

Name: Manasi. B. Kshirsagar

PRN: ST17111009

Roll No.: 06

Class: BE Comp SS

Code (Maximum):

```
%%cu
```

```
#include <cstdio>
```

```
#include <iostream>
```

```
using namespace std;
```

```
__global__ void maxi(int* a, int* b, int n)
```

```
{
```

```
    int block = 256 * blockIdx.x;
```

```
    int max = 0;
```

```
    for (int i = block; i < min(256 + block, n); i++)
```

```
    {
```

```
        if (max < a[i])
```

```
        {
```

```
            max = a[i];
```

```
        }
```

```
    }
```

```
    b[blockIdx.x] = max;
```

```
}
```

```

int main()
{
    int n;
    n = 200;
    int a[n];
    for (int i = 0; i < n; i++)
    {
        a[i] = rand() % n;
        cout << a[i] << " ";
    }
    cout<<"\n";
    cudaEvent_t start, end;
    int *ad, *bd;
    int size = n * sizeof(int);
    cudaMalloc(&ad, size);
    cudaMemcpy(ad, a, size, cudaMemcpyHostToDevice);
    int grids = ceil(n * 1.0f / 256.0f);
    cudaMalloc(&bd, grids * sizeof(int));

    dim3 grid(grids, 1);
    dim3 block(1, 1);

    cudaEventCreate(&start);
    cudaEventCreate(&end);
    cudaEventRecord(start);

```

```

while (n > 1)
{
    maxi<<<grids, block>>>(ad, bd, n);
    n = ceil(n * 1.0f / 256.0f);
    cudaMemcpy(ad, bd, n * sizeof(int),
cudaMemcpyDeviceToDevice);
}
    cudaEventRecord(end);
    cudaEventSynchronize(end);
    float time = 0;
    cudaEventElapsedTime(&time, start, end);
    int ans[2];
    cudaMemcpy(ans, ad, 4, cudaMemcpyDeviceToHost);
    cout<<"The maximum element is : " << ans[0] << endl;
    cout<<"The time required : ";
    cout<<time<<endl;
    return 0;
}

```

Output:



```
42 cudaEventCreate(&start);
43 cudaEventCreate(&end);
44 cudaEventRecord(start);
45
46 while (n > 1)
47 {
48     mini<>grids, block>>(ad, bd, n);
49     n = ceil(n * 1.0f / 256.0f);
50     cudaMemcpy(ad, bd, n * sizeof(int),
51               cudaMemcpyDeviceToDevice);
52 }
53 cudaEventRecord(end);
54 cudaEventSynchronize(end);
55 float time = 0;
56 cudaEventElapsedTime(&time, start, end);
57 int ans[2];
58 cudaMemcpy(ans, ad, 4, cudaMemcpyDeviceToHost);
59 cout<<"The maximum element is : " << ans[0] << endl;
60 cout<<"The time required : ";
61 cout<<time<<endl;
62 return 0;
63 }
```

183 86 177 115 193 135 186 92 40 21 162 27 90 59 163 126 140 26 172 136 11 168 167 29 182 130 62 123 67 135 120 2 22 58 69 167 193 56 11 42 29 173 21 119 184 137 198 124 115 170 13 124

The maximum element is : 109
The time required : 0.02864

Code (Minimum):

```
%%cu
```

```
#include <cstdio>
```

```
#include <iostream>
```

```
using namespace std;
```

```
__global__ void mini(int* a, int* b, int n)
```

```
{
```

```
    int block = 256 * blockIdx.x;
```

```
    int minimum = 0;
```

```
    for (int i = block; i < min(256 + block, n); i++)
```

```
    {
```

```
        if (minimum > a[i])
```

```
        {
```

```
            minimum = a[i];
```

```

    }
}
b[blockIdx.x] = minimum;
}

```

```

int main()
{
    int n;
    n = 200;
    int a[n];
    for (int i = 0; i < n; i++)
    {
        a[i] = rand() % n;
        cout << a[i] << " ";
    }
    cout<<"\n";
    cudaEvent_t start, end;
    int *ad, *bd;
    int size = n * sizeof(int);
    cudaMalloc(&ad, size);
    cudaMemcpy(ad, a, size, cudaMemcpyHostToDevice);
    int grids = ceil(n * 1.0f / 256.0f);
    cudaMalloc(&bd, grids * sizeof(int));

    dim3 grid(grids, 1);
    dim3 block(1, 1);

```

```

cudaEventCreate(&start);
cudaEventCreate(&end);
cudaEventRecord(start);

while (n > 1)
{
    mini<<<grids, block>>>(ad, bd, n);
    n = ceil(n * 1.0f / 256.0f);
    cudaMemcpy(ad, bd, n * sizeof(int),
cudaMemcpyDeviceToDevice);
}
cudaEventRecord(end);
cudaEventSynchronize(end);
float time = 0;
cudaEventElapsedTime(&time, start, end);
int ans[2];
cudaMemcpy(ans, ad, 4, cudaMemcpyDeviceToHost);
cout<<"The minimum element is : " << ans[0] << endl;
cout<<"The time required : ";
cout<<time<<endl;
return 0;
}

```

Output:



The screenshot shows a C++ code editor with a dark theme. The code is a CUDA kernel for finding the minimum element in an array. The output window at the bottom displays the results of the execution.

```
47 {  
48     min(<<grids, blocks>>)(ad, bd, n);  
49     n = ceil(n * 1.0f / 256.0f);  
50     cudaMemcpy(ad, bd, n * sizeof(int),  
51         cudaMemcpyDeviceToDevice);  
52 }  
53 cudaEventRecord(end);  
54 cudaEventSynchronize(end);  
55 float time = 0;  
56 cudaEventElapsedTime(&time, start, end);  
57 int ans[2];  
58 cudaMemcpy(ans, ad, 4, cudaMemcpyDeviceToHost);  
59 cout<<"The minimum element is : " << ans[0] << endl;  
60 cout<<"The time required : ";  
61 cout<<time<<endl;  
62 return 0;  
63 }
```

183 86 177 115 193 135 186 92 49 21 162 27 90 59 163 126 140 26 172 136 11 168 167 29 182 130 62 123 67 135 129 2 22 58 69 167 193 56 11 42 29 173 21 119 184 137 198 124 115 170 13 126
The minimum element is : 0
The time required : 0.03872

Code (Std Dev & Variance):

```
%%cu
```

```
#include<iostream>
```

```
#include<cstdio>
```

```
using namespace std;
```

```
__global__ void var(int *a,int *b,int n,float mean)
```

```
{
```

```
    int block=256*blockIdx.x;
```

```
    float sum=0;
```

```
    for(int i=block;i<min(block+256,n);i++)
```

```
    {
```

```
        sum=sum+(a[i]-mean)*(a[i]-mean);
```

```
    }
```

```
    b[blockIdx.x]=sum;
```

```

}
__global__ void sum(int *a,int *b,int n)
{
    int block=256*blockIdx.x;
    int sum=0;
    for(int i=block;i<min(block+256,n);i++)
    {
        sum=sum+a[i];
    }
    b[blockIdx.x]=sum;
}
int main()
{
    int n;
    n=200;
    int a[n];
    cout<<"Elements: ";
    for(int i=0;i<n;i++)
    {
        a[i]=rand()%n;
        cout<<a[i]<<" ";
    }
    int *ad,*bd;
    int size=n*sizeof(int);
    cudaMalloc(&ad,size);
    cudaMemcpy(ad,a,size,cudaMemcpyHostToDevice);

```



```

int grids=ceil(n*1.0f/256.0f);
cudaMalloc(&bd,grids*sizeof(int));
dim3 grid(grids,1);
dim3 block(1,1);
int p=n;
cudaEvent_t start,end;
cudaEventCreate(&start);
cudaEventCreate(&end);
cudaEventRecord(start);
while(n>1)
{
    sum<<<grid,block>>>(ad,bd,n);
    n=ceil(n*1.0f/256.0f);

    cudaMemcpy(ad,bd,n*sizeof(int),cudaMemcpyDeviceToDevice);
}
cudaEventRecord(end);
cudaEventSynchronize(end);
float time=0;
cudaEventElapsedTime(&time,start,end);
cout<<endl<<"The Time is "<<time<<endl;
int add[2];
n=p;
cudaMemcpy(add,ad,4,cudaMemcpyDeviceToHost);
float mean=0.0f;
mean=add[0]/(n*1.0f);

```

```

cout<<"The Mean is "<<mean<<endl;
cudaMalloc(&ad,size);
cudaMemcpy(ad,a,size,cudaMemcpyHostToDevice);
cudaMalloc(&bd,grids*sizeof(int));
var<<<grid,block>>>(ad,bd,n,mean);
n=ceil(n*1.0f/256.0f);
sum<<<grid,block>>>(bd,ad,n);
cudaMemcpy(add,ad,4,cudaMemcpyDeviceToHost);
float sd=sqrt(add[0]/p*1.0f);
cout<<"The Standard Deviation is "<<sd<<endl;
}

```

Output:



The screenshot shows a code editor with a C++ program that uses CUDA for parallel computation. The code calculates the mean and standard deviation of an array of 183 elements. The output in the terminal window is as follows:

```

Elements: 183 86 177 115 193 135 186 92 49 21 162 27 90 59 163 126 140 26 172 136 11 108 167 29 182 190 62 123 67 135 129 2 22 58 69 167 193 56 11 42 29 173 21 119 184 137 198 124 115
The time is 0.000032
The Mean is 102.31
The Standard Deviation is 57.3236

```