```
!apt-get --purge remove cuda nvidia* libnvidia-*
!dpkg -l | grep cuda- | awk '{print $2}' | xargs -n1 dpkg --purge
!apt-get remove cuda-*
!apt autoremove
!apt-get update
```

```
!wget https://developer.nvidia.com/compute/cuda/9.2/Prod/local_installers/cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64 -O cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64.deb
!dpkg -i cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64.deb
!apt-key add /var/cuda-repo-9-2-local/7fa2af80.pub
!apt-get update
!apt-get install cuda-9.2
```

```
!nvcc --version
```

```
!pip install git+git://github.com/andreinechaev/nvcc4jupyter.git
```

```
%load_ext nvcc_plugin
```

```
%%cu
#include<iostream>
#include<math.h>

#define n 20

using namespace std;

__global__ void minimum(int *input) {
    int tid = threadIdx.x;
    int step_size = 1;
    int number_of_threads = blockDim.x;

    while(number_of_threads>0) {
        if(tid < number_of_threads) {
            int first = tid*step_size*2;
            int second = first + step_size;
            if(input[second] < input[first])
              input[first] = input[second];
        }
        step_size <<= 1; //Doubled
        number_of_threads >>= 1; //Halved
    }
}

__global__ void maximum(int *input) {
    int tid = threadIdx.x;
    int step_size = 1;
    int number_of_threads = blockDim.x;

    while(number_of_threads>0) {
        if(tid < number_of_threads) {
            int first = tid*step_size*2;
            int second = first + step_size;
            if(input[second] > input[first])
              input[first] = input[second];
        }
        step_size <<= 1;
        number_of_threads >>= 1;
    }
}

__global__ void sum(int *input) {
    const int tid = threadIdx.x;
    int step_size = 1;
    int number_of_threads = blockDim.x;

    while(number_of_threads > 0) {
        if(tid < number_of_threads) {
            int first = tid * step_size * 2;
            int second = first + step_size;

            input[first] += input[second];
        }
        step_size <<= 1;
        number_of_threads >>= 1;

    }
}

__global__ void mean_diff_sq(float *input, float mean) {
    input[threadIdx.x] -= mean;
    input[threadIdx.x] *= input[threadIdx.x];
}

__global__ void sum_floats(float *input) {
    int tid = threadIdx.x;
    int step_size = 1;
    int number_of_threads = blockDim.x;

    while(number_of_threads > 0) {
        if(tid < number_of_threads) {
            int first = tid * step_size * 2;
            int second = first + step_size;

            input[first] += input[second];
        }
        step_size <<= 1;
        number_of_threads >>= 1;

    }
```

```
}

void copy_int_to_float(float *dest, int *src, int size){
    for(int i=0; i<size; i++)
        dest[i] = float(src[i]);
}

// Generates random numbers
void random_ints(int *input, int size) {
    for(int i=0; i<size; i++)  {
        input[i] = rand()%100;
        cout<<input[i]<<"  ";
    }
    cout<<endl;
}

int main() {
    int size = n*sizeof(int); //calculate no. of bytes for array

    int *arr;
    int *arr_d, result;

    arr = (int *)malloc(size);
    random_ints(arr, n);

    cudaMalloc((void **)&arr_d, size);

    //Minimum Element
    cudaMemcpy(arr_d, arr, size, cudaMemcpyHostToDevice);

    minimum<<<1,n/2>>>(arr_d);

    cudaMemcpy(&result, arr_d, sizeof(int), cudaMemcpyDeviceToHost);

cout<<"\nThe minimum element is "<<result<<endl;


//Maximum Element
cudaMemcpy(arr_d, arr, size, cudaMemcpyHostToDevice);

maximum<<<1,n/2>>>(arr_d);

cudaMemcpy(&result, arr_d, sizeof(int), cudaMemcpyDeviceToHost);

cout<<"The maximum element is "<<result<<endl;

//Sum of all elements
cudaMemcpy(arr_d, arr, size, cudaMemcpyHostToDevice);

sum<<<1,n/2>>>(arr_d);

cudaMemcpy(&result, arr_d, sizeof(int), cudaMemcpyDeviceToHost);

cout<<"The sum is "<<result<<endl;


//Mean
float mean = float(result)/n;
cout<<"The mean is "<<mean<<endl;

//Variance & Standard deviation
float *arr_float;
float *arr_std, stdValue;

arr_float = (float *)malloc(n*sizeof(float));
cudaMalloc((void **)&arr_std, n*sizeof(float));

copy_int_to_float(arr_float, arr, n);

    cudaMemcpy(arr_std, arr_float, n*sizeof(float), cudaMemcpyHostToDevice);

    mean_diff_sq <<<1,n>>>(arr_std, mean);
    sum_floats<<<1,n/2>>>(arr_std);

    cudaMemcpy(&stdValue, arr_std, sizeof(float), cudaMemcpyDeviceToHost);


    stdValue = stdValue / n;
    cout<<"The variance is "<<stdValue<<endl;
    stdValue = sqrt(stdValue);

    cout<<"The standard deviation is "<<stdValue<<endl;

    cudaFree(arr_d);

    return 0;
}
```

```
83  86  77  15  93  35  86  92  49  21  62  27  90  59  63  26  40  26  72  36

The minimum element is 21928
The maximum element is 21928
The sum is 21928
The mean is 1096.4
The variance is 0
The standard deviation is 0
```

✓ 1s completed at 19:44