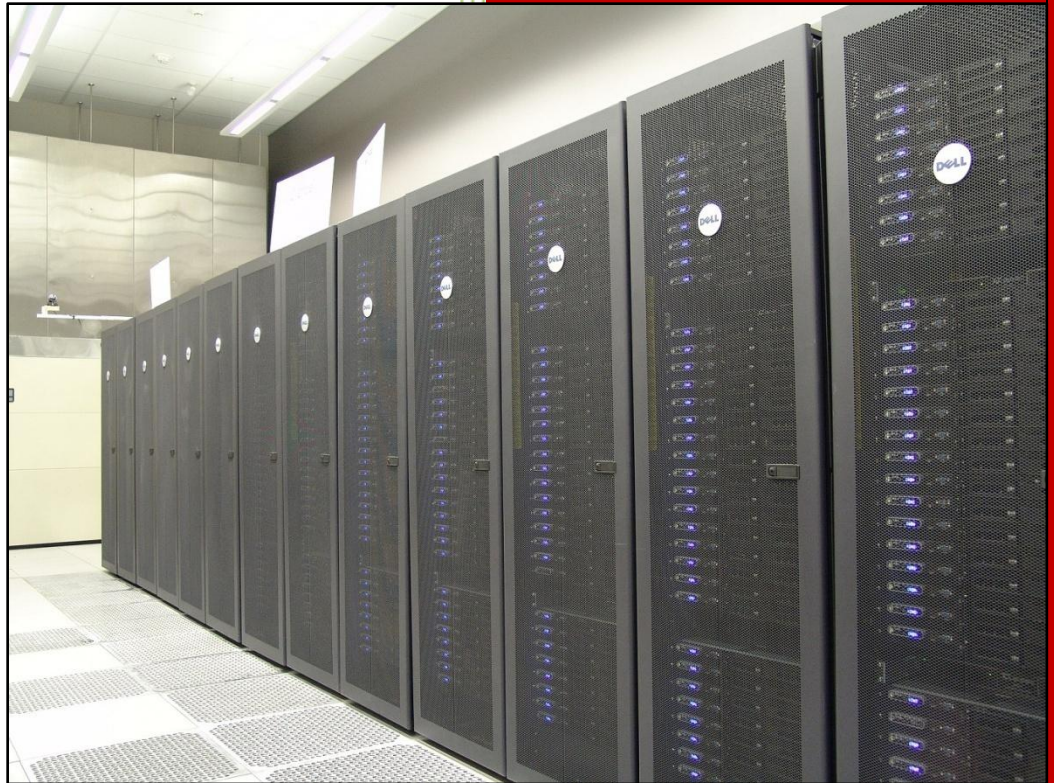


HPCC - Hrothgar

Getting Started User Guide – MPI Programming



High Performance Computing Center
Texas Tech University

Table of Contents

1. Introduction	3
2. Setting up the environment.....	3
Step 1: setting up user environment	3
Step 2: setting up MPI environment.....	4
3. Steps to execute C/Fortran program using MPI commands	5
3.1. Example - C Program.....	5
3.1.1. Source code.....	5
3.1.2. Source Code Explanation	6
3.1.3. Compiling the C Program	7
3.1.4. Job submission script	7
3.1.5. Submitting the job	9
3.2. Example – C Program for Server and clients.....	11
3.2.1. Source Code	11
3.2.2. Source Code Explanation	12
3.2.3. Compiling	12
3.2.4. Job Submission.....	12
3.3. Example - FORTRAN Program	12
3.3.1. Source Code	12
3.3.2. Compiling	13
3.3.3. Job submission	13

1. Introduction

The Message Passing Interface (MPI) aims at establishing a portable, efficient, and flexible standard for writing message passing programs [1]. Many scientific computation applications build their parallel processing modules on top of MPI, such as Gromacs, Fluent and MPIBlast. MPI can also be used in writing standalone parallel programs. This document introduces how to compile and submit MPI programs on Hrothgar, as well as some introductory knowledge of MPI.

2. Setting up the environment

For successful compiling and running MPI programs, users need to set up the required environment variables. Hrothgar is equipped with SoftEnv to set up the environment with minimum work by users. The use of SoftEnv is not required but highly recommended by HPCC staff.

Step 1: setting up user environment

If the user environment is already set up, please skip this step.

At the first use, the user should copy two sample dot-files: dot-bashrc is the start up script which evokes SoftEnv; dot-soft contains a list of software whose specific environment variables will be set up for the user.

```
$ cp /lustre/work/apps/examples/dot-bashrc .bashrc
$ cp /lustre/work/apps/examples/dot-soft .soft
$ ln -s .bashrc .bash_profile
```

Log out and log in again.

```

1:grendel.hpcc.ttu.edu - default - SSH Secure Shell
Quick Connect Profiles
File Edit View Window Help

-bash-3.2$ pwd
/home/pmane
-bash-3.2$ ls -al
total 60
drwxr-xr-x  4 pmane CS   4096 Jul 13 13:36 .
drwxr-xr-x 430 root  root 12288 Jul 12 08:35 ..
-rw-----  1 pmane CS  15117 Jul 13 12:16 .bash_history
drwxr-xr-x 12 root  root  4096 Jul 13 11:47 examples
-rw-----  1 pmane CS   1675 Jul  2 2009 pmane_id
-rw-r--r--  1 pmane CS    408 Jul  2 2009 pmane_id.pub
drwx-----  2 pmane CS   4096 Jul  6 2009 .ssh
-rw-----  1 pmane CS   5783 Jun  8 16:49 .viminfo
-rw-----  1 pmane CS    192 May  5 20:15 .Xauthority
-bash-3.2$ cp /lustre/work/apps/examples/dot-soft .soft
-bash-3.2$ cp /lustre/work/apps/examples/dot-bashrc .bashrc
-bash-3.2$ ln -s .bashrc .bash_profile
-bash-3.2$ ls -al
total 64
drwxr-xr-x  4 pmane CS   4096 Jul 13 13:45 .
drwxr-xr-x 430 root  root 12288 Jul 12 08:35 ..
-rw-----  1 pmane CS  15117 Jul 13 12:16 .bash_history
lrwxrwxrwx  1 pmane CS      7 Jul 13 13:45 .bash_profile -> .bashrc
-rwxr-xr-x  1 pmane CS    522 Jul 13 13:44 .bashrc
drwxr-xr-x 12 root  root  4096 Jul 13 11:47 examples
-rw-----  1 pmane CS   1675 Jul  2 2009 pmane_id
-rw-r--r--  1 pmane CS    408 Jul  2 2009 pmane_id.pub
-rwxr-xr-x  1 pmane CS    226 Jul 13 13:44 .soft
drwx-----  2 pmane CS   4096 Jul  6 2009 .ssh
-rw-----  1 pmane CS   5783 Jun  8 16:49 .viminfo
-rw-----  1 pmane CS    192 May  5 20:15 .Xauthority
-bash-3.2$

```

Connected to grendel.hpcc.ttu.edu SSH2 - aes128-cbc - hmac-md5 - n 80x31

Step 2: setting up MPI environment

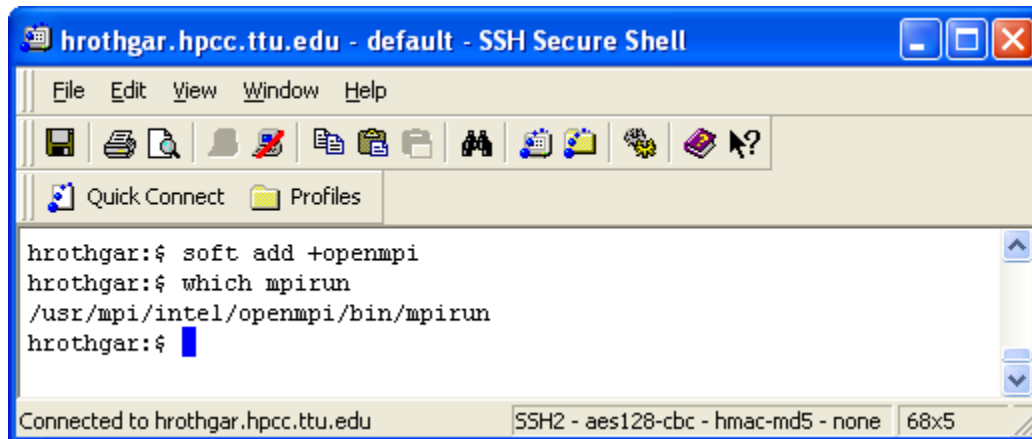
The latest version of MPI facility installed on Hrothgar is OpenMPI-1.4.2. To use it, use the following command:

```
$ soft add +openmpi
```

To check whether the environment is set to the correct MPI version, use the following command:

```
$ which mpirun
```

The command should return /usr/mpi/intel/openmpi/bin/mpirun, as shown in the screen shot.



```
hrothgar:$ soft add +openmpi
hrothgar:$ which mpirun
/usr/mpi/intel/openmpi/bin/mpirun
hrothgar:$
```

3. Steps to execute C/Fortran program using MPI commands

Generally, there are 4 steps required for the MPI C/ Fortran program to be executed.

Step1: Writing the source code in a <filename>.c file (C program) or <filename>.f90 (Fortran program)

Step2: Compiling the source codes to make an executable using command mpicc (C program) or mpif90 (Fortran program)

Step3: Copying the mpi script for job submission at appropriate location and making changes to this script according to the requirement

Step4: Submitting the MPI job

3.1. Example - C Program

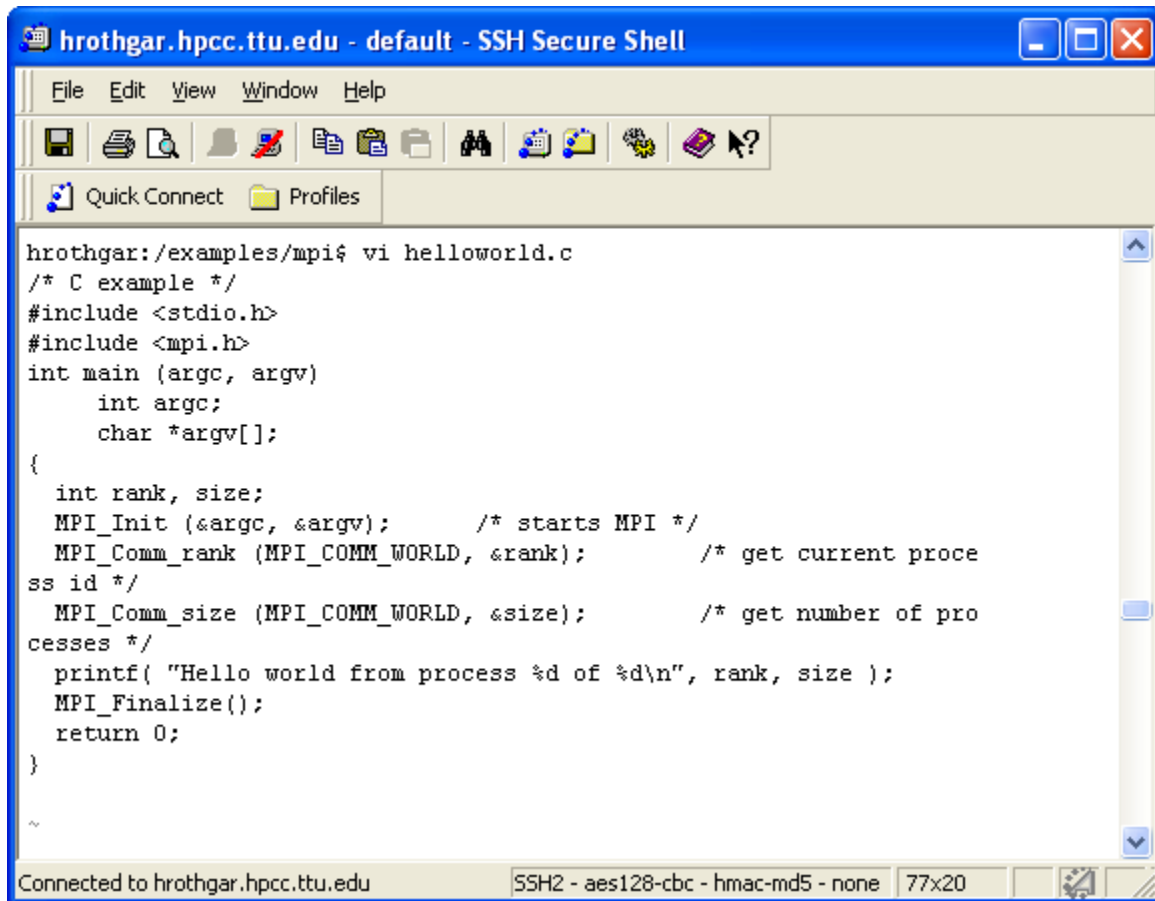
3.1.1. Source code

This code comes from reference [3]. In this program, each process returns a greeting “Hello world” followed by its rank. See Section 3.1.2 for detailed explanation.

```
/* C Example */
#include <stdio.h>
#include <mpi.h>
int main (argc, argv)
int argc;
char *argv[];
{
int rank, size;
MPI_Init (&argc, &argv); /* starts MPI */
MPI_Comm_rank (MPI_COMM_WORLD, &rank); /* get current process id */
MPI_Comm_size (MPI_COMM_WORLD, &size); /* get number of processes */
printf( "Hello world from process %d of %d\n", rank, size );
MPI_Finalize();
return 0;
}
```

Open any editor (vi/emacs/gedit) to try this program:

\$ **vi helloworld.c** and copy paste the above code, as seen in the below screen shot.



```
hrothgar.hpcc.ttu.edu - default - SSH Secure Shell
File Edit View Window Help
[Icons]
Quick Connect Profiles

hrothgar:/examples/mpi$ vi helloworld.c
/* C example */
#include <stdio.h>
#include <mpi.h>
int main (argc, argv)
    int argc;
    char *argv[];
{
    int rank, size;
    MPI_Init (&argc, &argv);      /* starts MPI */
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);      /* get current process id */
    MPI_Comm_size (MPI_COMM_WORLD, &size);      /* get number of processes */
    printf( "Hello world from process %d of %d\n", rank, size );
    MPI_Finalize();
    return 0;
}
~

Connected to hrothgar.hpcc.ttu.edu  SSH2 - aes128-cbc - hmac-md5 - none  77x20
```

3.1.2. Source Code Explanation

This is the simplest MPI program, but it contains the essential functions of all MPI programs: MPI_Init, MPI_Comm_rank, MPI_Comm_size and MPI_Finalize [1].

MPI_Init initializes the MPI execution environment. This function must be called in every MPI program, must be called before any other MPI functions and must be called only once in an MPI program.

MPI_Comm_rank determines the rank of the calling process within the communicator. A process' rank is between 0 and number of processes - 1, and often referred to as a task ID. It is stored in the variable "rank".

MPI_Comm_size determines the number of processes in the group associated with a communicator. The number of processes is stored in the variable "size".

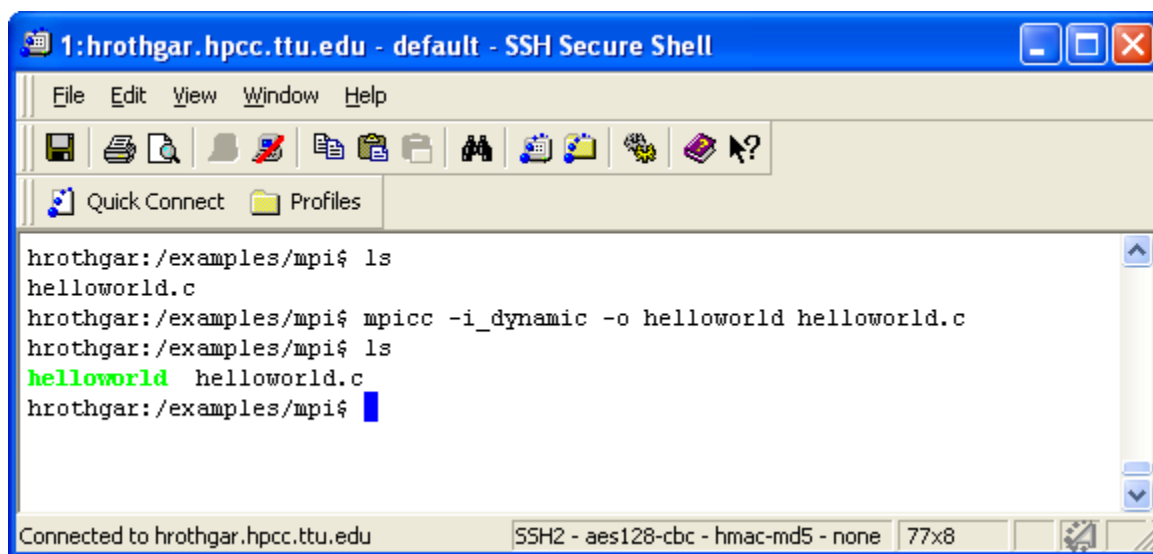
MPI_Finalize terminates the MPI execution environment. This function should be the last MPI routine called in every MPI program - no other MPI routines may be called after it.

The above functions should be written to any MPI programs. The main part is usually between the initialization functions and `MPI_Finalize()`. In this program, each process prints "Hello world" followed by the rank of this process. This function is implemented by the line `printf("Hello world from process %d of %d\n", rank, size);`

3.1.3. Compiling the C Program

The compiling program of C version of MPI program is `mpicc`. The command to compile the file *helloworld.c* to the executable *helloworld* is:

```
$ mpicc -i_dynamic -o helloworld helloworld.c
```



The screenshot shows an SSH terminal window titled "1:hrothgar.hpcc.ttu.edu - default - SSH Secure Shell". The terminal displays the following commands and output:

```
hrothgar:/examples/mpi$ ls
helloworld.c
hrothgar:/examples/mpi$ mpicc -i_dynamic -o helloworld helloworld.c
hrothgar:/examples/mpi$ ls
helloworld helloworld.c
hrothgar:/examples/mpi$
```

The status bar at the bottom indicates the connection details: "Connected to hrothgar.hpcc.ttu.edu", "SSH2 - aes128-cbc - hmac-md5 - none", and "77x8".

After compiling, an executable called *helloworld* is generated.

Note: If you forget `-i_dynamic` switch it will throw warning saying `"/opt/intel/Compiler/11.1/064/lib/intel64/libimf.so: warning: warning: feupdateenv is not implemented and will always fail"`.

3.1.4. Job submission script

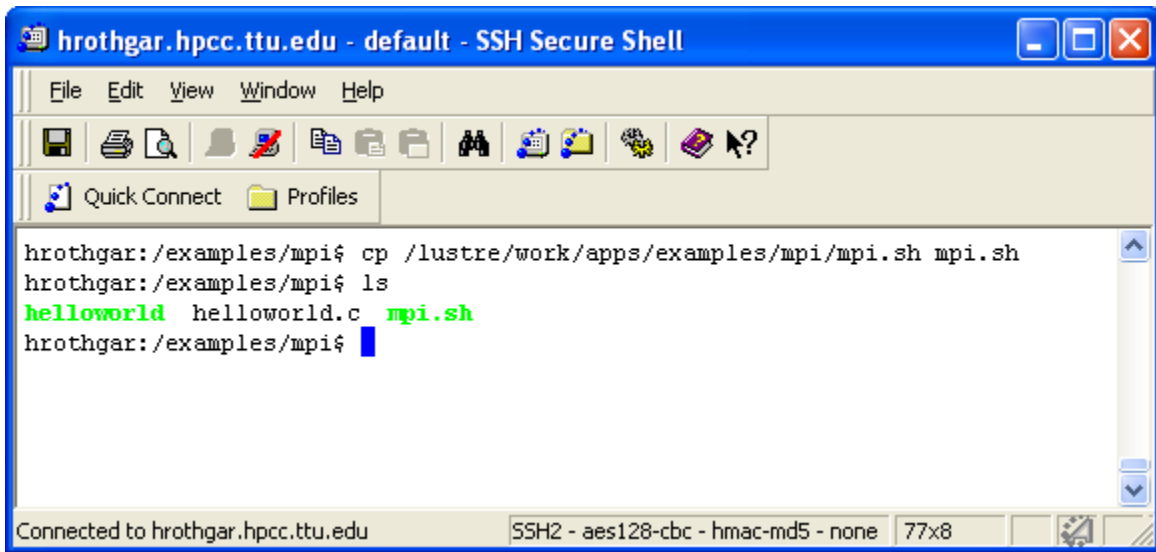
On Hrothgar, users should always submit jobs to Sun Grid Engine. Sun Grid Engine will schedule the jobs to free nodes in the cluster for execution. The following is a script file to submit the "helloworld" program to 12 cores. It is also available at `/lustre/work/apps/examples/mpi/mpi.sh`. Use the command to copy it:

```
$ cp -r /lustre/work/apps/examples/mpi/mpi.sh mpi.sh
```



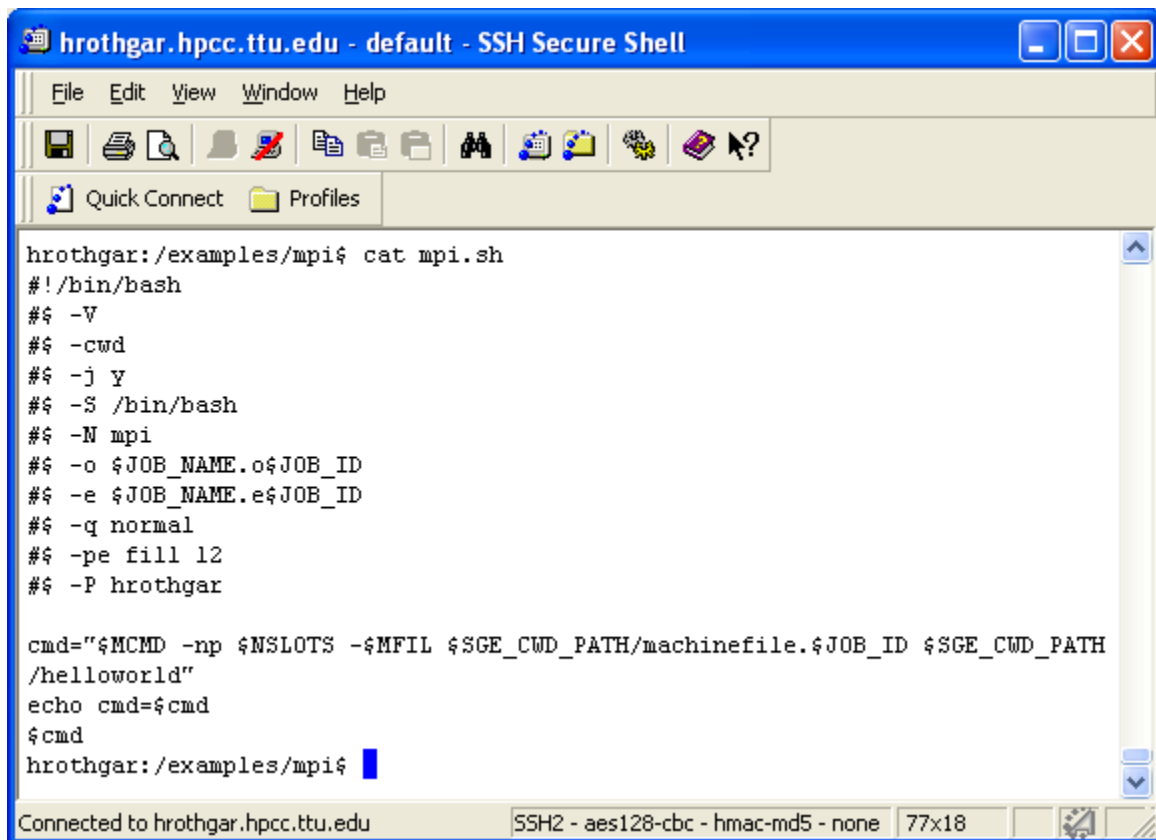
```
#!/bin/bash
#$ -V
#$ -cwd
#$ -j y
#$ -S /bin/bash
#$ -N mpi
#$ -o $JOB_NAME.o$JOB_ID
#$ -e $JOB_NAME.e$JOB_ID
#$ -q normal
#$ -pe fill 12
#$ -P hrothgar

cmd="$MCMD -np $NSLOTS -xMFIL $SGE_CWD_PATH/machinefile.$JOB_ID
$SGE_CWD_PATH/helloworld"
echo cmd=$cmd
$cmd
```



```
hrothgar:/examples/mpi$ cp /lustre/work/apps/examples/mpi/mpi.sh mpi.sh
hrothgar:/examples/mpi$ ls
helloworld helloworld.c mpi.sh
hrothgar:/examples/mpi$
```

The lines starting with “#\$” are Sun Grid Engine options. # \$ -N mpi means that the name of this job is “mpi”, and the name can be referred as \$JOB_NAME; # \$ -o \$JOB_NAME.o\$JOB_ID and # \$ -e \$JOB_NAME.e\$JOB_ID set the standard output file and error output file, where \$JOB_NAME refers to the name of the job, and \$JOB_ID refers to the job’s ID (a number representing the job) in the cluster when the job is submitted; # \$ -q normal tells the scheduler to submit the job to the “normal” queue; # \$ -pe fill 12 tells the scheduler to assign 12 cores to this job.



```
hrothgar:/examples/mpi$ cat mpi.sh
#!/bin/bash
# -V
# -cwd
# -j Y
# -S /bin/bash
# -N mpi
# -o $JOB_NAME.o$JOB_ID
# -e $JOB_NAME.e$JOB_ID
# -q normal
# -pe fill 12
# -P hrothgar

cmd="$MCMD -np $NSLOTS -m $MFIL $SGE_CWD_PATH/machinefile.$JOB_ID $SGE_CWD_PATH
/helloworld"
echo cmd=$cmd
$cmd
hrothgar:/examples/mpi$
```

3.1.5. Submitting the job

For submitting the jobs to the compute nodes the command is:

```
$ qsub mpi.sh
```

Users can check the status of the job by the command:

```
$ qstat
```

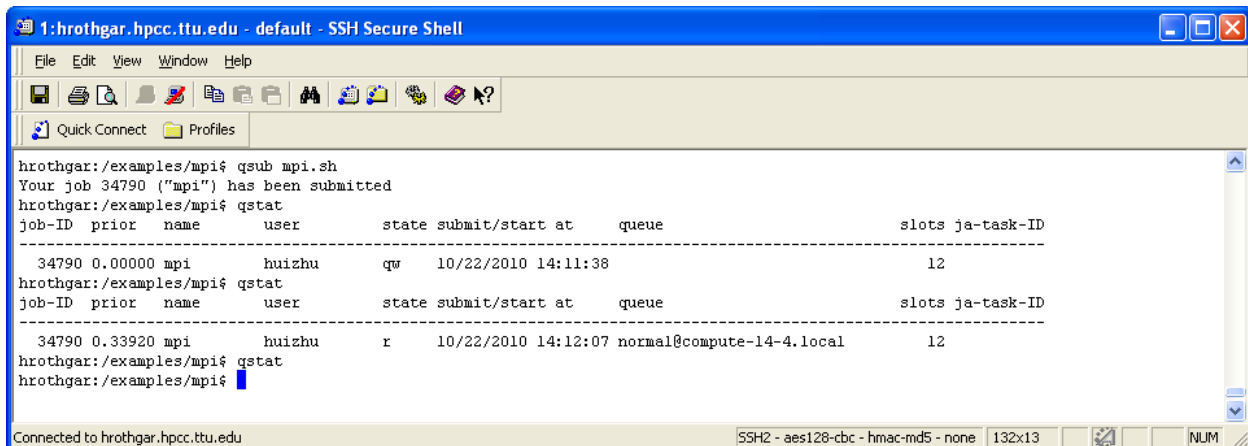
Users can monitor the status of the job by watch command:

```
$ watch qstat
```

By default, “watch qstat” command calls the “qstat” command every two seconds, and show the results on the screen.

After the job is finished, qstat commands returns nothing. Open the mpi.o<\$JOB_ID> file to view the result.

The following is the screen shot of the procedure:



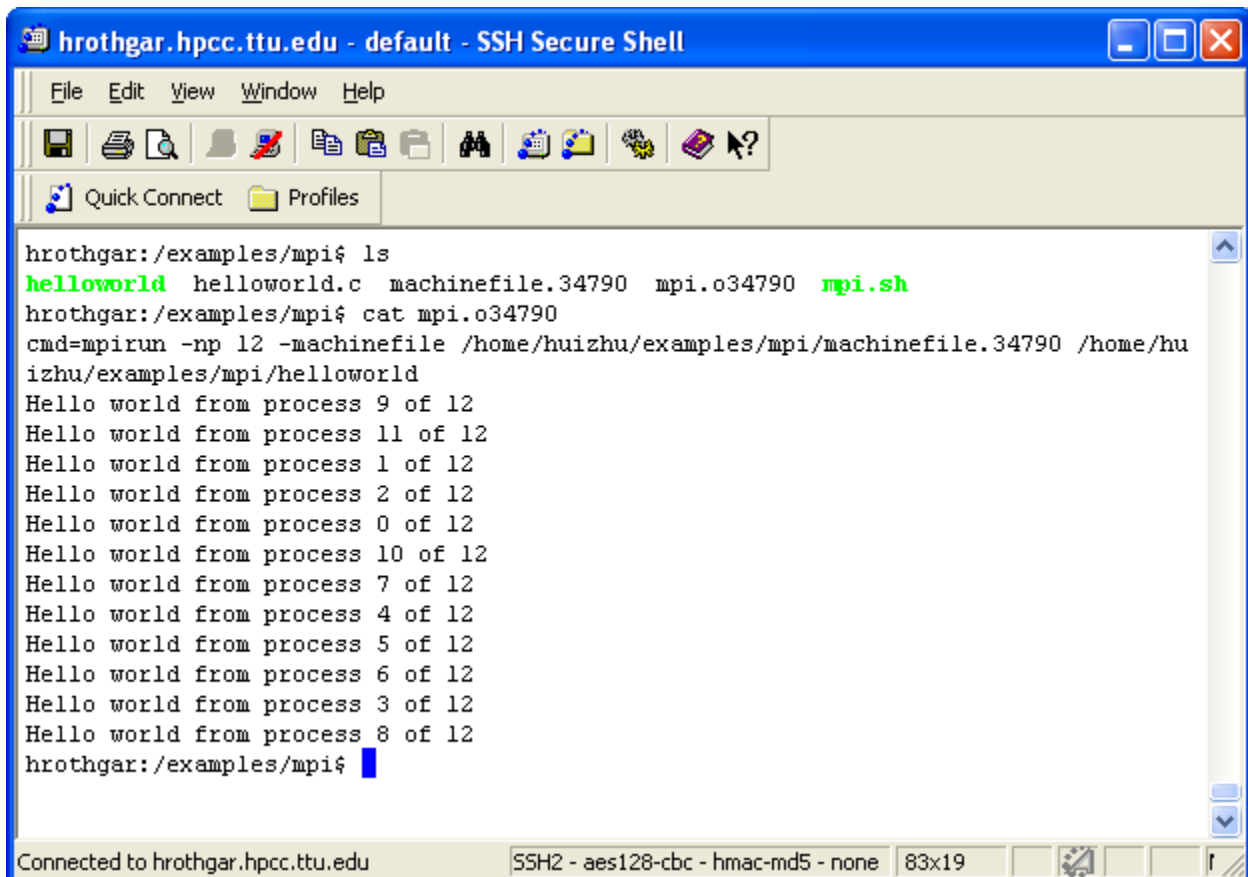
```
1:hrothgar.hpcc.ttu.edu - default - SSH Secure Shell
File Edit View Window Help
Quick Connect Profiles

hrothgar:/examples/mpi$ qsub mpi.sh
Your job 34790 ("mpi") has been submitted
hrothgar:/examples/mpi$ qstat
job-ID prior name user state submit/start at queue slots ja-task-ID
-----
34790 0.00000 mpi huizhu qw 10/22/2010 14:11:38 12
hrothgar:/examples/mpi$ qstat
job-ID prior name user state submit/start at queue slots ja-task-ID
-----
34790 0.33920 mpi huizhu r 10/22/2010 14:12:07 normal@compute-14-4.local 12
hrothgar:/examples/mpi$ qstat
hrothgar:/examples/mpi$
```

Connected to hrothgar.hpcc.ttu.edu SSH2 - aes128-cbc - hmac-md5 - none 132x13

In the above screen shot, the output of the first `qstat` command indicate that the job is in “qw” state, which means it is waiting in the queue. The second `qstat` command indicates that the job is in “r” state, which means it is running. The third `qstat` command returns nothing, which means the job finished.

The result will be in `mpi.o<JOB_ID>`, can shown in the below screen shot for this particular example



```
hrothgar.hpcc.ttu.edu - default - SSH Secure Shell
File Edit View Window Help
Quick Connect Profiles

hrothgar:/examples/mpi$ ls
helloworld helloworld.c machinefile.34790 mpi.o34790 mpi.sh
hrothgar:/examples/mpi$ cat mpi.o34790
cmd=mpirun -np 12 -machinefile /home/huizhu/examples/mpi/machinefile.34790 /home/huizhu/examples/mpi/helloworld
Hello world from process 9 of 12
Hello world from process 11 of 12
Hello world from process 1 of 12
Hello world from process 2 of 12
Hello world from process 0 of 12
Hello world from process 10 of 12
Hello world from process 7 of 12
Hello world from process 4 of 12
Hello world from process 5 of 12
Hello world from process 6 of 12
Hello world from process 3 of 12
Hello world from process 8 of 12
hrothgar:/examples/mpi$
```

Connected to hrothgar.hpcc.ttu.edu SSH2 - aes128-cbc - hmac-md5 - none 83x19

3.2. Example – C Program for Server and clients

This is another “Hello World” program that involves a server process and several client processes. This code is slightly modified from reference [2].

3.2.1. Source Code

```
#include <stdio.h>
#include <string.h>
#include "mpi.h"
int main (int argc, char* argv[])
{ /* main */
    const int maximum_message_length = 100;
    const int server_rank= 0;
    char message[maximum_message_length+1];
    MPI_Status status; /* Info about receive status */
    int my_rank; /* This process ID */
    int num_procs; /* Number of processes in run */
    int source; /* Process ID to receive from */
    int destination; /* Process ID to send to */
    int tag = 0; /* Message ID */
    int mpi_error_code; /* Error code for MPI calls */
    mpi_error_code= MPI_Init(&argc, &argv);
    mpi_error_code= MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    mpi_error_code= MPI_Comm_size(MPI_COMM_WORLD, &num_procs);

    /* clients processes */
    if (my_rank != server_rank) {
        sprintf(message, "Hello world from process # %d!", my_rank);
        destination = server_rank;
        mpi_error_code= MPI_Send(message, strlen(message) + 1, MPI_CHAR, destination,
        tag, MPI_COMM_WORLD);
    } /* if (my_rank != server_rank) */

    /* server process */
    else {
        for (source = 0; source < num_procs; source++) {
            if (source != server_rank) {
                mpi_error_code= MPI_Recv(message, maximum_message_length + 1,
                MPI_CHAR, source, tag, MPI_COMM_WORLD, &status);
                fprintf(stderr, "%s\n", message);
            } /* if (source != server_rank) */
        } /* for source */
    } /* if (my_rank != server_rank)...else */
    mpi_error_code= MPI_Finalize();
} /* main */
```

3.2.2. Source Code Explanation

The program in Section 3.2.1 assigns the process with rank 0 as the server, and other processes as clients. The server and the clients' complete different tasks, and their tasks are distinguished by checking whether the process' rank equals 0.

Besides the four essential functions mentioned in the last section, there are two more widely used communication routines in this program, MPI_Send and MPI_Recv. Both of them are blocking routines, which return after the entire message has been sent or received.

In this program, each client sends a "Hello World" message to the server by calling the routine: MPI_Send (message, strlen(message)+1, MPI_CHAR, destination, tag, MPI_COMM_WORLD), where "message" is a string of message; "strlen(message) + 1" is the length of the message; "MPI_CHAR" indicates the type of this message is character; "destination" is the server process in this program; "tag" is used to distinguish messages to or from the same process. In this particular case, each client only sends one message, and the messages received by the server are from different clients each other, so it is unnecessary to distinguish the messages by the tag.

At the server side, the server process receive messages from the clients by calling MPI_Recv (message, maximum_message_length+1, MPI_CHAR, source, tag, MPI_COMM_WORLD, &status) in a "for" loop, and "source" is assigned to each client's rank in each iteration. After the server receives all messages from the clients, it prints them out.

3.2.3. Compiling

The compiling of the source code is the same as in Section 3.1.3.

3.2.4. Job Submission

The job submission of the source code is the same as in Section 3.1.4 and Section 3.1.5.

3.3. Example - FORTRAN Program

3.3.1. Source Code

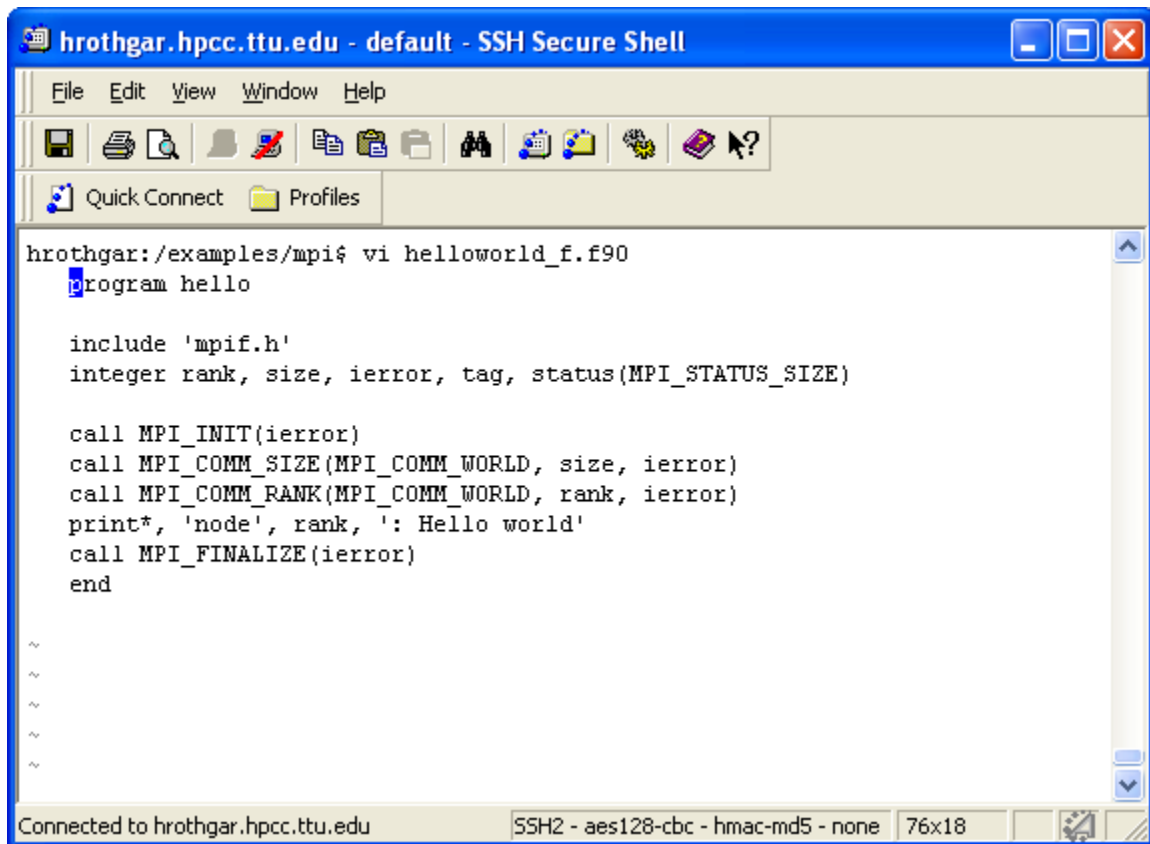
This code comes from reference [3]. Name the code as **helloworld_f.f90**

```
program hello
  include 'mpif.h'
  integer rank, size, ierror, tag, status(MPI_STATUS_SIZE)

  call MPI_INIT(ierror)
  call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierror)
  call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierror)
  print*, 'node', rank, ': Hello world'
  call MPI_FINALIZE(ierror)
end
```

Open any editor (vi/emacs/gedit) to try this program:

\$ *vi helloworld_f.f90* and copy paste the above code, as seen in the below screen shot.



The screenshot shows a terminal window titled "hrothgar.hpcc.ttu.edu - default - SSH Secure Shell". The window has a menu bar (File, Edit, View, Window, Help) and a toolbar with various icons. Below the toolbar, there are tabs for "Quick Connect" and "Profiles". The main text area displays the following Fortran code being edited in the vi editor:

```
hrothgar:/examples/mpi$ vi helloworld_f.f90
program hello

  include 'mpif.h'
  integer rank, size, ierror, tag, status(MPI_STATUS_SIZE)

  call MPI_INIT(ierror)
  call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierror)
  call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierror)
  print*, 'node', rank, ': Hello world'
  call MPI_FINALIZE(ierror)
end
```

Below the code, there are several tilde (~) characters indicating the end of the file. The status bar at the bottom of the window shows "Connected to hrothgar.hpcc.ttu.edu", "SSH2 - aes128-cbc - hmac-md5 - none", and "76x18".

3.3.2. Compiling

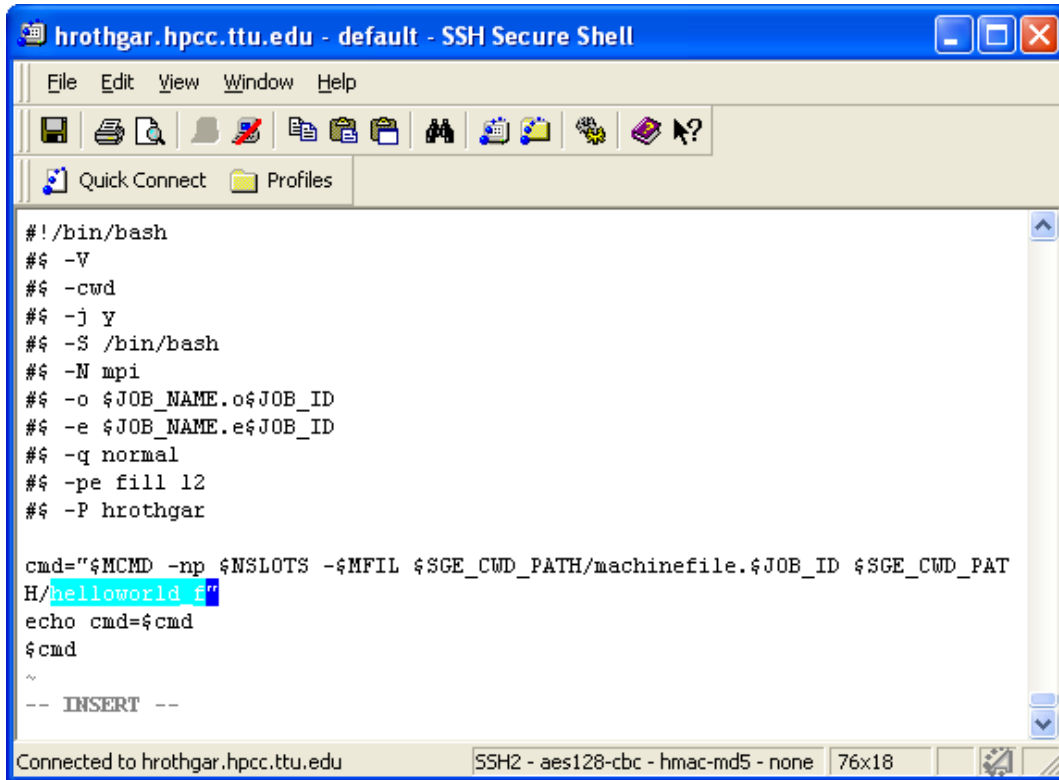
\$ *mpif90 -i_dynamic -o helloworld_f helloworld_f.f90*

This command compiles the file *helloworld_f.f90* to the executable *helloworld_f*.

3.3.3. Job submission

The job submission procedure for the Fortran program is similar as in the Section 3.1.4 and 3.1.5 (C program). The only difference in this Fortran example is that the name of the executable is *helloworld_f* rather than *helloworld*.

Snapshot of making changes to the mpi.sh script



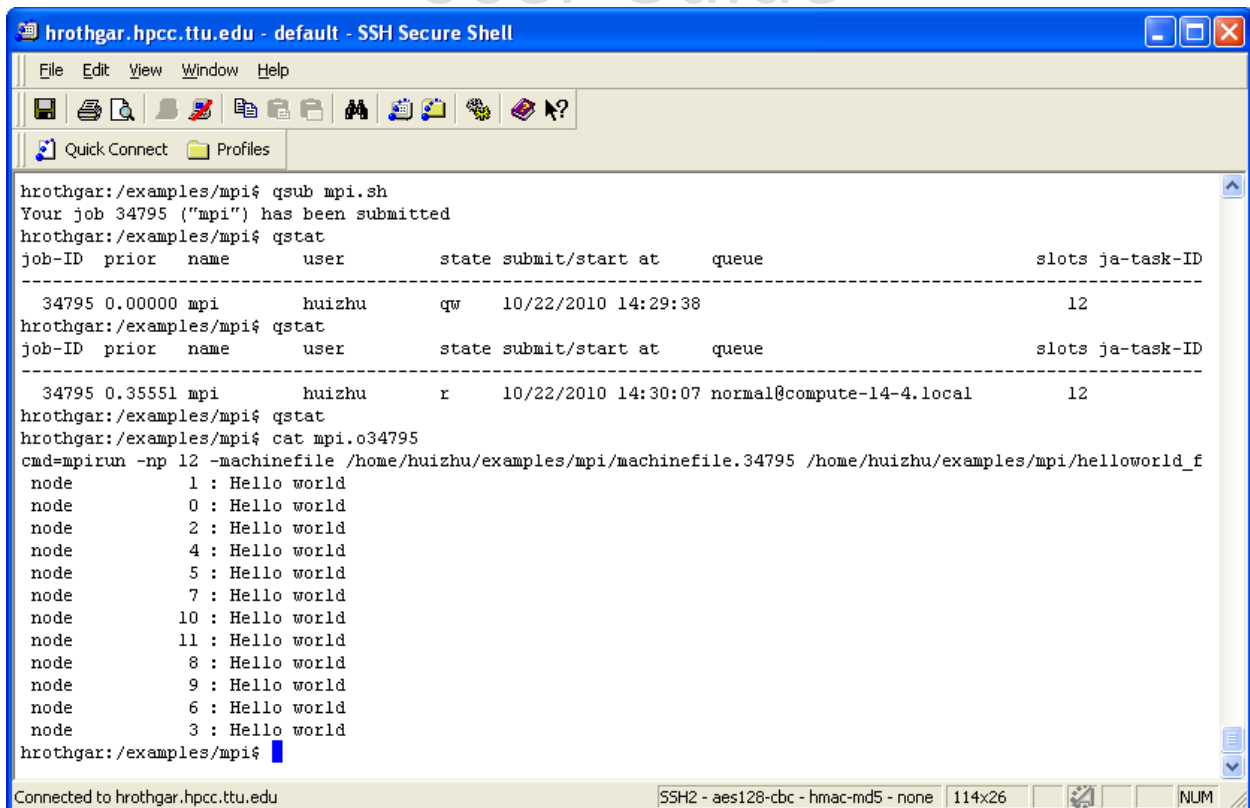
```
hrothgar.hpcc.ttu.edu - default - SSH Secure Shell
File Edit View Window Help
Quick Connect Profiles

#!/bin/bash
# -V
# -cwd
# -j Y
# -S /bin/bash
# -N mpi
# -o $JOB_NAME.o$JOB_ID
# -e $JOB_NAME.e$JOB_ID
# -q normal
# -pe fill 12
# -P hrothgar

cmd="$MCMD -np $NSLOTS -MFIL $SGE_CWD_PATH/machinefile.$JOB_ID $SGE_CWD_PATH/helloworld_f"
echo cmd=$cmd
$cmd
~
-- INSERT --

Connected to hrothgar.hpcc.ttu.edu  SSH2 - aes128-cbc - hmac-md5 - none 76x18
```

Snapshot of submitting the Fortran job and checking the output



```
hrothgar.hpcc.ttu.edu - default - SSH Secure Shell
File Edit View Window Help
Quick Connect Profiles

hrothgar:/examples/mpi$ qsub mpi.sh
Your job 34795 ("mpi") has been submitted
hrothgar:/examples/mpi$ qstat
job-ID prior name user state submit/start at queue slots ja-task-ID
-----
34795 0.00000 mpi huizhu qw 10/22/2010 14:29:38 12
hrothgar:/examples/mpi$ qstat
job-ID prior name user state submit/start at queue slots ja-task-ID
-----
34795 0.35551 mpi huizhu r 10/22/2010 14:30:07 normal@compute-14-4.local 12
hrothgar:/examples/mpi$ cat mpi.o34795
cmd=mpirun -np 12 -machinefile /home/huizhu/examples/mpi/machinefile.34795 /home/huizhu/examples/mpi/helloworld_f
node 1 : Hello world
node 0 : Hello world
node 2 : Hello world
node 4 : Hello world
node 5 : Hello world
node 7 : Hello world
node 10 : Hello world
node 11 : Hello world
node 8 : Hello world
node 9 : Hello world
node 6 : Hello world
node 3 : Hello world
hrothgar:/examples/mpi$

Connected to hrothgar.hpcc.ttu.edu  SSH2 - aes128-cbc - hmac-md5 - none 114x26 NUM
```

4. References

- [1] *Blaise Barney*, Message Passing Interface (MPI), <https://computing.llnl.gov/tutorials/mpi/>
- [2] *Henry Neeman*, Supercomputing in Plain English Part VI: Distributed Multiprocessing, http://www.oscer.ou.edu/Workshops/DistributedParallelism/sipe_distribmem_20090324.ppt
- [3] Hello World MPI Examples, http://www.dartmouth.edu/~rc/classes/intro_mpi/hello_world_ex.html

User Guide

Last updated: 10/22/2010

For Additional Assistance Contact: hpccsupport@ttu.edu

For Comments/Suggestions on user guide hpcc@ttu.edu

User Guide