DEPARTMENT OF INFORMATION TECHNOLOGY, NITK, SURATHKAL

Parallel Computing

Lab-2-12th August 2024

Note: Execute the following programs given below. Observe the results of each program, write the analysis of the results and take the screenshot of the results. Upload in the Moodle along with the code and screenshots of the results.

Total marks: 2+2+2+2+2 = 10

Use OpenMP to parallelize the following programs.

1. Implement the parallel version of calculation of Pi (π) using #pragma omp parallel. The formula for calculation of Pi (π) is

$$\int_0^1 \frac{4.0}{(1+x\,2)} dx = \pi$$

Run the parallel code and take the execution time with 1, 2, 4, 8, 16, 32 threads. Record the timing.

2. Develop an OpenMp program for matrix multiplication (C=A*B). Analyze the speedup and efficiency of the parallelized code. Vary the size of your matrices from 250, 500, 750, 1000, and 2000 and measure the runtime with one thread. For each matrix size, change the number of threads from 2,4,8., and plot the speedup versus the number of threads. Compute the efficiency.
3. Develop a multi-threaded program to generate and print ‘n’ Fibonacci Series. One thread has to generate the numbers up to the specified limit and another thread has to print them. Ensure proper synchronization. (Note: If your output seems to be OK, try increasing the number of threads and/or n).
4. Write an OpenMP program to perform vector addition of two one-dimensional arrays of size 5 using 5 threads.

5. Write an OpenMP program to understand and analyze the concepts of *private(), firstprivate()*.
   (i) First execute the program by declaring the variable as *private()*, note down the results and write your observation.
   (ii) Execute the same program with *firstprivate()*, record the results and write your observation.

****

(iii) To examine the above scenario, the functions such as

(iv) omp_get_num_procs(),

(v) To examine the above scenario, the functions such as

(vi) omp_get_num_procs(),

omp_set_num_threads(),

(vii) omp_get_num_threads(), omp_in_parallel(), omp_get_dynamic() and

(viii) omp_get_nested() are listed and the explanation is given below to

(ix) explore the concept practically

6. Write a simple OpenMP program to

demonstrate the parallel loop construct