```
!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
!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



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


%load_ext nvcc_plugin


%%cu


#include<iostream>
#include<cstdio>
#include<cstdlib>
#include<cuda_runtime.h>
using namespace std;


__global__ void minimum(int *input)
{
  int tid=threadIdx.x;
  auto 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=step_size*2;
      number_of_threads/=2;
  }

}

__global__ void max(int *input)
{
   int tid=threadIdx.x;
```

```
    auto 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*=2;
        number_of_threads/=2;
    }
}

__global__ void sum(int *input)
{
    const int tid=threadIdx.x;
    auto  step_size=1;
    int number_of_threads=blockDim.x;
    while(number_of_threads>0)
    {
        if(tid<number_of_threads)
        {
            const int first=tid*step_size*2;
            const int second=first+step_size;
            input[first]=input[first]+input[second];
        }
    step_size = step_size*2;;
    number_of_threads =number_of_threads/2;
    }
}

__global__ void average(int *input) //You can use above sum() to calculate sum and
{
    const int tid=threadIdx.x;
    auto  step_size=1;
    int number_of_threads=blockDim.x;
    int totalElements=number_of_threads*2;
    while(number_of_threads>0)
    {
        if(tid<number_of_threads)
        {
            const int first=tid*step_size*2;
            const int second=first+step_size;
            input[first]=input[first]+input[second];
        }
        step_size = step_size*2;;
    number_of_threads =number_of_threads/2;
    }
    input[0]=input[0]/totalElements;
}
```

```
int main()
{

  cout<<"Enter the no of elements"<<endl;
  int n;
  n=10;
  srand(n);
  int *arr=new int[n];
  int min=20000;
   //# Generate Input array using rand()
  for(int i=0;i<n;i++)
  {
    arr[i]=rand()%20000;
      if(arr[i]<min)
        min=arr[i];
    cout<<arr[i]<<" ";
  }

  int size=n*sizeof(int); //calculate no. of bytes for array
  int *arr_d,result1;

  //# Allocate memory for min Operation
  cudaMalloc(&arr_d,size);
  cudaMemcpy(arr_d,arr,size,cudaMemcpyHostToDevice);

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

  cudaMemcpy(&result1,arr_d,sizeof(int),cudaMemcpyDeviceToHost);

  cout<<"The minimum element is \n "<<result1<<endl;

  cout<<"The min element (using CPU) is"<<min;


  //#MAX OPERATION
  int *arr_max,maxValue;
  cudaMalloc(&arr_max,size);
  cudaMemcpy(arr_max,arr,size,cudaMemcpyHostToDevice);

  max<<<1,n/2>>>(arr_max);

  cudaMemcpy(&maxValue,arr_max,sizeof(int),cudaMemcpyDeviceToHost);

  cout<<"The maximum element is \n "<<maxValue<<endl;

  //#SUM OPERATION
  int *arr_sum,sumValue;
  cudaMalloc(&arr_sum,size);
  cudaMemcpy(arr_sum,arr,size,cudaMemcpyHostToDevice);

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

  cudaMemcpy(&sumValue,arr_sum,sizeof(int),cudaMemcpyDeviceToHost);

  cout<<"The sum of elements is \n "<<sumValue<<endl;
```

```
  cout<<"The average of elements is \n "<<(sumValue/n)<<endl;

  //# OR-----------

  //#AVG OPERATION
  int *arr_avg,avgValue;
  cudaMalloc(&arr_avg,size);
  cudaMemcpy(arr_avg,arr,size,cudaMemcpyHostToDevice);

  average<<<1,n/2>>>(arr_avg);

  cudaMemcpy(&avgValue,arr_avg,sizeof(int),cudaMemcpyDeviceToHost);

  cout<<"The average of elements is \n "<<avgValue<<endl;


  //# Free all allcated device memeory
   cudaFree(arr_d);
   cudaFree(arr_sum);
   cudaFree(arr_max);
   cudaFree(arr_avg);




return 0;

}
```

```
Enter the no of elements
9295 2008 8678 8725 418 2377 12675 13271 4747 2307 The minimum element is
 418
The min element (using CPU) is418The maximum element is
 13271
The sum of elements is
 57447
The average of elements is
 5744
The average of elements is
 5744
```

```
%%cu

#include <iostream>
#include <stdio.h>
#include <cuda.h>
#include <math.h>
#include <chrono>
#include <bits/stdc++.h>

using namespace std;
using namespace std::chrono;
```

```
__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;
        if(number_of_threads == 1)
          number_of_threads = 0;
        else
          number_of_threads = ceil((double)number_of_threads / 2);
    }
}

__global__ void minimum(int *input, int n) {
    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((first < n && second < n) && input[second] < input[first])
              input[first] = input[second];
        }
        step_size <<= 1;
        if(number_of_threads == 1)
          number_of_threads = 0;
        else
          number_of_threads = ceil((double)number_of_threads / 2);
    }
}

__global__ void gpu_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;
        if(number_of_threads == 1)
          number_of_threads = 0;
        else
```

```
            number_of_threads = ceil((double)number_of_threads / 2);
        }
        if(tid == 0) {
            int first = tid * step_size * 2;
            int second = first + step_size;
            input[first] += input[second];
        }
    }

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

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

    __global__ void gpu_sd(float *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;
            if(number_of_threads == 1)
              number_of_threads = 0;
            else
              number_of_threads = ceil((double)number_of_threads / 2);
        }
        if(tid == 0) {
            int first = tid * step_size * 2;
            int second = first + step_size;
            input[first] += input[second];
        }
    }

    long cpu_sum(int *input, int n) {
        long sum = 0;
        for(int i = 0 ; i < n ; i++) {
            sum += input[i];
        }
        return sum;
    }

    long cpu_min(int *arr, int n) {
        int min =  arr[0];
        for(int i = 1 ; i < n ; i++) {
            if(arr[i] < min)
```

```
            min = arr[i];
        }
        return min;
    }

    long cpu_max(int *arr, int n) {
        int max =  arr[0];
        for(int i = 1 ; i < n ; i++) {
            if(arr[i] > max)
              max = arr[i];
        }
        return max;
    }

    double cpu_sd(int *arr, int n, float mean) {
        float *arr_std = new float[n];
        for(int i = 0 ; i < n ; i++) {
            arr_std[i] = pow(((float)arr[i] - mean),2);
        }
        double total = 0;
        for(int i = 0 ; i < n ; i++) {
            total += arr_std[i];
        }
        total = total / n;
        return sqrt(total);
    }

    void random_init(int *arr, int n) {
        for(int i = 0 ; i < n ; i++) {
            arr[i] = rand()%1000;
        }
    }

    int main() {

        int *d;
        int n = 80;
        int *arr = new int[n];
        int result;
        int size = n * sizeof(int);
        random_init(arr,n);

        cout<<"Input Array: [";
        for(int i = 0 ; i < n ; i++) {
            cout<<arr[i]<<", ";
        }
        cout<<"]"<<endl;

        cout<<"======================================"<<endl;
        cudaMalloc((void **)&d,size);
        cudaMemcpy(d,arr,size,cudaMemcpyHostToDevice);

        float gpu_elapsed_time;
        cudaEvent_t gpu_start,gpu_stop;
        cudaEventCreate(&gpu_start);
```

```
        cudaEventCreate(&gpu_stop);
        cudaEventRecord(gpu_start,0);
        gpu_sum<<<1,n/2>>>(d);

        cudaEventRecord(gpu_stop, 0);
        cudaEventSynchronize(gpu_stop);
        cudaEventElapsedTime(&gpu_elapsed_time, gpu_start, gpu_stop);
        cudaEventDestroy(gpu_start);
        cudaEventDestroy(gpu_stop);

        cudaMemcpy(&result,d,sizeof(int),cudaMemcpyDeviceToHost);
        cout<<"GPU Sum is: "<<result<<"\n";
        float mean = (double)result/n;
        cout<<"GPU Mean is: "<<mean<<endl;

        float *arr_float = new float[n];
        float *arr_std, std;
        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);
        gpu_sd <<<1,n/2>>>(arr_std);

        cudaMemcpy(&std,arr_std,sizeof(float),cudaMemcpyDeviceToHost);
        cout<<"GPU Standard Deviation: "<<sqrt(std/n)<<endl;
        cout<<"======================================="<<endl;

        auto start = high_resolution_clock::now();
        ios_base::sync_with_stdio(false);

        result = cpu_sum(arr,n);
        cout<<"CPU Sum is: "<<result<<"\n";

        auto stop = high_resolution_clock::now();
        double time_taken = chrono::duration_cast<chrono::milliseconds>(stop - start).

        time_taken *= 1e-9;

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

        std = cpu_sd(arr, n, mean);
        cout<<"CPU Standard Deviation: "<<std<<endl;
        cout<<"======================================="<<endl;

        result = 0;
        cudaMemcpy(d,arr,size,cudaMemcpyHostToDevice);
        minimum<<<1,n/2>>>(d,n);
        cudaMemcpy(&result,d,sizeof(int),cudaMemcpyDeviceToHost);
        cout<<"GPU Min is: "<<result<<endl;

        result = cpu_min(arr,n);
        cout<<"CPU Min is: "<<result<<"\n";
        cout<<"======================================="<<endl;
```

```
    cudaMemcpy(d,arr,size,cudaMemcpyHostToDevice);
    maximum<<<1,n/2>>>(d);
    int gMax;
    cudaMemcpy(&result,d,sizeof(int),cudaMemcpyDeviceToHost);
    cout<<"GPU Max is: "<<result<<endl;

    result = cpu_max(arr,n);
    cout<<"CPU Max is: "<<result<<"\n";
    cout<<"======================================="<<endl;

    return 0;
}
```

```
Input Array: [383, 886, 777, 915, 793, 335, 386, 492, 649, 421, 362, 27, 690,
=======================================
GPU Sum is: 38524
GPU Mean is: 481.55
GPU Standard Deviation: 295.582
=======================================
CPU Sum is: 38524
CPU Mean is: 481.55
CPU Standard Deviation: 295.582
=======================================
GPU Min is: 11
CPU Min is: 11
=======================================
GPU Max is: 996
CPU Max is: 996
=======================================
```