

How to for compiling and running MPI Programs.

Prepared by Kiriti Venkat



What is MPI ?



- MPI stands for Message Passing Interface
- MPI is a library specification of message-passing, proposed as a standard by a broadly base committee of vendors ,implementers and users.
- MPI was designed for high performance on both massively parallel machines and on workstation clusters.

Goals of MPI



- Develop a widely used standard for writing message-passing programs. As such the interface should establish a practical, portable, efficient, and flexible standard for message passing.
- The design of MPI primarily reflects the perceived needs of application programmers.
- Allow efficient communication by
 - Avoid memory-to-memory copying,
 - allow overlap of computation and communications, and
 - offload to a communication coprocessor-processor, where available.
- Allow for implementations that can be used in a heterogeneous environment.

MPI Programming Environments in Linux Cluster



There are two basic MPI programming environments available in OSCAR, they are

- **LAM/MPI**
- **MPICH**

LAM/MPI



- LAM (Local Area Multicomputer) is an MPI programming environment and development system for heterogeneous computers on a network .
- With a LAM/MPI , a dedicated cluster or an existing network computing infrastructure can act as a single parallel computer.
- LAM/MPI is considered to be a “cluster friendly”, in that it offers daemon-based process startup/control as well as fast client-to-client message passing protocols.
- It can use TCP/IP and/or shared memory for message passing.

MPICH



- An open-source, portable implementation of the MPI Standard led by ANL.
- Implementation of MPI version 1.2 and also significant parts of MPI-2, particularly in the area of parallel I/O.
- MPMD (Multiple Program and Multiple Data) and heterogeneous are supported .
- Supports clusters of SMPs and massively parallel computers.
- MPICH also includes components of a parallel programming environment like tracing and logfile tools based on the MPI profiling interface ,including a scalable log file format (SLOG) ,parallel performance visualization tools, extensive correctness and performance tests and supports both small and large applications.

MPICH



Commands for switching between MPI programming environments in OSCAR.

\$ switcher mpi -show //displays default environment

\$ switcher mpi -list // list of environments that are available

\$ switcher mpi = lam-6.5.9 //This will set all future shells to use lam-6.5.9 as default environment

For more Information on switcher refer **man switcher**

Running MPI-Program in LAM environment



Before running MPI programs LAM daemon must be started on each node of cluster

```
$ recon -d lamhosts // recon is for test cluster is connected or not.  
Lamhost is list of host names (ip)
```

```
$ lamboot -v lamhosts //lamboot tool starts lam on the specified  
//cluster. Lamhosts are list of nodes on cluster.
```

Compiling MPI programs

```
$ mpicc -o foo foo.c // It compiles foo.c source code to foo object  
code.
```

```
$ mpif77 -o foo foo.f // It compiles foo.f source code to foo object code.
```

Running MPI-Program in LAM environment



\$ mpirun -v -np 2 foo // this runs foo program on the available nodes
.-np for running given number of copies of the program on given nodes.

To run multiple programs at the same time create a application schema file that lists each program and its nodes.

\$ cat appfile

n0 master

n0-1slave

\$ mpirun -v appfile // this runs master and slave programs on 2 nodes simultaneously



Monitoring MPI Applications

\$ mpitask //full mpi synchronization status of all processes and messages are displayed

\$ mpimsg //displayes messages sources and destinations and data types.

\$ lamclean -v //all users processes and messages are removed.

\$ lamhalt // removes lam daemons on all nodes.

Submitting jobs through PBS

qsub: submits job to PBS

qdel: deletes PBS job

qstat [-n]: displays current job status and node associations

pbsnodes [-a]: displays node status

pbsdsh: distributed process launcher

qsub command

```
$ qsub ring.qsub
```

Ring.qsub.o? as stdout and if error, output into **Ring.qsub.e?**

An example script is given in [http://xcr.cenit.latech.edu/mpiinfo.\(mpi.tar.gz\)](http://xcr.cenit.latech.edu/mpiinfo.(mpi.tar.gz))



Submitting jobs through PBS



qsub: submits job to PBS

qdel: deletes PBS job

qstat [-n]: displays current job status and node associations

pbsnodes [-a]: displays node status

pbsdsh: distributed process launcher

qsub command

```
$ qsub -N my_jobname -e my_stderr.txt -o my_stdout.txt -q workq -l \
      nodes=X:ppn=Y:all,walltime=1:00:00 my_script.sh
```

Example of my_script.sh

```
#!/bin/sh
echo Launchnode is `hostname`
lamboot lamhost
pbsdsh /path/to/my_executable
```

The above command distribute the job (my_executable) to x nodes and y processors using pbsdsh and output is piped into my_stdout.txt and if error, is piped into my_stderr.txt .

An example script is given in [http://xcr.cenit.latech.edu/mpiinfo.\(mpi.tar.gz\)](http://xcr.cenit.latech.edu/mpiinfo.(mpi.tar.gz))