# High Performance Computing
using OpenMP, OpenMPI and CUDA

R Mukesh

Instructor: **Dr. Noor Mahammad**

IIITDM Kancheepuram

July-November 2018

## Content Overview

## Introduction to High Performance Computing

High Performance Computing refers to solving highly resource-intensive problems quickly and efficiently, mostly through use of **parallel computing** infrastructure at multiple levels.

- Using shared memory multi-core and (or) multi-processor environment (OpenMP)
- Using distributed memory cluster of computers (OpenMPI)
- Using vector processors (CUDA)

## OpenMP

(Open Multi-Processing API)

## Introduction to OpenMP

- OpenMP (Open Multi-Processing) is an API for shared memory multi-threading or multi-processing programming using C/C++ and FORTRAN.
- OpenMP offers a rich set of compiler directives, library runtimes and environment variables that simplify creation and synchronisation of threads.

## Exercise 1

(a) Write a program to perform operations on array elements with and without using OpenMP programming and calculate the parallel fraction f. For example: $A[i] = (i + 1) * 2.0;$

(b) Write a program to add two matrices with and without using OpenMP programming and calculate the parallel fraction f.

Run the programs for 1, 2, 4, 8, 10, 12, 14, 18. 22, 26, 30, 34, 38, 42. 46, 50, 54, 58 and 62 threads. Note down the time taken and draw graph between number of threads and time taken.

## Exercise 2

Write program to perform matrix multiplication on row-major, column-major and block matrices.

(a) Run the program without OpenMP and note down the performance.

(b) Run the program with OpenMP for varying number of threads and note down the performance.

(c) Plot the graph between number of threads and execution time.

(d) Calculate the parallel fraction f.

## Exercise 3

Write a program to perform mean filtering on three random image matrices each of size 384 x 512.

(a) Write the programs in OpenMP with and without using sections.

(b) Run the program for 1, 2, 4, 6, 8, 12, 16, 20, 24, 28, 32 threads and plot the graph between number of threads and execution time.

**Note:** Using sections, computation would be performed on all matrices parallely, whereas otherwise computations on the matrices would be performed sequentially, i.e., matrix after matrix.