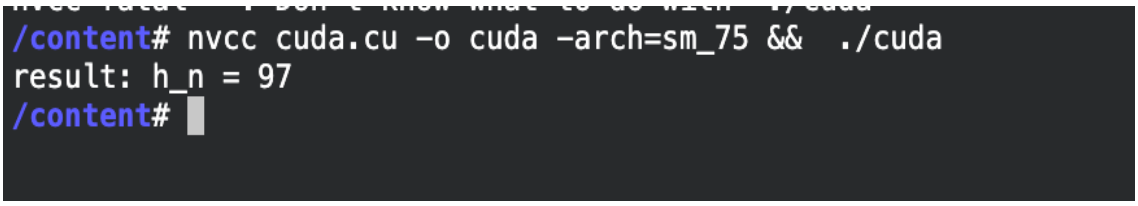
PRATIK DHIMAL
2407779

1. This program is very basic. It just explains the flow of execution of CUDA code. First we create device variables and allocate memory for the device variable using **cudaMalloc().** Then we use **cudaMemcpy()** to copy the value 19 to the device memory (**GPU MEMORY**) .Then we launch a kernel with <<<1,1>>> configuration which just creates one one block inside which is 1 thread. In the kernel function we have a simple v**alue change to 97**. Than we copy the value back to host memory so that we can display it and the final **result in the host in 97.** Lastly we free all the memory.



2. This program is a simple visualization program to visualize the threads and see its respective id where it is present and its parent block as well as grid. For instance in case 4 we are using kernel <<<2,dim3(2,3,4)>>> so here we will have 2 blocks in x direction. And inside each block we will have 2 threads in x, 3 in y, and 4 in z. So the kernel function just displays the respective ids.

```
Running with kernel <<<2, dim>>>()
blockIdx.x=0     blockIdx.y=0     blockIdx.z=0     threadIdx.x=0     threadIdx.y=0     threadIdx.z=0
blockIdx.x=0     blockIdx.y=0     blockIdx.z=0     threadIdx.x=1     threadIdx.y=0     threadIdx.z=0
blockIdx.x=0     blockIdx.y=0     blockIdx.z=0     threadIdx.x=0     threadIdx.y=1     threadIdx.z=0
blockIdx.x=0     blockIdx.y=0     blockIdx.z=0     threadIdx.x=1     threadIdx.y=1     threadIdx.z=0
blockIdx.x=0     blockIdx.y=0     blockIdx.z=0     threadIdx.x=0     threadIdx.y=2     threadIdx.z=0
blockIdx.x=0     blockIdx.y=0     blockIdx.z=0     threadIdx.x=1     threadIdx.y=2     threadIdx.z=0
blockIdx.x=0     blockIdx.y=0     blockIdx.z=0     threadIdx.x=0     threadIdx.y=0     threadIdx.z=1
blockIdx.x=0     blockIdx.y=0     blockIdx.z=0     threadIdx.x=1     threadIdx.y=0     threadIdx.z=1
blockIdx.x=0     blockIdx.y=0     blockIdx.z=0     threadIdx.x=0     threadIdx.y=1     threadIdx.z=1
blockIdx.x=0     blockIdx.y=0     blockIdx.z=0     threadIdx.x=1     threadIdx.y=1     threadIdx.z=1
blockIdx.x=0     blockIdx.y=0     blockIdx.z=0     threadIdx.x=0     threadIdx.y=2     threadIdx.z=1
blockIdx.x=0     blockIdx.y=0     blockIdx.z=0     threadIdx.x=1     threadIdx.y=2     threadIdx.z=1
blockIdx.x=0     blockIdx.y=0     blockIdx.z=0     threadIdx.x=0     threadIdx.y=0     threadIdx.z=2
blockIdx.x=0     blockIdx.y=0     blockIdx.z=0     threadIdx.x=1     threadIdx.y=0     threadIdx.z=2
blockIdx.x=0     blockIdx.y=0     blockIdx.z=0     threadIdx.x=0     threadIdx.y=1     threadIdx.z=2
blockIdx.x=0     blockIdx.y=0     blockIdx.z=0     threadIdx.x=1     threadIdx.y=1     threadIdx.z=2
blockIdx.x=0     blockIdx.y=0     blockIdx.z=0     threadIdx.x=0     threadIdx.y=2     threadIdx.z=2
blockIdx.x=0     blockIdx.y=0     blockIdx.z=0     threadIdx.x=1     threadIdx.y=2     threadIdx.z=2
blockIdx.x=0     blockIdx.y=0     blockIdx.z=0     threadIdx.x=0     threadIdx.y=0     threadIdx.z=3
blockIdx.x=0     blockIdx.y=0     blockIdx.z=0     threadIdx.x=1     threadIdx.y=0     threadIdx.z=3
blockIdx.x=0     blockIdx.y=0     blockIdx.z=0     threadIdx.x=0     threadIdx.y=1     threadIdx.z=3
blockIdx.x=0     blockIdx.y=0     blockIdx.z=0     threadIdx.x=1     threadIdx.y=1     threadIdx.z=3
blockIdx.x=0     blockIdx.y=0     blockIdx.z=0     threadIdx.x=0     threadIdx.y=2     threadIdx.z=3
blockIdx.x=0     blockIdx.y=0     blockIdx.z=0     threadIdx.x=1     threadIdx.y=2     threadIdx.z=3
blockIdx.x=1     blockIdx.y=0     blockIdx.z=0     threadIdx.x=0     threadIdx.y=0     threadIdx.z=0
blockIdx.x=1     blockIdx.y=0     blockIdx.z=0     threadIdx.x=1     threadIdx.y=0     threadIdx.z=0
blockIdx.x=1     blockIdx.y=0     blockIdx.z=0     threadIdx.x=0     threadIdx.y=1     threadIdx.z=0
blockIdx.x=1     blockIdx.y=0     blockIdx.z=0     threadIdx.x=1     threadIdx.y=1     threadIdx.z=0
blockIdx.x=1     blockIdx.y=0     blockIdx.z=0     threadIdx.x=0     threadIdx.y=2     threadIdx.z=0
blockIdx.x=1     blockIdx.y=0     blockIdx.z=0     threadIdx.x=1     threadIdx.y=2     threadIdx.z=0
blockIdx.x=1     blockIdx.y=0     blockIdx.z=0     threadIdx.x=0     threadIdx.y=0     threadIdx.z=1
blockIdx.x=1     blockIdx.y=0     blockIdx.z=0     threadIdx.x=1     threadIdx.y=0     threadIdx.z=1
blockIdx.x=1     blockIdx.y=0     blockIdx.z=0     threadIdx.x=0     threadIdx.y=1     threadIdx.z=1
blockIdx.x=1     blockIdx.y=0     blockIdx.z=0     threadIdx.x=1     threadIdx.y=1     threadIdx.z=1
blockIdx.x=1     blockIdx.y=0     blockIdx.z=0     threadIdx.x=0     threadIdx.y=2     threadIdx.z=1
blockIdx.x=1     blockIdx.y=0     blockIdx.z=0     threadIdx.x=1     threadIdx.y=2     threadIdx.z=1
blockIdx.x=1     blockIdx.y=0     blockIdx.z=0     threadIdx.x=0     threadIdx.y=0     threadIdx.z=2
```

3. This program does vector addition. We follow the same steps performed in **question 1.** Create device memory, copy the vector data to the device memory , launch the kernel with <<<1,128>> config, which means 128 threads in one block. And then perform simple vector addition. As all threads are in one block in x direction we can use the simple threadIdx.x for a unique identifier and solve the vector addition problem.

```
/content# nvcc cuda03.cu -o cuda03 -arch=sm_75 &&  ./cuda03
253 + 135 = 388
215 + 113 = 328
223 + 155 = 378
116 + 52  = 168
90  + 145 = 235
184 + 172 = 356
119 + 55  = 174
180 + 112 = 292
150 + 121 = 271
175 + 248 = 423
175 + 84  = 259
18  + 216 = 234
70  + 186 = 256
18  + 111 = 129
103 + 107 = 210
183 + 135 = 318
247 + 149 = 396
99  + 111 = 210
175 + 184 = 359
71  + 188 = 259
230 + 60  = 290
22  + 8   = 30
75  + 238 = 313
146 + 30  = 176
87  + 35  = 122
27  + 132 = 159
157 + 210 = 367
22  + 229 = 251
176 + 153 = 329
109 + 126 = 235
190 + 8   = 198
182 + 27  = 209
65  + 21  = 86
```

5. This program is just to check what happens if we create more thread per block than the threshold. For modern GPU the threshold is 1024 thread per block no matter what the dimension is the number of threads must not exceed 1024. SO in this program an exception is thrown as shown below as we have used 1025 threads as it is beyond the threshold.