گام اول:

```
 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "GeForce 920MX"
  CUDA Driver Version / Runtime Version          11.0 / 10.2
  CUDA Capability Major/Minor version number:    5.0
  Total amount of global memory:                 2048 MBytes (2147483648 bytes)
  ( 2) Multiprocessors, (128) CUDA Cores/MP:     256 CUDA Cores
  GPU Max Clock rate:                            993 MHz (0.99 GHz)
  Memory Clock rate:                             900 Mhz
  Memory Bus Width:                              64-bit
  L2 Cache Size:                                 1048576 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 4 copy engine(s)
  Run time limit on kernels:                     Yes
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  CUDA Device Driver Mode (TCC or WDDM):         WDDM (Windows Display Driver Model)
  Device supports Unified Addressing (UVA):      Yes
  Device supports Compute Preemption:            No
  Supports Cooperative Kernel Launch:            No
  Supports MultiDevice Co-op Kernel Launch:      No
  Device PCI Domain ID / Bus ID / location ID:   0 / 1 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 11.0, CUDA Runtime Version = 10.2, NumDevs = 1
Result = PASS
```

گام دوم:

```
cudaStatus = cudaDeviceSynchronize();
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "cudaDeviceSynchronize returned error code %d after launching addKernel!\n", cudaStatus);
    goto Error;
}

// Copy output vector from GPU buffer to host memory.
cudaStatus = cudaMemcpy(c, dev_c, matSizeX * matSizeY * sizeof(int), cudaMemcpyDeviceToHost);
if (cudaStatus != cudaSuccess) {
    fprintf(stderr, "cudaMemcpy failed!");
    goto Error;
}
cudaEventRecord(stop, NULL);
cudaStatus = cudaEventSynchronize(stop);
float mSecTotal;
cudaStatus = cudaEventElapsedTime(&mSecTotal, start, stop);
printf("Time: %f \n\n", mSecTotal);

or:
cudaFree(dev_c);
cudaFree(dev_a);
cudaFree(dev_b);

return cudaStatus;
```

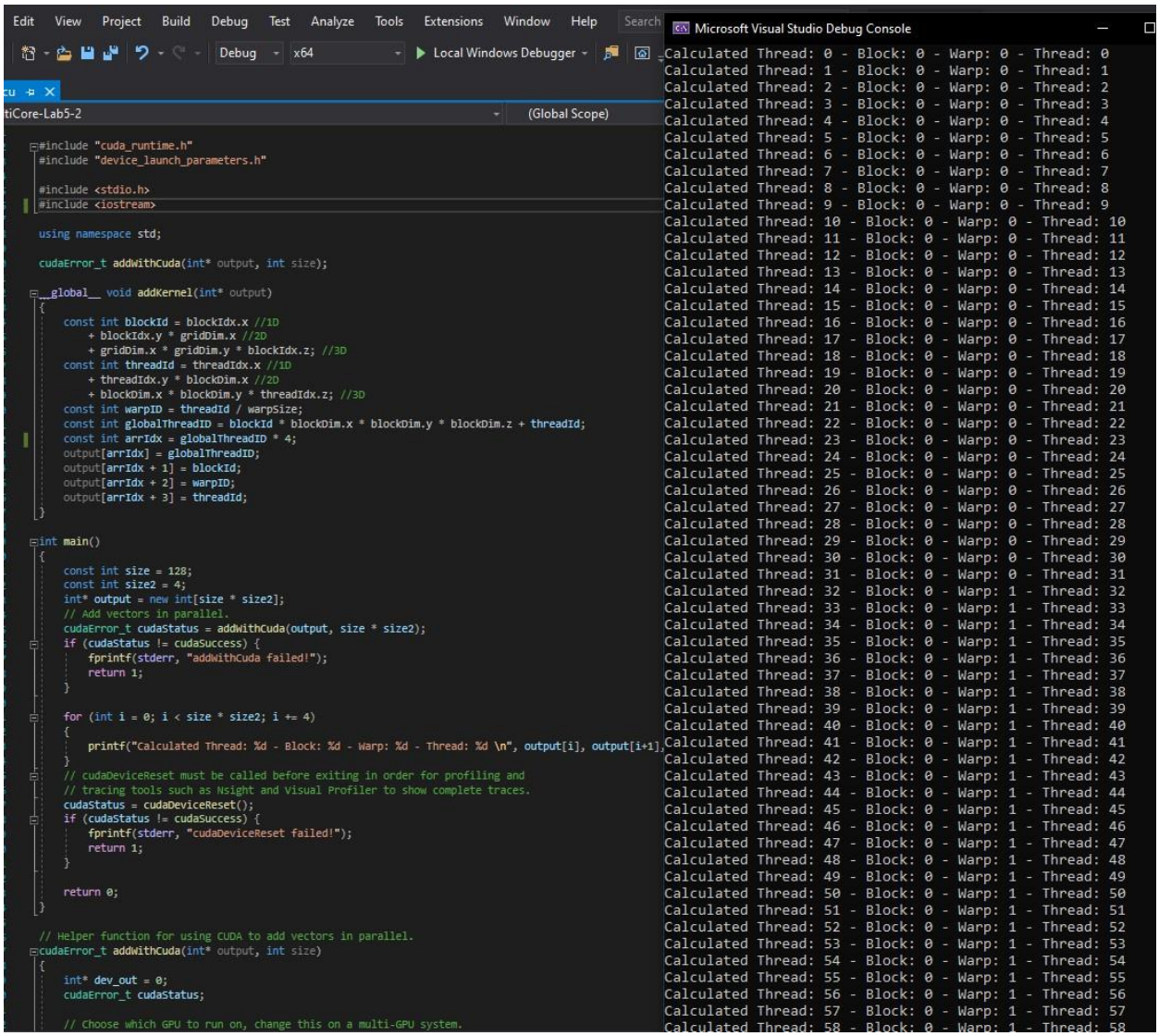
```
Microsoft Visual Studio Debug Console
Time: 0.385248

[-] Vector elements:
0       1       2       3
4       5       6       7
8       9       10      11
12      13      14      15

[-] Vector elements:
16      17      18      19
20      21      22      23
24      25      26      27
28      29      30      31

[-] Vector elements:
16      18      20      22
24      26      28      30
32      34      36      38
40      42      44      46
```

گام سوم:

```
udaError_t addWithCuda_1(int* c, const int* a, const int* b, unsigned int matSizeX, unsigned int matSizeY, unsigned int N, int baseSize);
udaError_t addWithCuda_2(int* c, const int* a, const int* b, unsigned int matSizeX, unsigned int matSizeY, unsigned int N, int baseSize);
oid fillMat(int* v, int matSizeX, int matSizeY);
oid printMat(int* v, int matSizeX, int matSizeY);


__global__ void addKernel_nSums(int* c, const int* a, const int* b, unsigned int N)
{

    int threadID = threadIdx.x + (threadIdx.y * blockDim.x);
    threadID *= N;

    for (int x = threadID; x < N + threadID; x++)
    {
        c[x] = a[x] + b[x];
    }
}


__global__ void addKernel_nBlocks(int* c, const int* a, const int* b)
{

    const int blockId = blockIdx.x //1D
        + blockIdx.y * gridDim.x //2D
        + gridDim.x * gridDim.y * blockIdx.z; //3D
    const int threadId = threadIdx.x //1D
        + threadIdx.y * blockDim.x //2D
        + blockDim.x * blockDim.y * threadIdx.z; //3D
    const int globalThreadID = blockId * blockDim.x * blockDim.y * blockDim.z + threadId;

    c[globalThreadID] = a[globalThreadID] + b[globalThreadID];
}


int main()
{
    const int squareOfN = 10;
    const int baseSize = 32;
    const int matSizeX = baseSize * squareOfN;
    const int matSizeY = baseSize * squareOfN;
    int* a;
    int* b;
    int* c;
    int* d;
    a = (int*)malloc(sizeof(int) * matSizeX * matSizeY);
    b = (int*)malloc(sizeof(int) * matSizeX * matSizeY);
    c = (int*)malloc(sizeof(int) * matSizeX * matSizeY);
    d = (int*)malloc(sizeof(int) * matSizeX * matSizeY);

    fillMat(a, matSizeX, matSizeY);
```

```
Microsoft Visual Studio Debug Console
N Blocks:
Time: 2.050848

N Sums:
Time: 1.286400


C:\Users\TheRe\source\repos\MultiCore-Lab5-1\x64
\Debug\MultiCore-Lab5-1.exe (process 11716) exit
ed with code 0.
To automatically close the console when debuggin
g stops, enable Tools->Options->Debugging->Autom
atically close the console when debugging stops.
Press any key to close this window . . .
```

گام چهارم:


Edit View Project Build Debug Test Analyze Tools Extensions Window Help Search
Debug x64 Local Windows Debugger
tiCore-Lab5-2 (Global Scope)
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include <stdio.h>
#include <iostream>
using namespace std;
cudaError_t addWithCuda(int* output, int size);
__global__ void addKernel(int* output)
{
const int blockId = blockIdx.x //1D
+ blockIdx.y * gridDim.x //2D
+ gridDim.x * gridDim.y * blockIdx.z; //3D
const int threadId = threadIdx.x //1D
+ threadIdx.y * blockDim.x //2D
+ blockDim.x * blockDim.y * threadIdx.z; //3D
const int warpID = threadId / warpSize;
const int globalThreadID = blockId * blockDim.x * blockDim.y * blockDim.z + threadId;
const int arrIdx = globalThreadID * 4;
output[arrIdx] = globalThreadID;
output[arrIdx + 1] = blockId;
output[arrIdx + 2] = warpID;
output[arrIdx + 3] = threadId;
}
int main()
{
const int size = 128;
const int size2 = 4;
int* output = new int[size * size2];
// Add vectors in parallel.
cudaError_t cudaStatus = addWithCuda(output, size * size2);
if (cudaStatus != cudaSuccess) {
fprintf(stderr, "addWithCuda failed!");
return 1;
}
for (int i = 0; i < size * size2; i += 4)
{
printf("Calculated Thread: %d - Block: %d - Warp: %d - Thread: %d \n", output[i], output[i+1],
}
// cudaDeviceReset must be called before exiting in order for profiling and
// tracing tools such as Nsight and Visual Profiler to show complete traces.
cudaStatus = cudaDeviceReset();
if (cudaStatus != cudaSuccess) {
fprintf(stderr, "cudaDeviceReset failed!");
return 1;
}
return 0;
}
// Helper function for using CUDA to add vectors in parallel.
cudaError_t addWithCuda(int* output, int size)
{
int* dev_out = 0;
cudaError_t cudaStatus;
// Choose which GPU to run on, change this on a multi-GPU system.
Microsoft Visual Studio Debug Console
Calculated Thread: 0 - Block: 0 - Warp: 0 - Thread: 0
Calculated Thread: 1 - Block: 0 - Warp: 0 - Thread: 1
Calculated Thread: 2 - Block: 0 - Warp: 0 - Thread: 2
Calculated Thread: 3 - Block: 0 - Warp: 0 - Thread: 3
Calculated Thread: 4 - Block: 0 - Warp: 0 - Thread: 4
Calculated Thread: 5 - Block: 0 - Warp: 0 - Thread: 5
Calculated Thread: 6 - Block: 0 - Warp: 0 - Thread: 6
Calculated Thread: 7 - Block: 0 - Warp: 0 - Thread: 7
Calculated Thread: 8 - Block: 0 - Warp: 0 - Thread: 8
Calculated Thread: 9 - Block: 0 - Warp: 0 - Thread: 9
Calculated Thread: 10 - Block: 0 - Warp: 0 - Thread: 10
Calculated Thread: 11 - Block: 0 - Warp: 0 - Thread: 11
Calculated Thread: 12 - Block: 0 - Warp: 0 - Thread: 12
Calculated Thread: 13 - Block: 0 - Warp: 0 - Thread: 13
Calculated Thread: 14 - Block: 0 - Warp: 0 - Thread: 14
Calculated Thread: 15 - Block: 0 - Warp: 0 - Thread: 15
Calculated Thread: 16 - Block: 0 - Warp: 0 - Thread: 16
Calculated Thread: 17 - Block: 0 - Warp: 0 - Thread: 17
Calculated Thread: 18 - Block: 0 - Warp: 0 - Thread: 18
Calculated Thread: 19 - Block: 0 - Warp: 0 - Thread: 19
Calculated Thread: 20 - Block: 0 - Warp: 0 - Thread: 20
Calculated Thread: 21 - Block: 0 - Warp: 0 - Thread: 21
Calculated Thread: 22 - Block: 0 - Warp: 0 - Thread: 22
Calculated Thread: 23 - Block: 0 - Warp: 0 - Thread: 23
Calculated Thread: 24 - Block: 0 - Warp: 0 - Thread: 24
Calculated Thread: 25 - Block: 0 - Warp: 0 - Thread: 25
Calculated Thread: 26 - Block: 0 - Warp: 0 - Thread: 26
Calculated Thread: 27 - Block: 0 - Warp: 0 - Thread: 27
Calculated Thread: 28 - Block: 0 - Warp: 0 - Thread: 28
Calculated Thread: 29 - Block: 0 - Warp: 0 - Thread: 29
Calculated Thread: 30 - Block: 0 - Warp: 0 - Thread: 30
Calculated Thread: 31 - Block: 0 - Warp: 0 - Thread: 31
Calculated Thread: 32 - Block: 0 - Warp: 1 - Thread: 32
Calculated Thread: 33 - Block: 0 - Warp: 1 - Thread: 33
Calculated Thread: 34 - Block: 0 - Warp: 1 - Thread: 34
Calculated Thread: 35 - Block: 0 - Warp: 1 - Thread: 35
Calculated Thread: 36 - Block: 0 - Warp: 1 - Thread: 36
Calculated Thread: 37 - Block: 0 - Warp: 1 - Thread: 37
Calculated Thread: 38 - Block: 0 - Warp: 1 - Thread: 38
Calculated Thread: 39 - Block: 0 - Warp: 1 - Thread: 39
Calculated Thread: 40 - Block: 0 - Warp: 1 - Thread: 40
Calculated Thread: 41 - Block: 0 - Warp: 1 - Thread: 41
Calculated Thread: 42 - Block: 0 - Warp: 1 - Thread: 42
Calculated Thread: 43 - Block: 0 - Warp: 1 - Thread: 43
Calculated Thread: 44 - Block: 0 - Warp: 1 - Thread: 44
Calculated Thread: 45 - Block: 0 - Warp: 1 - Thread: 45
Calculated Thread: 46 - Block: 0 - Warp: 1 - Thread: 46
Calculated Thread: 47 - Block: 0 - Warp: 1 - Thread: 47
Calculated Thread: 48 - Block: 0 - Warp: 1 - Thread: 48
Calculated Thread: 49 - Block: 0 - Warp: 1 - Thread: 49
Calculated Thread: 50 - Block: 0 - Warp: 1 - Thread: 50
Calculated Thread: 51 - Block: 0 - Warp: 1 - Thread: 51
Calculated Thread: 52 - Block: 0 - Warp: 1 - Thread: 52
Calculated Thread: 53 - Block: 0 - Warp: 1 - Thread: 53
Calculated Thread: 54 - Block: 0 - Warp: 1 - Thread: 54
Calculated Thread: 55 - Block: 0 - Warp: 1 - Thread: 55
Calculated Thread: 56 - Block: 0 - Warp: 1 - Thread: 56
Calculated Thread: 57 - Block: 0 - Warp: 1 - Thread: 57
Calculated Thread: 58 - Block: 0 - Warp: 1 - Thread: 58