RIT Computer Engineering Cluster

The RIT Computer Engineering cluster contains 16 computers for parallel programming using MPI. One computer, rocks.ce.rit.edu, serves as the master controller or head node for the cluster and is accessible from the Internet. The other 15 machines, named compute-0-0, compute-0-1, and so on through compute-0-14, are attached to a private LAN segment and are visible only to each other and the cluster head node. Shorter names for the machines are c0-0, c0-1, etc.

The hardware for each cluster node consists of the following:

- Intel SR1530CL Server Chassis
    - Intel S5000VCL Motherboard
- Two Intel Xeon 5140 (2.33GHz) Dual-Core Processors
- Two 1GB PC2-5300 Buffered ECC CL5 Dual Rank Kingston Memory Modules
- 250GB SATA 3.0Gbps Maxtor or Seagate Hard Drive

The nodes are connected via a Netgear Prosafe 48-port Gigabit Switch (GS748T). The login node, *cluster*, is identical to a compute node, with exception of double the amount of RAM and disk space.

Additionally, there is a specific MPI interface installed on each machine and this should be used exclusively for MPI traffic. The interface is named mpi, mpi-0-0, mpi-0-1, through mpi-0-14. Please use these names in your machines file instead of the compute names.

To connect to the cluster, simply use a SSH or SFTP client to connect to:

<username>@rocks.ce.rit.edu

using your DCE login information (username and password).

The head node only supports secure connections using SSH and SFTP; normal Telnet and FTP protocols simply won't work. Once you're inside the cluster, however, you may telnet or rlogin to any machine on the cluster at your convenience.

SSH Clients

Putty, a very small and extremely powerful SSH client, is available from:
    http://www.chiark.greenend.org.uk/~sgtatham/putty/
or from the mirror sight:
    http://www.putty.nl/
This SSH client supports X11 forwarding, so if you use an XWindow emulator such as Exceed, ReflectionX, or Xming, you may open graphical applications remotely over the SSH connection. The website also includes a command line secure FTP client.

WinSCP is an excellent graphical FTP/SFTP/SCP client for Windows. It is available from:
    http://winscp.net/eng/index.php

Xming X Server is a free X Window Server for Windows. It is available from:
    http://www.straightrunning.com/XmingNotes/

Using Message Passing Interface on the RIT Computer Engineering Cluster

MPI is designed to run Single Program Multiple Data (SPMD) parallel programs on homogeneous cluster or supercomputer systems. The RIT Computer Engineering cluster contains a full implementation (version 1.2.7) of Argonne National Laboratories' *mpich* parallel programming library. MPI uses shell scripts and the remote shell to start, stop, and run parallel programs remotely. Thus, MPI programs terminate cleanly, and require no additional housekeeping or special process management.

Summary

Test machines with:	/opt/mpich/gnu/sbin/tstmachines -v -machinefile=mymachines
Compile using:	mpicc [linking flags]
Run programs with:	mpirun –np # executable

Specifying Machines

MPI is configured to use all of the machines on the cluster by default. You may create your own list of machines in any text file in your home directory with one machine per line. For example:

```
[abc1234@rocks abc1234]$ cat mymachines
mpi-0-0
mpi-0-1
mpi-0-2
```

Why create your own list of machines? MPI assigns tasks to processors sequentially in the order they are listed in the machines file. By default, all users use all of the machines in order, that is, mpi-0-0, mpi-0-1, mpi-0-2, and so on. If everyone starts a program requiring two processes, they will run on mpi-0-0 and mpi-0-1 while mpi-0-2 through mpi-0-14 sit idle. Make sure you have enough machines to run your programs in parallel! It is also recommended that the cluster node not be specified as many students will be using it for programs which do not use MPI.

Testing Machine Availability

If a computer listed in the current machines file is assigned a task by mpirun, and that machine is for some reason not available, the startup script usually fails miserably. To verify that your machines are available, use the tstmachines command.

For example, to test the default and personal machine files:

```
/opt/mpich/gnu/sbin/tstmachines -v
/opt/mpich/gnu/sbin/tstmachines -v -machinefile=/home/abc1234/mymachines
```

The script should report no errors for on each of the machines:

```
Trying true on mpi-0-0 ...
Trying ls on mpi-0-0 ...
Trying user program on mpi-0-0 ...
```

Compiling

To compile a MPI program, use the mpicc script. This script is a preprocessor for the compiler, which adds the appropriate libraries as appropriate. As it is merely an interface to the compiler, you may need to add the appropriate –l library commands, such as –lm for the math functions. In addition, you may use –c and –o to produce object files or rename the output.

For example, to compile the test program:

```
[abc1234@rocks mpi]$ mpicc greetings.c -o greetings
```

Running MPI Programs

Use the mpirun program to execute parallel programs. The most useful argument to mpirun is –np, followed by the number of machines required for execution and the program name.

```
[abc1234@rocks mpi]$ mpirun -np 3 greetings
Greetings from process 1!
Greetings from process 2!
Process 0 of 3 on mpi-0-0 done
Process 1 of 3 on mpi-0-1 done
Process 2 of 3 on mpi-0-2 done
```

General syntax for mpirun is

mpirun [–machinefile <machinefile>] –np <np> program

Programming Notes

- All MPI programs require MPI_Init and MPI_Finalize.
- All MPI programs generally use MPI_Comm_rank and MPI_Comm_size.
- Printing debug output prefixed with the process's rank is extremely helpful.
- Printing a program initialization or termination line with the machine's name (using MPI_Get_processor_name) is also suggested.
- If you're using C++, or C with C++ features (such as declarations other than at the start of the declaration) try using mpiCC instead of mpicc.

```
//
// greetings.c
//
#include <stdio.h>
#include <string.h>
#include "mpi.h"

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

        // General identity information
        int my_rank;            // Rank of process
        int p;                  // Number of processes

        char my_name[100];      // Local processor name
        int my_name_len;        // Size of local processor name

        // Message packaging
        int source;
        int dest;
        int tag=0;
        char message[100];
        MPI_Status status;

        //
        // Start MPI
        //
        MPI_Init( &argc, &argv );

        // Get rank and size
        MPI_Comm_rank( MPI_COMM_WORLD, &my_rank );
        MPI_Comm_size( MPI_COMM_WORLD, &p );
        MPI_Get_processor_name( my_name, &my_name_len );

        if( my_rank != 0 )
        {
                // Create the message
                sprintf( message, "Greetings from process %d!", my_rank );

                // Send the message
                dest = 0;
                MPI_Send( message, strlen(message)+1, MPI_CHAR,
                        dest, tag, MPI_COMM_WORLD );
        }
        else
        {
                for( source = 1; source < p; source++ )
                {
                        MPI_Recv( message, 100, MPI_CHAR, source,
                                tag, MPI_COMM_WORLD, &status );
                        printf( "%s\n", message );

                }
        }

        // Print the closing message
        printf( "Process %d of %d on %s done\n", my_rank, p, my_name );
        MPI_Finalize();

}
```