# ASSIGNMENT NO 3

## Title of the Assignment:

Implement Min, Max, Sum and Average operations using Parallel Reduction.

## Problem Statement:

Implement Parallel Reduction using Min, Max, Sum and Average operations. 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:

To study and implementation of directive based parallel programming model. To study and implement the operations on vector, generate o/p as two computed max values as well as time taken to find each value.

## Outcome:

Students will understand the implementation of sequential program augmented with compiler directives to specify parallelism. Students will understand the implementation of operations on vector, generate o/p as two computed max with respect to time.

## Prerequisites:

64-bit Open source Linux or its derivative, Programming Languages: C/C++,CUDA Tutorials.

## Software Requirement:

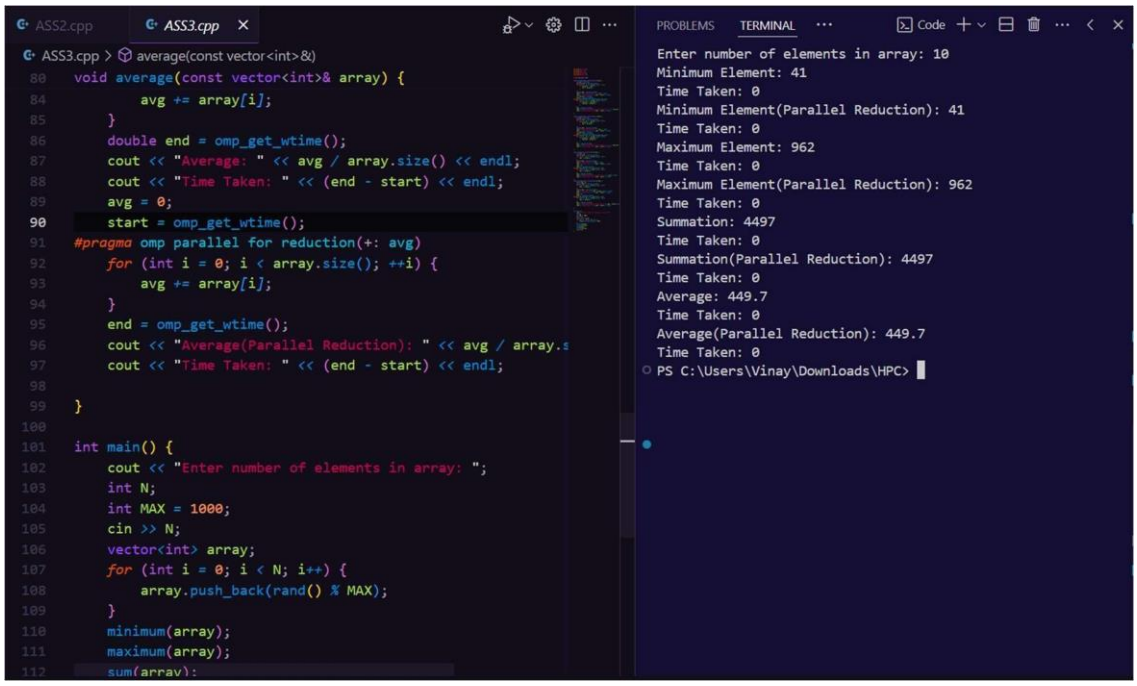
Ubuntu 14.04, GPU Driver 352.68, CUDA Toolkit 8.0, CUDNN Library v5.0

## Source Code:

```
#include<iostream>
#include<omp.h>
#include<bits/stdc++.h>
using namespace std;
void minimum(vector<int> array){
int min = INT_MAX;
double start = omp_get_wtime();
for(auto i = array.begin(); i != array.end();i++){
if(*i < min){
min = *i;
}
}
double end = omp_get_wtime();
cout << "Minimum Element: " << min << endl;
cout << "Time Taken: " << (end-start) << endl;
int min_ele = INT_MAX;
start = omp_get_wtime();
#pragma omp parallel for reduction(min: min_ele)
for(auto i = array.begin(); i != array.end();i++){
if(*i < min_ele){
min_ele = *i;
}
}
end = omp_get_wtime();
cout << "Minimum Element(Parallel Reduction): " << min_ele << endl;
cout << "Time Taken: " << (end-start) << endl;
}
void maximum(vector<int> array){
int max = INT_MIN;
double start = omp_get_wtime();
for(auto i = array.begin(); i != array.end();i++){
if(*i > max){
max = *i;
}
}
double end = omp_get_wtime();
cout << "Maximum Element: " << max << endl;
cout << "Time Taken: " << (end-start) << endl;
int max_ele = INT_MIN;
start = omp_get_wtime();
#pragma omp parallel for reduction(max: max_ele)
for(auto i = array.begin(); i != array.end();i++){
if(*i > max_ele){
max_ele = *i;
}
}
end = omp_get_wtime();
cout << "Maximum Element(Parallel Reduction): " << max_ele << endl;
cout << "Time Taken: " << (end-start) << endl;
}
void sum(vector<int> array){
int sum = 0;
```

```
double start = omp_get_wtime();
for(auto i = array.begin(); i != array.end();i++){
sum += *i;
}
double end = omp_get_wtime();
cout << "Summation: " << sum << endl;
cout << "Time Taken: " << (end-start) << endl;
sum = 0;
start = omp_get_wtime();
#pragma omp parallel for reduction(+: sum)
for(auto i = array.begin(); i != array.end();i++){
sum += *i;
}
end = omp_get_wtime();
cout << "Summation(Parallel Reduction): " << sum << endl;
cout << "Time Taken: " << (end-start) << endl;
}
void average(vector<int> array){
float avg = 0;
double start = omp_get_wtime();
for(auto i = array.begin(); i != array.end();i++){
avg += *i;
}
double end = omp_get_wtime();
cout << "Average: " << avg / array.size() << endl;
cout << "Time Taken: " << (end-start) << endl;
avg = 0;
start = omp_get_wtime();
#pragma omp parallel for reduction(+: avg)
for(auto i = array.begin(); i != array.end();i++){
avg += *i;
}
end = omp_get_wtime();
cout << "Average(Parallel Reduction): " << avg / array.size() << endl;
cout << "Time Taken: " << (end-start) << endl;
}
int main(){
cout << "Enter number of elements in array: ";
int N;
int MAX = 1000;
cin >> N;
vector<int> array;
for(int i = 0; i < N; i++){
array.push_back(rand() % MAX);
}
minimum(array);
maximum(array);
sum(array);
average(array);
return 0;
}
```

## OUTPUT:



```
ASS2.cpp    ASS3.cpp
ASS3.cpp > average(const vector<int>&)
void average(const vector<int>& array) {
        avg += array[i];
    }
    double end = omp_get_wtime();
    cout << "Average: " << avg / array.size() << endl;
    cout << "Time Taken: " << (end - start) << endl;
    avg = 0;
    start = omp_get_wtime();
#pragma omp parallel for reduction(+: avg)
    for (int i = 0; i < array.size(); ++i) {
        avg += array[i];
    }
    end = omp_get_wtime();
    cout << "Average(Parallel Reduction): " << avg / array.s
    cout << "Time Taken: " << (end - start) << endl;

}

int main() {
    cout << "Enter number of elements in array: ";
    int N;
    int MAX = 1000;
    cin >> N;
    vector<int> array;
    for (int i = 0; i < N; i++) {
        array.push_back(rand() % MAX);
    }
    minimum(array);
    maximum(array);
    sum(array);
```

```
PROBLEMS    TERMINAL    Code
Enter number of elements in array: 10
Minimum Element: 41
Time Taken: 0
Minimum Element(Parallel Reduction): 41
Time Taken: 0
Maximum Element: 962
Time Taken: 0
Maximum Element(Parallel Reduction): 962
Time Taken: 0
Summation: 4497
Time Taken: 0
Summation(Parallel Reduction): 4497
Time Taken: 0
Average: 449.7
Time Taken: 0
Average(Parallel Reduction): 449.7
Time Taken: 0
PS C:\Users\Vinay\Downloads\HPC>
```

## Conclusion:

We have successfully implemented Min, Max, Sum and Average operations using Parallel Reduction.