ASSIGNMENT HPC-1

Roll No: 41205

Problem Statement:

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:

1. To under basics of parallel programming with CUDA
2. Apply parallel programming concepts on vectors

Outcome: One will be able to write parallel programs and compare sequential programs with parallel in terms of performance.

Pre-requisites:
1. 64-bit Linux OS
2. Programming Languages: C/C++

Hardware Specification:
1. x86_64 bit
2. 2/4 GB DDR RAM
3. 80 - 500 GB SATA HD
4. 1GB NIDIA TITAN X Graphics Card

Software Specification:
1. Ubuntu 14.04
2. GPU Driver 352.68
3. CUDA Toolkit 8.0
4. CUDNN Library v5.0

Theory:

Parallel Programming:
There are two fundamental types of parallelism in applications:
- Task parallelism

- Data parallelism

Task parallelism arises when there are many tasks or functions that can be operated independently and largely in parallel. Task parallelism focuses on distributing functions across multiple cores.
Data parallelism arises when there are many data items that can be operated on at the same time. Data parallelism focuses on distributing the data across multiple cores

CUDA:

CUDA programming is especially well-suited to address problems that can be expressed as data-parallel computations. Any applications that process large data sets can use a data- parallel model to speed up the computations. Data-parallel processing maps data elements to parallel threads. The first step in designing a data parallel program is to partition data across threads, with each thread working on a portion of the data.

CUDA Architecture:
A heterogeneous application consists of two parts:

- Host code
- Device code

Host code runs on CPUs and device code runs on GPUs. An application executing on a heterogeneous platform is typically initialized by the CPU. The CPU code is responsible for managing the environment, code, and data for the device before loading compute-intensive tasks on the device. With computationally intensive applications, program sections often exhibit a rich amount of data parallelism. GPUs are used to accelerate the execution of this portion of data parallelism. When a hardware component that is physically separate from the CPU is used to accelerate computationally intensive sections of an application, it is referred to as a hardware accelerator. GPUs are arguably the most common example of a hardware accelerator.

Syntax:

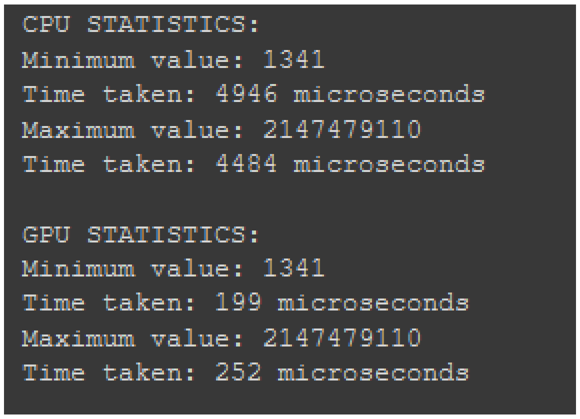
```
__global__ void function(args) {
     int idx = blockIdx.x * blockDim.x + threadIdx.x;
     // perform operations
}

functions<<<blocks, threads>>>(args);
```

Test Cases:

#	Input	Expected Output	Actual Output	Result
1	Compute minimum of randomly generated vector	Value: 1341 GPU faster than CPU	Value: 1341 CPU: 4946 microseconds GPU: 252 microseconds	Success
2	Compute sum of randomly generated vector	Value: 26185517 GPU faster than CPU	Value: 26185517 CPU: 1412 microseconds GPU: 151 microseconds	Success
3	Compute average of randomly generated vector	Value: 499 GPU faster than CPU	Value: 499 CPU: 1412 microseconds GPU: 151 microseconds	Success

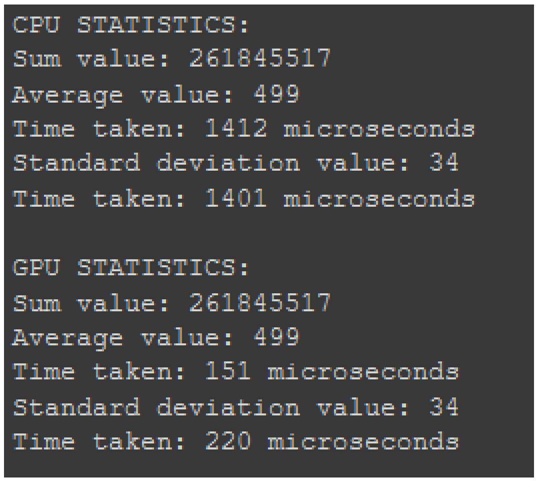
Output:

Minimum, Maximum in a vector



```
CPU STATISTICS:
Minimum value: 1341
Time taken: 4946 microseconds
Maximum value: 2147479110
Time taken: 4484 microseconds

GPU STATISTICS:
Minimum value: 1341
Time taken: 199 microseconds
Maximum value: 2147479110
Time taken: 252 microseconds
```

Sum, Average, Standard Deviation of a vector



```
CPU STATISTICS:
Sum value: 261845517
Average value: 499
Time taken: 1412 microseconds
Standard deviation value: 34
Time taken: 1401 microseconds

GPU STATISTICS:
Sum value: 261845517
Average value: 499
Time taken: 151 microseconds
Standard deviation value: 34
Time taken: 220 microseconds
```

Conclusion: We have implemented parallel reduction using Min, Max, Sum and Average Operations. We have implemented CUDA program for calculating Min, Max, Arithmetic mean and Standard deviation Operations on N-element vector.