HPC Course - January 2020

OPENMP Hands on Exercises

- Hello World

*** Openmp Schedules**

- Data Scoping (private firstprivate..)
- 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 2 - Understanding how a "schedule" clause partitions the iteration space of a for loop

The entire exercise consisting of the following functions is coded in the file named Ex2Schedules.cpp or Ex2Schedule.f95.

View the cpp or f95 file

```
$cd /home/proj/16/secpraba/2019SepOPENMP/OMPSchedules
$cd <yourworkingdirectory>/OMPSchedules
$ls
$vi Ex2Schedules.cpp
OR
$vi Ex2Schedules.f95
```

OpenMPSchedules Part 1:

One could use the code provided in this file to execute and analyse or write their own code to implement the following: Write a function that runs a for loop in an openmp parallel region, using an arbitrary iteration space(N). It is preferable to have a small N, preferably in 2 digits as it is easier to interpret the logs. The output just runs over the loop and prints the OMP Threadnumber along with the iteration that it processed.

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

void ompForWithoutScheduleClause(int N)
{
```

```
  printf("OMP for without any schedule clause \n");
  #pragma omp parallel
  {
    #pragma omp for
    for (int i=0; i<N; i++) {
       printf("Thread Number %d - loop iteration %d \n", omp_get_thread_num(), i);
       // do something with i
        }
  }
}

int main (int argc, char *argv[])
{
   ompForWithoutScheduleClause(25);
}
```

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 ./Ex2Schedules.out
Make sure that the executable and current working directory are updated to files that you are working on.
```

Submit a job

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

View the ouput

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

Understanding the output

The iteration space (N=24 in my case), is divided by the number of threads (24 in my case) and each iteration is assigned to one thread. Using OMP_NUM_THREADS = 30, one can observe that N/OMP_NUM_THREADS (30/24) , rounded to the next integer, i.e. 2 iterations are assigned to each thread. Thread 0 - iterations (0,1) Thread 1 = iterations (2,3) .. and so on...

```
OMP for without any schedule clause
Thread Number 0 - loop iteration 0
Thread Number 7 - loop iteration 14
Thread Number 0 - loop iteration 1
Thread Number 2 - loop iteration 4
Thread Number 2 - loop iteration 5
Thread Number 12 - loop iteration 24
Thread Number 12 - loop iteration 25
```

```
Thread Number 1 - loop iteration 2
Thread Number 1 - loop iteration 3
Thread Number 3 - loop iteration 6
Thread Number 3 - loop iteration 7
Thread Number 9 - loop iteration 18
Thread Number 9 - loop iteration 19
Thread Number 8 - loop iteration 16
Thread Number 8 - loop iteration 17
Thread Number 11 - loop iteration 22
Thread Number 11 - loop iteration 23
Thread Number 7 - loop iteration 15
Thread Number 14 - loop iteration 28
Thread Number 14 - loop iteration 29
Thread Number 6 - loop iteration 12
Thread Number 6 - loop iteration 13
Thread Number 10 - loop iteration 20
Thread Number 10 - loop iteration 21
Thread Number 13 - loop iteration 26
Thread Number 13 - loop iteration 27
Thread Number 4 - loop iteration 8
Thread Number 4 - loop iteration 9
Thread Number 5 - loop iteration 10
Thread Number 5 - loop iteration 11
```

##################################

OpenMPSchedules Part 2:

Add a new function with a parallel for loop with a static schedule clause and a chunksize. Try the same with a static schedule and no chunksize.

```
#pragma omp parallel for schedule static(chunksize)
```

Each thread will execute a chunk of loop iterations. For example if number of threads is 4 and `total iterations is 25, with schedule of chunksize 5 0,1,2,3,4 will be processed by thread0 5,6,7,8,9 will be processed by thread1 10,11,12,13,14 by thread2 15,16,17,18,19 by thread3 20,21,22,23,24

by thread0

The code looks as follows

```
void ompLoopStaticChunkExample(int N)
{
 printf("schedule static chunksize = 5 \n");
 #pragma omp parallel
 {
   #pragma omp for schedule(static,5)
   for (int i=0; i<N; i++) {
      printf("Thread Number %d - loop iteration %d \n", omp_get_thread_num(), i);
      // do something with i
      }
  }
}
```

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 ./Ex2Schedules.out
Make sure that the executable and current working directory are updated to files that you are working on.
```

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

##################################

OpenMPSchedules Part 3:

Add a new function with a parallel for loop with a guided schedule clause and a chunksize Repeat the same without a chunksize. GUIDED

- If you specify a value for n, the iterations of a loop are divided into chunks
- such that the size of each successive chunk is exponentially decreasing.
- n specifies the size of the smallest chunk, except possibly the last.
- If you do not specify a value for n, the default value is 1.
- The size of the initial chunk is proportional to CEILING(number_of_iterations / number_of_threads) iterations.
- Subsequent chunks are proportional to CEILING(number_of_iterations_remaining / number_of_threads) iterations.
- If n is greater than 1, each chunk must contain at least n consecutive iterations (except for the last chunk to be assigned, which can have fewer than n iterations.)
- As each thread finishes a chunk, it dynamically obtains the next available chunk.
- You can use guided scheduling in a situation in which
- multiple threads in a team might arrive at a DO work-sharing construct at varying times,
- and each iteration requires roughly the same amount of work.
- For example, if you have a DO loop preceded by one or more work-sharing SECTIONS or DO constructs with NOWAIT clauses,
- you can guarantee that no thread waits at the barrier longer than it takes another thread to execute its final iteration, or final k iterations if a chunk size of k is specified.
- The GUIDED schedule requires the fewest synchronizations of all the scheduling methods.

```
#pragma omp parallel for schedule guided(chunksize)
```

The code looks as follows

```
void ompLoopGuidedChunkExample(int N)
{
 printf("schedule guided chunksize = %d \n", 5);
 #pragma omp parallel
 {
   #pragma omp for schedule(guided,5)
   for (int i=0; i<N; i++) {
```

```
        printf("Thread Number %d - loop iteration %d \n", omp_get_thread_num(), i);
        // do something with i
        }
    }
}
```

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 ./Ex2Schedules.out
Make sure that the executable and current working directory are updated to files that you are working on.
```

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 Iterations are dynamically assigned to threads in blocks as threads request them until no blocks remain to be assigned. Similar to DYNAMIC (please write your own function code for this) except that the block size decreases each time a parcel of work is given to a thread. The size of the initial block is proportional to: number_of_iterations / number_of_threads Subsequent blocks are proportional to number_of_iterations_remaining / number_of_threads The chunk parameter defines the minimum block size. The default chunk size is 1. As seen in the output of the following function. Some threads perform a large numer of iterations while others do not. This is a feature of Dynamic scheduling where work is assigned to the free thread. If there are multiple free threads as in our case where we are NOT doing any other work, we could observe an extremely random pattern of iteration space to threads.