Setting up a Beowulf Cluster Using Open MPI on Linux

2 December 2009 Lawrence Woodman

#Beowulf Clusters #Distributed Processing #High Performance Computing #Linux #MPI #Parallel Processing

I have been doing a lot of work recently on Linear Genetic Programming. This requires a great deal of processing power and to meet this I have been using Open MPI to create a Linux cluster. What follows is a quick guide to getting a cluster running. The basics really are very simple and, depending on the size, you can get a simple cluster running in less than half an hour, assuming you already have the machines networked and running Linux.

What is a Beowulf Cluster?

A Beowulf Cluster is a collection of privately networked computers which can be used for parallel computations. They consist of commodity hardware running open source software such as Linux or a BSD, often coupled with PVM (Parallel Virtual Machine) or MPI (Message Passing Interface). A standard set up will consist of one master node which will control a number of slave nodes. The slave nodes are typically headless and generally all access the same files from a server. They have been referred to as Beowulf Clusters since before 1998 when the original Beowulf HOWTO was released.

What is Open MPI?

Open MPI is one of the leading MPI-2 implementations used by many of the TOP 500 Supercomputers. It is open source and is developed and maintained by a consortium of academic, research, and industry partners. The Open MPI project has the stated aim of building the best Message Passing Interface (MPI) library available, which judging by where it is used, I would say they

are well on their way to doing.

The main way that it is used is to start a program from the master node and specify how many processes you want to use, the program is then started for each of those process whether this is on the same machine or over multiple machines. The processes can communicate using the MPI library and all data sent to STDOUT from the slave nodes is piped to STDOUT on the master node.

Setting up the Machines

I am making several assumptions in this article:

- You have at least two machines connected via a TCP/IP network.
- Each machine is using the same processor architecture, to ease sharing your program executables.
- The machines all have Linux installed.
- Each machine has a common login user name, such as `mpiuser`. I will refer to the common login as `mpiuser` for the rest of this article.
- Each machine has `mpiuser` sharing the same home folder, usually through NFS, or has the relevant parts of the home folder synchronised, perhaps with `rsync`.
- I will refer to each of the slaves nodes as `slave1`, `slave2`, etc. You, however, can choose whatever host names you like.

One of the machines will be the master node, this is the machine from which you will control the cluster and run your programs. The other nodes will be known as slaves.

Installing and Configuring Open MPI on all Nodes

Under Debian Lenny you need to install the following packages:

- openmpi-bin
- openmpi-common
- libopenmpi1
- libopenmpi-dev (Not actually needed for a slave node.)

This can be done as root, on Debian, with:

```
$ apt-get install openmpi-bin openmpi-common libopenmpi1 libopenmpi-dev
```

The names may vary on your distribution, but the most important thing to remember is that you should have the same version of

Open MPI on all the machines, for it to work properly.

Under Debian Lenny, the Open MPI executables install into `/usr/bin`. If they install into a different location, then you will need to ensure that this location is in the PATH for `mpiuser`. You may also need to ensure that LD_LIBRARY_PATH points to `/usr/lib`

Slave Nodes

So that the master node can control the slave nodes, each slave node needs an SSH server installed. Under Debian, install the following package:

- openssh-server

Under Debian, this can be done as root with:

```
$ apt-get install openssh-server
```

Master Node

Setting up SSH

Before continuing with setting up Open SSH please refer to their site. I am providing just enough information here to get the cluster working and make no guarantees as to how secure this method is.

To be able to control the slave nodes from the master node you need an SSH client installed. Under Debian, install the following package:

- openssh-client

Under Debian, this can be done as root with:

```
$ apt-get install openssh-client
```

While logged in as `mpiuser`, create SSH public/private key pairs with password: `h475k2!4553ffe` (Choose your own password) using file: `/home/mpiuser/.ssh/id_dsa`.

```
$ ssh-keygen -t dsa
```

Make sure that each machine knows that this user is authorized to log into them.

```
$ cp /home/mpiuser/.ssh/id_dsa.pub /home/mpiuser/.ssh/authorized_keys
```

If you are not sharing the `/home/mpiuser` folder then make sure that each slave node knows that `mpiuser` on the master node is authorized to log into them. For `slave1` use:

```
$ scp /home/mpiuser/.ssh/id_dsa.pub mpiuser@slave1:.ssh/authorized_keys
```

Correct file permissions (This will also need to be done on the slave nodes if not sharing the home folder):

```
$ chmod 700 /home/mpiuser/.ssh
$ chmod 600 /home/mpiuser/.ssh/authorized_keys
```

To test that this has worked try using SSH to connect to your slaves from the master, using the password you entered when creating the key pairs above:

```
$ ssh slave1
```

Configuring Open MPI

To let Open MPI know which machines to run your programs on, you can create a file to store this. I will call this file `/home/mpiuser/.mpi_hostfile` and it could contain the following:

```
# The Hostfile for Open MPI

# The master node, 'slots=2' is used because it is a dual-processor machine.
localhost slots=2

# The following slave nodes are single processor machines:
slave1
slave2
slave3
```

Compiling Programs

OpenMPI can be used with a variety of languages, two of the most popular are FORTRAN and 'C'. If your programs are written in 'C', then you can either use `mpicc` instead of your normal 'C' compiler or you can pass the additional arguments directly to your 'C' compiler. With `mpicc` the arguments you pass are passed to your normal 'C' compiler.

If you want to use `mpicc` to compile a 'C' source file called

`testprogram.c` :

```
$ mpicc testprogram.c
```

If you want to see what will be passed to your 'C' compiler when compiling and linking:

```
$ mpicc -showme
```

Running Your Programs on the Cluster

Ensure That SSH Doesn't Ask For a Password

Because Open MPI will use SSH to connect to each of the machines and run your program, you need to ensure that the password doesn't have to be entered for each connection.

Use `ssh-agent` to remember the password while logged in as `mpiuser` :

```
$ eval `ssh-agent`
```

Tell `ssh-agent` the password for the SSH key:

```
$ ssh-add ~/.ssh/id_dsa
```

Test by logging into one of the slave nodes, while logged in as mpiuser; this shouldn't ask for a password:

```
$ ssh slave1
```

Starting a Program on the Cluster

You can start a program on just one machine, and have it execute on multiple processors/cores. This is handy if you have a powerful machine or are debugging/testing a program. Alternatively, you can run the program over your cluster and request as many processes as you want to be run over it.

To run `myprogram` on two processes on the local machine:

```
$ mpirun -np 2 ./myprogram
```

To run `myprogram` over five processes on the cluster using the `.mpi_hostfile` created above (Note that `myprogram` must be in the same

location on each machine):

```
$ mpirun -np 2 --hostfile .mpi_hostfile ./myprogram
```

An Example Test Program

The following program will send a random number from the master node to all the slave nodes.

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

#include <mpi.h>

int main(int argc, char *argv[])
{
  const int MASTER = 0;
  const int TAG_GENERAL = 1;

  int numTasks;
  int rank;
  int source;
  int dest;
  int rc;
  int count;
  int dataWaitingFlag;

  char inMsg;
  char outMsg;

  MPI_Status Stat;

  // Initialize the MPI stack and pass 'argc' and 'argv' to each slave node
  MPI_Init(&argc,&argv);

  // Gets number of tasks/processes that this program is running on
  MPI_Comm_size(MPI_COMM_WORLD, &numTasks);

  // Gets the rank (process/task number) that this program is running on
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  // If the master node
  if (rank == MASTER) {

    // Send out messages to all the sub-processes
    for (dest = 1; dest < numTasks; dest++) {
      outMsg = rand() % 256;       // Generate random message to send to slave nodes

      // Send a message to the destination
      rc = MPI_Send(&outMsg, 1, MPI_CHAR, dest, TAG_GENERAL, MPI_COMM_WORLD);
      printf("Task %d: Sent message %d to task %d with tag %d\n",
             rank, outMsg, dest, TAG_GENERAL);
    }

  }
```

```
  // Else a slave node
  else  {
    // Wait until a message is there to be received
    do {
      MPI_Iprobe(MASTER, 1, MPI_COMM_WORLD, &dataWaitingFlag, MPI_STATUS_IGNORE);
      printf("Waiting\n");
    } while (!dataWaitingFlag);

    // Get the message and put it in 'inMsg'
    rc = MPI_Recv(&inMsg, 1, MPI_CHAR, MASTER, TAG_GENERAL, MPI_COMM_WORLD, &Stat);

    // Get how big the message is and put it in 'count'
    rc = MPI_Get_count(&Stat, MPI_CHAR, &count);
    printf("Task %d: Received %d char(s) (%d) from task %d with tag %d \n",
            rank, count, inMsg, Stat.MPI_SOURCE, Stat.MPI_TAG);
  }

  MPI_Finalize();
}
```

If you save the above source code to a file called `testprogram.c` then compile it using:

```
$ mpicc testprogram.c
```

Then run it either over 20 processes on the local machine:

```
$ mpirun -np 20 ./a.out
```

Or over 5 processes on the cluster:

```
$ mpirun -np 5 --hostfile .mpi_hostfile ./a.out
```

You should see on the master node that the output to STDOUT from the slave nodes is redirected to STDOUT on the master node. The master node will output what it is sending and each slave node will output what it has received or if it is waiting. Once each node has finished, the master node will exit and return you to the command prompt.

Where Can I Find More Information?

For Beowulf clusters take a look at beowulf.org and Cluster Monkey. There is lots of information available for Open MPI at the Open MPI Project and there is an excellent tutorial for programming using MPI at Lawrence Livermore National Laboratory.

Share This Post

Reddit Facebook Twitter Email

Feedback/Discuss

Email

Related Articles

Beware of Immutable Lists for F# Parallel Processing

19 April 2014

#F# #Parallel Processing #Programming

With F#, the list often feels like the default choice of data structure. It is immutable and hence easy to reason about, however its use can come at a great cost. If you are using lists to process la... Read More

Using C-Kermit to Exchange Files With Telnet BBS's

9 April 2013

#BBS #Linux #Retro

Most BBSs that are still running now do so via telnet. In many ways this is great as it allows people from all around the world to access a BBS as if it were local to them. The problem comes though, ... Read More

Connecting to a Remote Serial Port over TCP/IP

2 April 2013

#Emulation #Linux #Retro

Most modern machines don't have a serial port as standard; you could use a USB to serial lead, however, if you have another machine available that does have a serial port you can access it remotely ove... Read More

Using Netcat to Create ad hoc Links Between Applications or Machines

25 March 2013

#Linux

Netcat is a simple Unix utility which reads and writes data across network connections using the TCP or UDP protocol. It is often described as the "Swiss-army knife for TCP/IP" because of it... Read More

Getting Colour ANSI Emulation to Work Properly When Connecting to a BBS With Telnet Under Linux

14 February 2010

#ANSI #BBS #Linux #Retro #Text Mode

I have noticed that the number of people interested in using telnet to access BBSs seems to be growing, which I'm really pleased to see. However lots of people seem to be having trouble getting colour... Read More