THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE
Department of Electrical and Computer Engineering

CPE 412/512
Fall Semester 2015

Homework Assignment Number 1

1) Implement a MPI version, PThreads version, and an OpenMP version of a “hello world” type program on the Alabama Supercomputer Authority’s Dense Memory Cluster, DMC, system (dmc.asc.edu). Successfully compile and execute these programs and capture the output of these files. Include this output as part of the homework report
(a) Answer the question as to whether the outputs always appear the same each time the program is executed and speculate on why or why not this is the case.
(b) What happens when the Pthread version of the program is run when the *pthread_mutex_lock(&MUTEX)* and the *pthread_mutex_unlock(&MUTEX)* statements are commented out or removed? If the program behaved differently explain why this was the case.
(c) What happens when the OpenMP version of the program is run when the *#pragma omp critical* statements are removed? If the program behaved differently explain why you believe this was the case.

These files can be found on Canvas on the Homework 1 assignment page and under the ~/share directory on the DMC.

2) Create a hybrid MPI/PThread or a hybrid MPI/OpenMP version of the “hello world” program where each MPI process produces two threads each that outputs a “Hello World from MPI Process ?? Thread zz” message. Illustrate how the program works for a 4 process 2 thread per process version running in the interactive mode on the ASC DMC system (dmc.asc.edu).

3) Follow the program turn-in procedure outline below. *Due date all students Friday September 4, 2015 11:59 PM,*

<u>Homework Assignment Turn in Procedure</u>

You are to upload a single document file on the HW1 Assignment on the UAH Course Canvas^{TM} site on or before its due date. This submission should contain the source code and the resulting output that was produced by the actual execution of the program. Acceptable file formats include pdf, doc,docx, and ascii text.

Alabama Supercomputer Authority Temporary Class Account Information

We have made an account for you on the machines at the Alabama Supercomputer Center. These systems are an SGI Ultraviolet, and a locally architected fat node cluster called a Dense Memory Cluster (DMC).
Below is some information for new users of these machines. You can login, following the directions below, using the login and password provided by your instructor.

WARNING: This is a temporary account. The account and all the files in it will be deleted at the end of the semester.

The best source of information on using these computers is the HPC User Manual, which can be accessed on the web at http://www.asc.edu/html/man.pdf

The command ?ascdocs? shows a menu drive listing of locally written documentation. It displays text files, and gives paths for you to copy or download other types of files.

The computers can be accessed using secure shell (an encrypted form of telnet). Secure shell is installed on many Linux and Unix machines, and can be called with a command like this

```
     ssh UserID@dmc.asc.edu
```

The first time you login, you will have to enter the password a second time, then set a new password.

Files can be copied to and from the DMC using secure copy "scp" or secure ftp "sftp". These programs are often included with secure shell software.

You can get help from the Alabama Supercomputer Center staff at hpc@asc.edu or (256) 971-7448 or (800) 338-8320. The staff can help with using the applications on the computers, accessing the system, etc.

In order to get the maximum possible utilization of the computers, computational jobs are executed through the Moab job queueing system. Moab can be accessed via PBS commands like "qstat" to check queue status.

Scripts are provided for more easily running jobs using many of the third party software packages. Most of these scripts are in the /apps/scripts directory and have names starting with "run". For example, a Gaussian 09 job can be submitted to a queue by typing

```
     rung09 input_file_name
```

The queue script will prompt you for the name of the queue and any limits you may wish to set on time, memory or file size. It then prompts for a date and time to run the job, job name, and output directory.
All prompts can be satisfied by pressing "Enter" to give default values of the queue limits, immediate submittal, a default job name, and results being returned to the current directory. Only class accounts can submit jobs to the class queue.

Accounts have a disk quota limit. To see your disk quota, type "quota". Queue limits and interactive use limits can be seen by typing "qlimits".

Please contact the Alabama Supercomputer Center staff if you have any other questions or concerns about using the supercomputers.

David Young, Ph.D.
HPC Computational Specialist, CSC
dyoung@asc.edu
(256) 971-7434

Alabama Supercomputer Center
Alabama Research and Education Network
http://www.asc.edu/
FAX: (256) 971-7491

```
/*
   MPI Example - Hello World - C++ Version (utilizing C function calling Conventions)
   FILE: hello_world_MPI.cpp

   Compilation on dmc.asc.edu

   first set up environment by typing from the command line

      module load openmpi

   to compile the program type

      mpic++ hello_world_MPI.cpp -o hello_world_MPI

   to run on eight processors type

      mpirun -np 8 ./hello_world_MPI
*/

using namespace std;
#include <iostream>
#include <mpi.h>

int main (int argc, char *argv[]) {
   MPI_Status status;
   int nmtsks, rank;

   MPI_Init(&argc,&argv); // Initalize MPI environment
   MPI_Comm_size(MPI_COMM_WORLD,&nmtsks); //get total number of processes
   MPI_Comm_rank(MPI_COMM_WORLD,&rank); // get process identity number

   cout << "Hello World from MPI Process #" << rank << endl << flush;

   /* Only root MPI process does this */
   if (rank == 0) {
     cout << "Number of MPI Processes = " << nmtsks << endl;
   }

  /* Terminate MPI Program -- clear out all buffers */
  MPI_Finalize();

}
```

```
/*
   MPI Example - Hello World - C++ Version
   using the full C++ syntax
   FILE: hello_world_MPI2.cpp

   Compilation on dmc.asc.edu

   first set up environment by typing from the command line

      module load openmpi

   to compile the program type

      mpic++ hello_world_MPI2.cpp -o hello_world_MPI2

   to run on eight processors type

      mpirun -np 8 ./hello_world_MPI2
*/

using namespace std;
#include <iostream>
#include <mpi.h>

int main (int argc, char *argv[]) {
   MPI_Status status;
   int nmtsks, rank;

   MPI::Init(argc,argv); // Initalize MPI environment

   nmtsks=MPI::COMM_WORLD.Get_size(); //get total number of processes

   rank= MPI::COMM_WORLD.Get_rank(); // get process identity number

   cout << "Hello World from MPI Process #" << rank << endl << flush;

   /* Only root MPI process does this */
   if (rank == 0) {
     cout << "Number of MPI Processes = " << nmtsks << endl;
   }

  /* Terminate MPI Program -- clear out all buffers */
  MPI::Finalize();

}
```

```
/*
  Pthreads Example - Hello World - C/C++ Version
File: hello_world_PTH.cpp

  first set up environment by typing from the command line the
  following two module load commands
   module load intel

  Compilation on dmc.asc.edu
     to compile the program
         icc hello_world_PTH.cpp -o hello_world_PTH -lpthread
     to run type
       ./hello_world_PTH
*/

using namespace std;
#include <iostream>

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>
#define NTHREADS 8 /* set number of threads to 8 */

pthread_mutex_t MUTEX; // globaly defined lock variable used to insure
                       // uninterrupted print operations from each thread

// This is the worker thread code
void *hello(void * arg) {
   int myid=*(int *) arg;

   // enter critical region
   pthread_mutex_lock(&MUTEX);

   cout << "Hello World, from PThread " << myid << endl;

   // exit critical region
   pthread_mutex_unlock(&MUTEX);

  return arg;
}
// this is the main thread's code -- it spawns the worker threads and
// then waits for all the worker threads to return before it exits
int main(int argc,char *argv[])
{
  int tid;                        // tid thread number
  pthread_t threads[NTHREADS];    // holds various thread info
  int ids[NTHREADS];              // holds thread args
  int errcode;                    // pthread error code

   // initialize mutex variable -- this variable is used to insure that
   // all couts are automic meaning that they are not interrupted
   pthread_mutex_init(&MUTEX,NULL);
   /* create the threads */
   for (tid=0; tid<NTHREADS; tid++) {
      ids[tid]=tid;
      errcode=pthread_create(
              &threads[tid],// thread information structure
              NULL,         // thread attributes -- NULL means assume defaults
              hello,        // function name that is to represent thread
              &ids[tid]);  // pthread created thread id for the created thread
      // check for error during thread creation
      if (errcode) {
         cerr << "Pthread Creation Error: " << strerror(errcode) << endl;
         exit(1);
      }
   }

   // print out number of threads
   pthread_mutex_lock(&MUTEX); // enter critical region
   cout << "Number of threads = " << NTHREADS << endl;
   pthread_mutex_unlock(&MUTEX); // exit critical region

   // wait for all threads to return before exiting main program (process)
   for (tid=0; tid<NTHREADS; tid++) {
      // wait for each thread to terminate
      errcode=pthread_join(
              threads[tid], //thread scheduler id information of selected thread
              NULL);        //thread return value -- not used in this case
      if (errcode) {
         cerr << "Pthread Join Error: " << strerror(errcode) << endl;
         exit(1);
      }
   }
   return(0);
}
```

```
/*
openMP Example - Hello World - C/C++ Version
FILE: hello_world_OMP.cpp

  first set up environment by typing from the command line the
  following two module load commands
   module load intel
   module load mpt

Compilation on dmc.asc.edu

   to set number of threads through environment variable in bash shell
   (have to do this before you run the program)
   export OMP_NUM_THREADS=8

   to compile the program

      icc -o hello_world_OMP -openmp hello_world_OMP.cpp

   to run -- set number of threads as explained above and then type

     ./hello_world_OMP

*/

using namespace std;
#include <iostream>
#include <omp.h>

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

int nthreads, tid;

   // Fork a team of threads giving them their own copies of variables
   #pragma omp parallel private(nthreads, tid)
   {
      // Obtain thread number
      tid = omp_get_thread_num();

      // print out Hello World Message -- do this as a noninterruptable op
      #pragma omp critical
      {
         cout << "Hello World from thread = " << tid << endl;
      }
      // get number of threads
      nthreads = omp_get_num_threads();

      // print out No of Threads Message -- do this as a noninterruptable op
      #pragma omp critical
      {
         // Only master thread does this
         if (tid == 0) {
            cout << "Number of Threads = "  << nthreads << endl;
         }
      }
   }  // All threads join master thread and disband

}
```