RIT Computer Engineering Cluster

The RIT Computer Engineering cluster contains 12 computers for parallel programming using MPI. One computer: **mpssubmit.rc.rit.edu** serves as the master controller or head node for the cluster and is accessible from the Internet. The other 12 machines, named cluster-node-01, through cluster-node-12, are attached to a private LAN segment and are visible only to each other and the cluster head node.

The hardware for each cluster node consists of the following:

- Supermicro AS-1012G Server Chassis
    - Supermicro H8SGL-F
- Singe Socket 12 Core AMD Opteron 2.4GHz
- 3x 8GB PC3-8500
- 60GB Crucial SSD

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

```
<username>@mpssubmit.rc.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.

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. 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

Before using these commands, you will need to load the MPI module for use. Run the following command:

```
module load openmpi
```

Compile using: `mpicc [linking flags]`
Run programs with: `srun –n <number of tasks or processes> executable`

Specifying Machines

The cluster is currently configured to execute jobs that are sent to the scheduler SLURM (Simple Linux Utility for Resource Management). You have no access to the compute nodes directly to run your jobs. Therefore, there is no way for you to specify which machines you want the processes to run on. In summary, the scheduler SLURM will handle this for you.

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@phoenix mpi]$ mpicc greetings.c -o greetings
```

Running MPI Programs

Use the srun command to execute parallel programs. The most useful argument to `srun` is `-n`, followed by the number of processors required for execution and the program name. The following is the output of the command to run the program. Your results will vary.

```
[jml1554@cluster-node-01 MultipleProcessorSystems]$ srun -n 3 greetings
Process 2 of 3 on cluster-node-01 done
Process 1 of 3 on cluster-node-01 done
Greetings from process 1!
Greetings from process 2!
Process 0 of 3 on cluster-node-01 done
```

General syntax for `srun` is:

```
srun –n <number of tasks or processes> program
```

While this will work for the general case, it will not work for you since you don't have access to the compute nodes. This command will need to be placed in a script that is passed to the scheduler. More information about this process can be found on the course website and in the Job Submission document.

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`.

CE Cluster Scheduling

As mentioned above, the CE cluster uses a scheduler called SLURM. The purpose of SLURM is to adequately maintain the resources that are provided by the compute nodes. As you are developing your applications, you will need to be familiar with some of the basic SLURM commands that are outlined below.

sinfo – used to display the current state of the cluster

Example:

```
[jml1554@cluster-secondary ~]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
class*       up    4:00:00     10   idle cluster-node-[01-10]
```

squeue – used to display the current job queue; with the –u option, you can provide a username to view the jobs

Example:

```
[jml1554@cluster-secondary project]$ squeue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
   5748     class p1d1500c  jml1554  CG       0:00      1 cluster-node-10
   5727     class p6d15c10  jml1554   R       0:26      1 cluster-node-01
   5728     class p6d15c10  jml1554   R       0:26      1 cluster-node-01
   5729     class p6d15c10  jml1554   R       0:26      1 cluster-node-02
   5730     class p6d150c1  jml1554   R       0:26      1 cluster-node-02
   5731     class p6d1500c  jml1554   R       0:26      1 cluster-node-03
   5732     class p6d5c100  jml1554   R       0:26      1 cluster-node-03
   5733     class p6d6c100  jml1554   R       0:26      1 cluster-node-04
   5734     class p6d1000c  jml1554   R       0:25      1 cluster-node-04
   5735     class p6d1001c  jml1554   R       0:25      1 cluster-node-05
   5736     class p11d27c1  jml1554   R       0:25      1 cluster-node-06
   5737     class p11d30c1  jml1554   R       0:25      1 cluster-node-07
   5738     class p11d33c1  jml1554   R       0:25      1 cluster-node-08
   5739     class p13d15c1  jml1554   R       0:25      5 cluster-node-[05-09]
   5740     class p13d15c1  jml1554   R       0:24      2 cluster-node-[09-10]
```

sbatch – used to submit a job to the queue; a number of options can be used in two forms: one the command line, or in the script. In either case you need to use a script to submit your work. The easier of the two ways is to have the options embedded in the script as shown below. Make sure that you give your script execute permissions: `chmod +x test.sh`

Script Example: test.sh

```
#!/bin/bash

# When the #SBATCH appears at the start of a line, it will
# be interpreted by the scheduler as a command for it

# Here, we set the cluster partition to use.
#SBATCH -p kgcoe-mps

# Tell the scheduler that we want to use 13 cores for our job
#SBATCH -n 13

# Give the location of the stdout and stderr to be directed to
#SBATCH -o test.out
#SBATCH -e test.err

# Give the job a name
# You should give a unique name to each job to make it easily identifiable
#SBATCH –J Test

# Other options can be provided. Refer to the SLURM documentation for more parameters.
# SLURM: https://computing.llnl.gov/tutorials/moab/
#        https://slurm.schedmd.com/documentation.html

# You may also refer to http://mps.ce.rit.edu for more information

# Your commands go below this line

# This command MUST be in your script, otherwise the job will not run properly.
module load openmpi

# This is where you need to provide the srun command
# $SLURM_NPROCS is set by SLURM when it handles the job. This value will be equal
# to the number given to –n from above. In this case, it will be 13.
# This should NOT be changed to a number; it will ensure that you are using only
# what you neeed.
srun –n $SLURM_NPROCS greetings
```

To submit the job to SLURM, use the following command:

```
sbatch test.sh
```

You will see the following output if your job is submitted successfully:

```
Submitted batch job 5749
```

After your job completes, you can view the output from the text files test.out and/or test.err using any text editor. A fragment of the output file is provided below.

```
[jml1554@cluster-secondary MultipleProcessorSystems]$ more test.out
Process 2 of 10 on cluster-node-01 done
Process 8 of 10 on cluster-node-01 done
Process 4 of 10 on cluster-node-01 done
Process 5 of 10 on cluster-node-01 done
Process 10 of 10 on cluster-node-01 done
Greetings from process 1!
Greetings from process 2!
Process 12 of 13 on cluster-node-02 done
```

Below is a sample program which you can compile and run:

```
//
// 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();

}
```