HPC Course - January 2020

Connection to CRAY/SahasraT

All lab exercises can be executed on Cray XC40 (SahasraT)

Connect to SahasraT using ssh or putty

Credentials

```
$ssh <username>@xc40.serc.iisc.ernet.in
username:
password:
```

Cray Programming Environment

```
List all modules loaded to the user environment
$module list
List all modules available on Cray
$module avail
Load a specific module
$module load craype-x86-skylake
Print all environment variables
$printenv
```

Some linux commands

```
Print working directory
$pwd
Listing files and direcotries in working directory
$ls
Listing files in some directory
$ls <some_directory>
Changing directory
$cd <path>
Viewing/editing a file
$vi <filename>
$vi <path>\<filename>
```

Changing permissions: “chmod”

Every file, directory, and link has a set of permissions. These permissions consist of permission groups and permission types. Any time you run ls -l you’ll see a familiar line of -rwx------ or similar combination of the letters r, w, x and - (dashes). These are the permissions for the file or directory. The permission groups are:

1. User - a particular user (account)
2. Group - a particular group of users (may be user-specific group with only one member)
3. Other - other users in the system

The permission types are:

1. Read - For files, this gives permission to read the contents of a file
2. Write - For files, this gives permission to write data to the file. For directories it allows users to add or remove files to a directory.
3. Execute - For files this gives permission to execute a file as through it were a script. For directories, it allows users to open the directory and look at the contents

```
$chmod u+rwx 2020JanOPENMP
$ls -l
```

zip and unzip

```
$unzip myfile.zip
If we would like to make our own zip archive, we use zip:
$zip myfiles.zip myfile1 myfile2 myfile3
```

OPENMP Hands on Exercises

- Hello World
- Data Scoping (private firstprivate..)
- Openmp Schedules
- Compute PI
- Fibonacci series computation
- Matrix Multiplication
- Mandelbrot set computation

1. Copy Sample Code for OPEMP Lab Session

Before starting the course, the example programs and jobscripts used in this tutorial must be copied to your home directory, so that you can work with your personal copy. All examples are present in the “/home/proj/16/secpraba/2020JanOPENMP” directory. Copy folder and change permissions to read write and execute all files in the folder that you created.

a. Create a folder/directory with your name. Try to make it as unique as possible.

```
    $mkdir <yourname>[Number]
```

b. Copy all code for openmp lab sessions into your folder that you just created

```
$scp -r /home/proj/16/secpraba/2020JanOPENMP /<workingdirectory>/<yourname>[Number]
```

Exercise 1 - Running a simple serial Hello World using a jobscript on Cray

1. View the cpp or f95 file

```
$vi Ex1HelloWorld.cpp
OR
$vi Ex1HelloWorld.f95
```

Use an editor like vi to write a program "MyHelloWorld.cpp" [vi MyHelloWorld.cpp]

OpenMPHelloWorld Part 1:

2. First write a serial program that prints "Hello World!"

```
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char *argv[])
{
     printf("Hello World! \n");
     return 0;
}
```

3. Compile the serial program

Compile the program using CC for C++ and cc for c code. Please note that the default cray enviroment defines the actual compilers that will be used with these commands. These could be the gnu compilers or intel compilers depending upon the module loaded in the enviroment.

```
$CC MyHelloWorld.cpp -o MyHelloWorld.out
```

Alternatively one could use the makefile provided in the folder.

```
$make
```

DO NOT RUN THE EXECUTABLE IN INTERACTIVE MODE!!!

4. Program execution jobscript

Execute the serial Program USING A JOBSCRIPT in batch mode. You must first identify and elaborate the number of resources you want on Cray. In our example, the number of resources would be the number of openmp threads that we want to use. As discussed in the lecture, these could be set as an environment variable.

```
$vi jobscript24Threads
```

This is what you should see:

```
#!/bin/sh
#This job should be redirected to idqueue
#PBS -N OMPHelloWorld_24threads
```

```
#PBS -l select=1:ncpus=24
#PBS -l walltime=00:02:00
#PBS -l place=scatter
#PBS -l accelerator_type="None"
#PBS -S /bin/sh@sdb -V
export OMP_NUM_THREADS=24
export OMP_DISPLAY_ENV=TRUE
. /opt/modules/default/init/sh
cd <CURRENT WORKING DIRECTORY THAT HAS THE EXECUTABLE THAT YOU JUST COMPILED>
aprun -j 1 -n 1 -N 1 -d $OMP_NUM_THREADS ./Ex1HelloWorld.out
```

5. Make sure that the "cd" command has the folder name of your current working folder that has the helloworld executable and code.

6. If you have given another name to the output executable in the makefile or compile command, make sure that the "aprun" in teh jobscript points to that executable.

7. Save the jobscript and submit a job using PBS command qsub

```
$module load pbs
$qsub <jobscript>
```

This should return a job id of the form. Please note: each person will get a different number for a unique batch job id assigned to his/her submitted job

```
288654.sdb
```

8. verify of job has completed using the "qstat" command

```
$qstat
$qstat -a
```

```
$qstst -u <username>
$ls -ltr
```

9. Check the output, Discuss

```
$vi OMPHelloWorld_24threads.o<your_job_id>
$vi OMPHelloWorld_24threads.o288654
```

OPENMP HelloWorld Exercise Part A:

Parallelise the program by making the following changes

1. include the openmp header file
2. add a parallel region using #pragma omp parallel outside the printf("Hello World!")
3. identify the number of processors in the machine using omp_get_num_procs() //Runtime function
4. print number of processors
5. print max number of threads (default) using omp_get_max_threads() runtime function By default, most compilers will create 1 thread for each core.

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

int main (int argc, char *argv[])
{
  printf("Number of Processors = %d \n",omp_get_num_procs());
  printf("Max Number of Threads = %d \n",omp_get_max_threads());
  printf("Number of Threads outside parallel region = %d \n",omp_get_num_threads());
  #pragma omp parallel
  {
```

```
        printf("Parallel Region 1 - Hello World!");
    }
}
```

6. Compile the program using make
7. You do not need to change the jobscript if the name of the executable and path is same
8. submit another job using qsub
9. report the output/discuss

OPENMP HelloWorld Exerise Part B:

Use of omp_get_thread_num() inside the parallel region

1. declare int tid = 0; before the parallel region (variable that holds the thread id of each thread in the parallel region)
2. Edit the code to include the "private clause" by changing #pragma omp parallel to #pragma omp parallel private (tid)
3. Inside the parallel region add the following code

```
    tid = omp_get_thread_num();
    printf("Parallel Region 1 - MyHello from thread =%d\n", tid);
```

4. Change the printf or cout statement in the parallel region as follows

```
    printf("Hello World from thread = %d\n", tid);
```

```
The code now looks like this...
  #pragma omp parallel private(tid)
  {
      tid = omp_get_thread_num();
      printf("Parallel Region 1 - MyHello from thread =%d\n", tid);
```

```
}
```

5. Compile and re-run the program using "qsub". Run multiple times to see the changing order of tid.

OPENMP HelloWorld Part C:

Use of environment variables to override the number of threads in code to something other than max threads printed in part B

1. Note the environment variable set in the jobscript. Change this to any value (preferably less than 24 and submit the job and report changes if any in the output)

```
export OMP_NUM_THREADS=24
```

OPENMP HelloWorld Part D:

Set the variable NUM_THREADS to explicitly override the number of threads in the environment variable.

This could be done by change the parallel region as follows:

```
//------------------------------------------------------------------------
//start parallel region 2 with 10 threads. override the number of threads
//that are specified in the environment variable.
//Also illustrates the use of a master thread and a barrier statement
//------------------------------------------------------------------------
/* Fork a team of threads giving them their own copies of variables */
#pragma omp parallel num_threads(10) private(nthreads, tid)
{

        /* Obtain thread number */
```

```
            tid = omp_get_thread_num();
            printf("Hello World from thread = %d\n", tid);

        #pragma omp barrier

            /* Only master thread does this */
            #pragma omp master //if (tid == 0)
            {
                   nthreads = omp_get_num_threads();
                   printf("Number of threads inside parallel region= %d\n", nthreads);
            }

    }  /* All threads join master thread and disband */
```

Submit another job and report the output of the program

END OF HELLOWORLD EXERCISE
