**11.2 IPC: MPI(C library for message passing between processes of different systems) Distributed memory programming.**

**Objectives:**

1. To learn about IPC through MPI.

2. Use of IPC mechanism to write effective application programs.

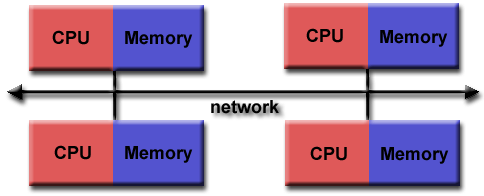
3. configure cluster and experiment MPI program on it.

**Theory:**

**What is MPI** ? (Message Passing Interface)

* MPI is a **specification** for the developers and users of message passing libraries. By itself, it is NOT a library - but rather the specification of what such a library should be.
* MPI primarily addresses the **message-passing parallel programming model:** data is moved from the address space of one process to that of another process through cooperative operations on each process.
* Simply stated, the goal of the Message Passing Interface is to provide a widely used standard for writing message passing programs.

 **Programming Model:**

* Originally, MPI was designed for distributed memory architectures, which were becoming increasingly popular at that time (1980s - early 1990s).  
  Fig 11.2 MPI
* As architecture trends changed, shared memory SMPs were combined over networks creating hybrid distributed memory / shared memory systems.
* MPI implementors adapted their libraries to handle both types of underlying memory architectures seamlessly. They also adapted/developed ways of handling different interconnects and protocols.
* Today, MPI runs on virtually any hardware platform:
  + Distributed Memory
  + Shared Memory
  + Hybrid
* The programming model clearly remains a distributed memory model however, regardless of the underlying physical architecture of the machine.
* All parallelism is explicit: the programmer is responsible for correctly identifying parallelism and implementing parallel algorithms using MPI constructs.

**Reasons for Using MPI:**

* Standardization
* Portability
* Performance Opportunities
* Functionality
* Availability

**Program:**

#include <boost/mpi.hpp>

#include <iostream>

int main(int argc, char\* argv[])

{

boost::mpi::environment env(argc, argv);

boost::mpi::communicator world;

if (world.rank() == 0) {

world.send(1, 9, 32);

world.send(2, 9, 33);

} else {

int data;

world.recv(0, 9, data);

std::cout << "In process " << world.rank( ) << "with data " << data

<< std::endl;

}

return 0;

}

**Output:**

In process 1 with data 32

In process 2 with data 33

**Conclusion:**

1.From a usability standpoint, IPC is easy to use. However, the MPI library is dependant on native MPI implementations,

**References:**

[1] http://mpitutorial.com/tutorials/mpi-introduction/