```
!nvcc --version
code = """
#include<iostream>
#include<math.h>

#define n 8

using namespace std;

__global__ void minimum(int *input) {
int tid = threadIdx.x;
int step_size = 1;
int number_of_threads = blockDim.x;
printf("No of threads = %d", number_of_threads);
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 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]);
}

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 n=8;
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);
//MIN
cudaMemcpy(arr_d, arr, size, cudaMemcpyHostToDevice);
minimum<<<1,n/2>>>(arr_d);
cudaMemcpy(&result, arr_d, sizeof(int), cudaMemcpyDeviceToHost);
cout<<"The minimum element is "<<result<<endl;
//MAX
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
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;
//AVERAGE
float mean = float(result)/n;
cout<<"The mean is "<<mean<<endl;
//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;
}
"""
text_file = open("ass1.cu", "w")
text_file.write(code)
text_file.close()
!nvcc assign1.cu
!./a.out
!nvprof ./a.out
```

OUTPUT:::

```
83 86 77 15 93 35 86 92 49 21 62 27 90 59 63 26 40 26 72 36 11 68 67 29 82 30 62 23 67 35 29 2
22 58 69 67 93 56 11 42 29 73 21 19 84 37 98 24 15 70 13 26 91 80 56 73 62 70 96 81 5 25 84 27
36 5 46 29 13 57 24 95 82 45 14 67 34 64 43 50 87 8 76 78 88 84 3 51 54 99 32 60 76 68 39 12 26
86 94 39 95 70 34 78 67 1 97 2 17 92 52 56 1 80 86 41 65 89 44 19 40 29 31 17 97 71 81 75 No of
threads = 64No of threads = 64No of threads = 64No of threads = 64No of threads = 64No of
threads = 64No of threads = 64No of threads = 64No of threads = 64No of threads = 64No of
threads = 64No of threads = 64No of threads = 64No of threads = 64No of threads = 64No of
threads = 64No of threads = 64No of threads = 64No of threads = 64No of threads = 64No of
threads = 64No of threads = 64No of threads = 64No of threads = 64No of threads = 64No of
threads = 64No of threads = 64No of threads = 64No of threads = 64No of threads = 64No of
threads = 64No of threads = 64No of threads = 64No of threads = 64No of threads = 64No of
threads = 64No of threads = 64No of threads = 64No of threads = 64No of threads = 64No of
threads = 64No of threads = 64No of threads = 64No of threads = 64No of threads = 64No of
threads = 64No of threads = 64No of threads = 64No of threads = 64No of threads = 64No of
threads = 64No of threads = 64No of threads = 64No of threads = 64No of threads = 64No of
threads = 64No of threads = 64No of threads = 64No of threads = 64No of threads = 64No of
threads = 64No of threads = 64No of threads = 64No of threads = 64
The minimum element is 1
The maximum element is 99
The sum is 6711
The mean is 52.4297
The variance is 818.292
The standard deviation is 28.6058
```

```
83 86 77 15 93 35 86 92 49 21 62 27 90 59 63 26 40 26 72 36 11 68 67 29 82 30 62 23 67 35 29 2
22 58 69 67 93 56 11 42 29 73 21 19 84 37 98 24 15 70 13 26 91 80 56 73 62 70 96 81 5 25 84 27
36 5 46 29 13 57 24 95 82 45 14 67 34 64 43 50 87 8 76 78 88 84 3 51 54 99 32 60 76 68 39 12 26
86 94 39 95 70 34 78 67 1 97 2 17 92 52 56 1 80 86 41 65 89 44 19 40 29 31 17 97 71 81 75
==202== NVPROF is profiling process 202, command: ./a.out No of threads = 64No of threads =
64No of threads = 64No of threads = 64No of threads = 64No of threads = 64No of threads = 64No
of threads = 64No of threads = 64No of threads = 64No of threads = 64No of threads = 64No of
threads = 64No of threads = 64No of threads = 64No of threads = 64No of threads = 64No of
threads = 64No of threads = 64No of threads = 64No of threads = 64No of threads = 64No of
threads = 64No of threads = 64No of threads = 64No of threads = 64No of threads = 64No of
threads = 64No of threads = 64No of threads = 64No of threads = 64No of threads = 64No of
threads = 64No of threads = 64No of threads = 64No of threads = 64No of threads = 64No of
threads = 64No of threads = 64No of threads = 64No of threads = 64No of threads = 64No of
threads = 64No of threads = 64No of threads = 64No of threads = 64No of threads = 64No of
threads = 64No of threads = 64No of threads = 64No of threads = 64No of threads = 64No of
threads = 64No of threads = 64No of threads = 64No of threads = 64No of threads = 64No of
threads = 64No of threads = 64No of threads = 64No of threads = 64No of threads = 64No of
threads = 64No of threads = 64The minimum element is 1 The maximum element is 99 The sum is
6711 The mean is 52.4297 The variance is 818.292 The standard deviation is 28.6058 ==202==
Profiling application: ./a.out ==202== Profiling result: Type Time(%) Time Calls Avg Min Max
Name GPU activities: 88.38% 322.21us 1 322.21us 322.21us 322.21us minimum(int*) 2.67% 9.7270us
4 2.4310us 2.2720us 2.6240us [CUDA memcpy DtoH] 2.16% 7.8720us 1 7.8720us 7.8720us 7.8720us
sum_floats(float*) 2.03% 7.3920us 1 7.3920us 7.3920us 7.3920us maximum(int*) 2.01% 7.3280us 4
1.8320us 1.6320us 2.2720us [CUDA memcpy HtoD] 1.98% 7.2320us 1 7.2320us 7.2320us 7.2320us
sum(int*) 0.77% 2.8160us 1 2.8160us 2.8160us 2.8160us mean_diff_sq(float*, float) API calls:
98.45% 171.28ms 2 85.642ms 15.104us 171.27ms cudaMalloc 0.61% 1.0695ms 5 213.90us 14.475us
988.50us cudaLaunchKernel 0.35% 614.91us 8 76.864us 16.880us 427.01us cudaMemcpy 0.30% 522.30us
1 522.30us 522.30us 522.30us cuDeviceTotalMem 0.25% 434.70us 97 4.4810us 156ns 171.89us
cuDeviceGetAttribute 0.01% 24.809us 1 24.809us 24.809us 24.809us cuDeviceGetName 0.01% 19.245us
1 19.245us 19.245us 19.245us cudaFree 0.00% 3.4640us 1 3.4640us 3.4640us 3.4640us
cuDeviceGetPCIBusId 0.00% 2.2390us 3 746ns 185ns 1.3080us cuDeviceGetCount 0.00% 1.8640us 2
932ns 287ns 1.5770us cuDeviceGet 0.00% 358ns 1 358ns 358ns 358ns cuDeviceGetUuid
```