TITLE	**Parallel Computing Using CUDA**
PROBLEM STATEMENT / DEFINITION	a) Implement Parallel Reduction using Min, Max, Sum and Average operations. b) Write a CUDA program that, given an N-element vector, find- • The maximum element in the vector • The minimum element in the vector • The arithmetic mean of the vector • The standard deviation of the values in the vector Test for input N and generate a randomized vector V of length N (N should be large). The program should generate output as the two computed maximum values as well as the time taken to find each value.
OBJECTIVE	• Learn parallel decomposition of problem. • Learn parallel computing using CUDA.
S/W PACKAGES AND HARDWARE APPARATUS USED	1. Operating System : 64-bit Open source Linux or its derivative 2. Programming Language: C/C++ 3. NVidea GPU 4. CUDA API
REFERENCES	• Jason sanders, Edward Kandrot, “CUDA by Example”, Addison-Wesley, ISBN-13: 978-0-13-138768-3 • Shane Cook, “CUDA Programming: A Developer's Guide to Parallel Computing with GPUs”, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA 2013 ISBN: 9780124159884
STEPS	Refer to theory, algorithm, test input, test output
INSTRUCTIONS FOR WRITING JOURNAL	1. Date 2. Assignment no. 3. Problem definition 4. Learning objective 5. Learning outcome 6. Related Mathematics 7. Concepts related Theory 8. Test cases 9. Program code with proper documentation. 10. Output of program. 11. Conclusion and applications (the verification and testing of outcomes)

Assignment No. HPC A1

- **Aim:**
  Parallel Computing Using CUDA.
- **Problem Statement / Definition:**

a) Implement Parallel Reduction using Min, Max, Sum and Average operations.
b) Write a CUDA program that, given an N-element vector, find-

- The maximum element in the vector
- The minimum element in the vector
- The arithmetic mean of the vector
- The standard deviation of the values in the vector

Test for input N and generate a randomized vector V of length N (N should be large). The program should generate output as the two computed maximum values as well as the time taken to find each value.

- **Prerequisites**
  C/C++ Programming

- **Learning Objectives**
  - Learn parallel decomposition of problem.
  - Learn parallel computing using CUDA.

- **Learning Outcome:**

Students will be able to decompose problem into sub problems, to learn how to use GPUs, to learn to solve sub problem using threads on GPU cores.

- **Related Mathematics**

Mathematical Model
Let S be the system set:
S = {s; e; X; Y; Fme;DD;NDD; Fc; Sc}
s=start state
e=end state
X=set of inputs
X = {X1}
where
X1 = Elements of Vector
Y= Output Set (Minimum / Maximum / Mean / Sum / Standard Deviation)
Fme is the set of main functions
Fme = {f1,f2,f3}
where
f1 = decomposition function
f2 = function to find Minimum / Maximum / Sum
f3 = function to merge results

f4 = function to Mean / Standard Deviation
DD= Deterministic Data
Vector of elements
NDD=Non-deterministic data
No non deterministic data
Fc =failure case:
No failure case identified for this application

- **Theory**

  Dividing a computation into smaller computations and assigning them to different processors for parallel execution are the two key steps in the design of parallel algorithms.
  The process of dividing a computation into smaller parts, some or all of which may potentially be executed in parallel, is called **decomposition**. Tasks are programmer-defined units of computation into which the main computation is subdivided by means of decomposition. Simultaneous execution of multiple tasks is the key to reducing the time required to solve the entire problem. Tasks can be of arbitrary size, but once defined, they are regarded as indivisible units of computation. The tasks into which a problem is decomposed may not all be of the same size.

A recursive program for finding the minimum in an array of numbers A of length n

Suppose we have an array A with n elements. Decompose this array into subgroups with elements 2..So the total subgroups will be n/2. Find minimum from each subgroup parallely. As a result , we get n/2 elements. Apply this same procedure recursively till we get single element. This element will be the smallest among all the elements of the given array.

Overall recursive procedure to find minimum element is as follows:

```
procedure RECURSIVE_MIN (A, n)
    begin
    if (n = 1) then
    min := A[0];
    else
    lmin := RECURSIVE_MIN (A, n/2);
    rmin := RECURSIVE_MIN (&(A[n/2]), n - n/2);
    if (lmin < rmin) then
    min := lmin;
    else
    min := rmin;
    endelse;

    endelse;

    return min;

end RECURSIVE_MIN
```

Consider an array {4,9,1,7,8,11,2,12}. Divide this array into subgroups as shown in following figure. So we have {4,9}, {1,7}, {8,11}, {2,12}. Find minimum from each subgroup. So, we get {4,1,8,2}. Again divide this into subgroups {4,1}, {2,12}.. Find minimum from each subgroup; we get {1,2}. Find minimum among 1 and 2. That is 1. Hence 1 is the minimum or smallest among all the elements of array.

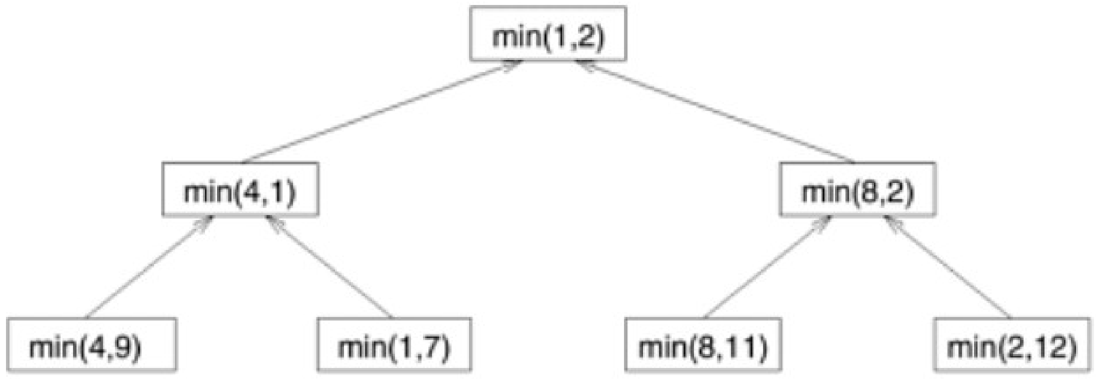

Fig: Finding Minimum by recursive decomposition

Similarly, we can find maximum among elements in an array. We can find sum of all the elements of array with the same procedure i.e. by decomposition and recursion. For average, take sum by recursion and divide it by number of elements. Standard deviation is given by formula

$$\sigma = \sqrt{\frac{\sum (x-\overline{x})^2}{n}}$$

where $\overline{x}$ is mean.

How to run CUDA Program on Remote Machine

1. Open Terminal

2. Get log in to remote system which has GPU and CUDA installed in it.
e.g. ssh student@10.10.15.21

3. Once you get logged in to system, create a cude file with extension .cu and write code in it.

e.g. cat >> sample.cu
Write code here

Press Ctrl+D to come outside the cat command.

4. Compile CUDA program using nvcc command.
e.g. nvcc sample.cu

5. It will create executable file a.out. Rut it.
e.g. ./a.out

When you are compiling using nvcc command, you may get compiler error “nvcc command not found”

In this case, on remote machine, of which you are using GPU,
you have to run following commands:

Open the terminal and type:
gedit ~/.bashrc

This will open .bashrc for editing.

Note: You have check path of CUDA bin folder. Suppose path is /usr/local/cuda-8.0/bin

Add the following to the end of your .bashrc file.
export PATH="$PATH:/usr/local/cuda-8.0/bin"

This sets your PATH variable to the existing PATH plus what you add to the end.
Run following command to reload the configuration.
source ~/.bashrc

- **Test data:**
  Take array with different values of elements.