HPC Course - January 2020

OPENMP Hands on Exercises

- Hello World [folder:OMPHelloWorld]
- Openmp Schedules [folder:OMPSchedules]
- #### Data Scoping (private firstprivate..) [folder:OMPDataScoping]
- Compute PI [folder: OMPComputePI]
- Fibonacci series computation [folder:OMPFibonacci]
- Matrix Multiplication [folder:OMPMatrixMult]
- Mandelbrot set computation [folder:OMPmandelbrot]

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 3 - Variable scoping in OpenMP

The entire exercise consisting of the following functions is coded in the file named Ex3DataScoping.cpp or Ex3DataScoping.f95 in folder

View the cpp or f95 file

```
 $cd /home/proj/16/secpraba/2019SepOPENMP/OMPSchedules
$cd <yourworkingdirectory>/OMPSchedules
$ls
$vi Ex3DataScoping.cpp
OR
$vi Ex3DataScoping.f95
```

Variable scoping Part 1:

Example of a shared variable. In a parallel region, any data declared outside it will be shared: any thread using a variable x will access the same memory location associated with that variable.

```
void exampleSharedVar()
{

  int x = 5;
  printf("X is a shared varible for all threads \n");
  printf("X is initialised to %d before the #pragma omp parallel region \n", x);
  #pragma omp parallel
  {
    x = x+1;
    printf("X is updated by thread %d to value %d \n", omp_get_thread_num(), x);

  }
  printf("Value of X at the end of the parallel region is: %d \n", x);

  x = 0;
  printf("X is reset to %d before the #pragma omp parallel for region \n");
  #pragma omp parallel for
  for (int i = 0; i < 10; i++)
  {
    x = x+1;
    printf("X is updated by thread %d to value %d \n", omp_get_thread_num(), x);

  }
  printf("Value of X at the end of the parallel for region is: %d \n", x);

}
```

Compile the program using the "make" command

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 Ex2DataScoping.cpp -o Ex2DataScoping.out
```

Alternatively one could use the makefile provided in the folder.

```
$vi makefile
$make
```

DO NOT RUN THE EXECUTABLE IN INTERACTIVE MODE!!!

Edit the jobscript "jobscriptCray24Threads"

```
$vi jobscriptCray24Threads
cd <CURRENT WORKING DIRECTORY THAT HAS THE EXECUTABLE THAT YOU JUST COMPILED>
aprun -j 1 -n 1 -N 1 -d $OMP_NUM_THREADS ./Ex2DataScoping.out
Make sure that the executable and current working directory are updated to files that yo
```

Submit a job

```
$qsub <jobscriptname>
$qstat -u <username>
```

View the ouput

```
$vi <jobname>.o<jobid>
$vi <jobname>.e<jobid>
```

Understanding the output

Discuss the output

#

Variable Scoping Part 2:

Example of a private variable

```
void examplePrivateVar()
{
  int x = 5;
  printf("X is a Private varible for all threads \n");
  printf("X is not explicitly initialised in the parallel region. Could result in bad da
  printf("X is initialised to %d before the #pragma omp parallel region \n", x);
  #pragma omp parallel private(x)
  {
    x = x+1;
    printf("X is updated by thread %d to value %d \n", omp_get_thread_num(), x);

  }
  printf("Value of X at the end of the parallel region is: %d \n", x);

  x = 0;
  printf("X is reset to %d before the #pragma omp parallel for region \n");
  printf("X is not explicitly initialised in the parallel region. Could result in bad da
  #pragma omp parallel for private(x)
  for (int i = 0; i < 10; i++)
  {
    x = x+1;
    printf("X is updated by thread %d to value %d \n", omp_get_thread_num(), x);

  }
  printf("Value of X at the end of the parallel for region is: %d \n", x);

}
```

Compile the program using the "make" command

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 Ex2Schedules.cpp -o Ex2Schedules.out
```

Alternatively one could use the makefile provided in the folder.

```
$vi makefile
$make
```

DO NOT RUN THE EXECUTABLE IN INTERACTIVE MODE!!!

Edit the jobscript "jobscriptCray24Threads"

```
$vi jobscriptCray24Threads
cd <CURRENT WORKING DIRECTORY THAT HAS THE EXECUTABLE THAT YOU JUST COMPILED>
aprun -j 1 -n 1 -N 1 -d $OMP_NUM_THREADS ./Ex3DataScoping.out
Make sure that the executable and current working directory are updated to files that yo
```

Submit a job

```
$qsub <jobscriptname>
$qstat -u <username>
```

View the ouput

```
$vi <jobname>.o<jobid>
$vi <jobname>.e<jobid>
```

Understanding the output

Discuss the output

#

OMPEMP Variable Scoping Part 3:

First Private private variables are completely separate from any variables by the same name in the surrounding scope. However, there are two cases where you may want some storage association between a private variable and a global counterpart. First of all, private variables are created with an undefined value. You can force their initialization with clause{firstprivate}. In the following function variable t behaves like a private variable, except that it is initialized to the outside value.

```
void firstPrivate()
{
  int t=2;
#pragma omp parallel firstprivate(t)
  {
    t += omp_get_thread_num();
    printf("t is updated by thread %d to value %d \n", omp_get_thread_num(), t);
  }
  printf("Value of t after the parallel region is %d \n", t);
}
```

Compile the program using the "make" command

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 Ex3DataScoping.cpp -o Ex3DataScoping.out
```

Alternatively one could use the makefile provided in the folder.

```
$vi makefile
$make
```

DO NOT RUN THE EXECUTABLE IN INTERACTIVE MODE!!!

Edit the jobscript "jobscriptCray24Threads"

```
$vi jobscriptCray24Threads
cd <CURRENT WORKING DIRECTORY THAT HAS THE EXECUTABLE THAT YOU JUST COMPILED>
aprun -j 1 -n 1 -N 1 -d $OMP_NUM_THREADS ./Ex2Schedules.out
Make sure that the executable and current working directory are updated to files that yc
```

Submit a job

```
$qsub <jobscriptname>
$qstat -u <username>
```

View the ouput

```
$vi <jobname>.o<jobid>
$vi <jobname>.e<jobid>
```

Understanding the output

Discuss the output

#

OMPEMP Variable Scoping Part 4:

Last Private Secondly, you may want a private value to be preserved to the environment outside the parallel region. This really only makes sense in one case, where you preserve a private variable from the last iteration of a parallel loop, or the last section in an sections construct. This is done with clause{lastprivate}: NOTE: lastprivate works only only a omp parallel for OR omp sections Also note that the value is from the LAST iteration of the for loop, NOT from the thread that runs last

```
void lastPrivate()
{

  int t=2;
  int m= 0;
#pragma omp parallel for firstprivate(t) lastprivate(m)
  for (int i = 0; i < 10; i++)
  {
    t += omp_get_thread_num();
    m = i*i;
    printf("t is updated by thread %d to value %d \n", omp_get_thread_num(), t);
  }
  printf("Value of t after the parallel region is %d \n", t);
  printf("Value of m after the parallel region is %d \n", m);


#pragma omp parallel for shared(t) lastprivate(m)
  for (int i = 0; i < 10; i++)
  {
    t = i*i;
    m = i*i;
    printf("t is updated by thread %d to value %d \n", omp_get_thread_num(), t);
    printf("m is updated by thread %d to value %d \n", omp_get_thread_num(), m);
  }
  printf("Value of t after the parallel region is %d \n", t);
  printf("Value of m after the parallel region is %d \n", m);

}
```

Compile the program using the "make" command

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 Ex3DataScoping.cpp -o Ex3DataScoping.out
```

Alternatively one could use the makefile provided in the folder.

```
$vi makefile
$make
```

DO NOT RUN THE EXECUTABLE IN INTERACTIVE MODE!!!

Edit the jobscript "jobscriptCray24Threads"

```
$vi jobscriptCray24Threads
cd <CURRENT WORKING DIRECTORY THAT HAS THE EXECUTABLE THAT YOU JUST COMPILED>
aprun -j 1 -n 1 -N 1 -d $OMP_NUM_THREADS ./Ex2Schedules.out
Make sure that the executable and current working directory are updated to files that yo
```

Submit a job

```
$qsub <jobscriptname>
$qstat -u <username>
```

View the ouput

```
$vi <jobname>.o<jobid>
$vi <jobname>.e<jobid>
```

Understanding the output

Discuss the output