comm_rank(MPI_COMM_WORLD, &rank);

```
if( rank == 0)
{
 int x;
MPI_Send( &x, 1, MPI_INT, 1, msgtag, MPI_COMM_WORLD);
 }
else if( rank == 1)
{
 int x;
MPI_Recv( &x, 1, MPI_INT, 0, msgtag, MPI_COMM_WORLD,status);
 }
```

THANK YOU