Dated :
Assessment No. : 1

SCENARIO – I

Write a simple OpenMP program for array addition (sum). It is helpful to understand how threads are created in OpenMP.

To examine the above scenario, we are going to take two one-dimensional arrays, each of size of 5. We will then create 5 threads. Each thread will be responsible for one addition operation.

Note: In OpenMP, pre-defined preprecessor directive #pragma omp parallel is used to create threads. We can mention the number of threads to be created as a parameter to num_threads(). If you are not mentioning num_threads(), then the number of threads to be created is equal to number of cores in processor. Thread id can be obtained by using predefined function omp_get_thread_num().

BRIEF ABOUT YOUR APPROACH:

SOURCE CODE:

EXECUTION:

RESULTS:

Dated :
Assessment No. : 1

SCENARIO – II

Write a simple OpenMP program for linear search. It is helpful to understand how threads can be executed in parallel to sequentially check each element of the list for the target value until a match is found or until all the elements have been searched.

To examine the above scenario, it is noteworthy to mention that with the parallel implementation, each and every element will be checked regardless of a match, though, parallely. This is due to the fact that no thread can directly return after finding the element. So, our parallel implementation will be slower than the serial implementation if the element to be found is present in the range [0, (n/p)-1] where n is the length of the array and p is the number of parallel threads/sub-processes.

Note: In OpenMP, to parallelize the for loop, the openMP directive is: #pragma omp parallel for.

BRIEF ABOUT YOUR APPROACH:

SOURCE CODE:

EXECUTION:

RESULTS:

Dated :
Assessment No. : 1

SCENARIO – III

Write a simple OpenMP program for Matrix Multiplication. It is helpful to understand how threads make use of the cores to enable course-grain parallelism.

Note that different threads will write different parts of the result in the array $\boldsymbol{a}$, so we dont't get any problems during the parallel execution. Note that accesses to matrices $\boldsymbol{b}$ and $\boldsymbol{c}$ are read-only and do not introduce any problems either.

Code Snippet

```
int alg_matmul2D(int m, int n, int p, float** a, float** b, float** c)
{
  int i,j,k;
#pragma omp parallel shared(a,b,c) private(i,j,k)
  {
#pragma omp for  schedule(static)
  for (i=0; i<m; i=i+1){
    for (j=0; j<n; j=j+1){
      a[i][j]=0.;
      for (k=0; k<p; k=k+1){
        a[i][j]=(a[i][j])+((b[i][k])*(c[k][j]));
      }
    }
  }
  }
  return 0;
}
```

BRIEF ABOUT YOUR APPROACH:

SOURCE CODE:

EXECUTION:

RESULTS: