Computer Science & Information Systems

**Systems for Data Analytics - Lab Sheet 3**

**Message Passing Model**

1. Objective:

Students should be able to

1. Get familiarity with the Message Passing model in Parallel computing environment
2. Get hands-on experience on MPI APIs

This lab sheet provides a quick introduction to using MPI(Message passing Interface). This exercise will introduce the different API used in MPI in C language.

A *process* is (traditionally) a program counter and address space. Processes may have multiple *threads* (program counters and associated stacks) sharing a single address space.

MPI is for communication among processes, which have separate address spaces.

**Interprocess communication consists of** Synchronization and Movement of data from one process’s address space to another’s.

The message-passing approach makes the exchange of data *cooperative*. Data is explicitly *sent* by one process and *received* by another. An advantage is that any change in the receiving process’s memory is made with the receiver’s explicit participation. Communication and synchronization are combined.

Process 0

Process 1

**Send(data)**

**Receive(data)**

*What is MPI?*

A *message-passing library specification -* extended message-passing model, not a language or compiler specification, not a specific implementation or product

For parallel computers, clusters, and heterogeneous networks, MPI is designed to provide access to advanced parallel hardware for end users, library writers, tool developers

1. Steps to be performed:

Lab used – remote lab with Machines – MPI01 and MPI02

Language used – C

‘Hello\_World.c’ - first step to building an MPI program is including the MPI header files with #include <mpi.h>

The MPI environment must be initialized with:

**MPI\_Init(int\* argc, char\*\*\* argv)** - all of MPI’s global and internal variables are constructed. For example, a communicator is formed around all of the processes that were spawned, and unique ranks are assigned to each process.

**MPI\_Comm\_size( MPI\_Comm communicator, int\* size)** - MPI\_Comm\_size reports the number of processes.

**MPI\_Comm\_rank(MPI\_Comm communicator, int\* rank)** - MPI\_Comm\_rank reports the rank, a number between 0 and size-1, identifying the calling process

**MPI\_Finalize()** is used to clean up the MPI environment. No more MPI calls can be made after this one

To compile: mpicc -o mpi\_hello-mpi mpi\_hello-mpi.c

To run: mpirun -n 4 -f host\_file ./mpi\_hello-mpi

-n flag to set the number of MPI processes

-f flag to get the hosts(IP address or hostnames) listed in host\_file

If you do not have the host\_file, use –hosts flag in the following way:

mpirun –np 4 –hosts 192.168.1.2,192.168.1.3,.. ./mpi\_hello\_world

1. Outputs/Results:

mpirun -n 4 -f host\_file ./mpi\_hello\_world

Hello world from processor host2, rank 1 out of 4 processors

Hello world from processor host1, rank 0 out of 4 processors

Hello world from processor host4, rank 3 out of 4 processors

Hello world from processor host3, rank 2 out of 4 processors

Or

mpirun –n 4 –hosts 192.168.1.2,192.168.1.3,.. ./mpi\_hello\_world

Hello world from processor 192.168.1.2, rank 1 out of 4 processors

Hello world from processor 192.168.1.3, rank 0 out of 4 processors

Hello world from processor 192.168.1.2, rank 3 out of 4 processors

Hello world from processor 192.168.1.3, rank 2 out of 4 processors

**Make sure the executable is present in the same path in all the machines/ hosts with the same name.**

1. Observations:

* Students to observe the creation of new processes due to MPI and how work is scheduled to those processes.
* Processes can be created and executed in multiple hosts in the same network