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 6 - Matrix Multiplication

Write code that computes the product of 2 matrices A and B (see the program or write soemthing like the code below)

```
 void matmul(long N, double *a, double *b, double *c) {
long i, j, k;

for (i = 0; i < N; i ++)
for (j = 0; j < N; j ++)
for (k = 0; k < N; k ++)
    c[i * N + j] += a[i * N + k] * b[k * N + j];
}
````
Program
```sh
Function Name: MatrixMultSerial(int NRA, int NCA, int NCB)
input:
      Number of Rows of Matrix A (NRA)
      Number of Cols of Matrix A (NCA)
      Number of Cols of Matrix B (NCB)
Process:
        generates the 2 matrices A and B using random numbers
    multiplies the 2 matrices to compute the product
Output:
  Result of matrix multiplication is written to a file MatrixMultiplyOutput.txt
  returns the time taken in seconds to complete Matrix Multiplication
```

1. Compile and run the program and see the time taken for the serial code

2. Using OPENMP Write matrix multiplication code in parallel

(see MatrixMultWithOMPFOR(...)) using #pragma omp for with NO schedule clause a) Which variables are private and which ones need to be shared - elaborate The function should have the same inputs as the serial function with an additional input parameter - int numThreads. This will allow you to time the parallel code for various number of threads. outputs to the parallel function should be similar to the serial code // Result of matrix multiplication is written to a file MatrixMultiplyOutput.txt // returns the time taken in seconds to complete Matrix Multiplication

3. For fixed matrix size, comopute the speedup using varies thread sizes

4. How does speedup change with increasing number of threads?

5. What would you expect to happen if the number of threads was increased to more than the number of cores?

6. Do you think there will be a change in the speed up if Schedule - chunk clause was used?

7. Add a new function which is similar to the parallel function, but accepts as input a chunk size

```
double MatrixMultWithScheduleChunk(int num_threads, int chunk_size, int NRA, int
```

8. add a static schedule with chunk size in the #pragma omp for

9. Evaluate speedup for various chunk sizes.

10. Use Blocking (Cache blocking) where matrices are divided into chunks or blocks and block matrix multiplication is performed.

This should give a better performance than the serial code and should be the optimal serial code. Parallelise Blocked matrix multiplication using openmp. Identify the optimal block size for Cray.

```
 b_t:=b transpose
num=N/BLOCK_SIZE;

for(i=0;i<num;i++){
    for (j=0; j<num; j++){
        for (k=0; k<BLOCK_SIZE; k++){
            for (m=0; m<BLOCK_SIZE; m++) {
                double sum=0.0;
                for (r = 0; r < num; r++){
                    for(p=0; p<BLOCK_SIZE; p++){
                    sum += a[i*BLOCK_SIZE*N + r*BLOCK_SIZE + k*N + p]*
                    b_t[j*BLOCK_SIZE*N + r*BLOCK_SIZE + m*N + p];
                    }
                }
                c[i*BLOCK_SIZE*N + j*BLOCK_SIZE + k*N + m] = sum;
            }
        }
    }
}
```