IT301 Assignment 9

NAME: SUYASH CHINTAWAR
ROLL NO.: 191IT109
TOPIC: CUDA PROGRAMMING - 1

NOTE: The codes have not been attached as already given in problems.

Q1. To know details of the device. Run the program and Explain the result.
SOLUTION:

Output:

```
✓ [13] Device count:1

        CUDA Device #0
        Name:Tesla K80
        Compute capability: 3.7
        Warp Size 32
        Total global memory:3407020032 bytes
        Total shared memory per block: 49152 bytes
        Total registers per block : 65536
        Clock rate: 823500 khz
        Maximum threads per block:1024
        Maximum dimension 0 of block: 1024
        Maximum dimension 1 of block: 1024
        Maximum dimension 2 of block: 64
        Maximum dimension 0 of grid: 2147483647
        Maximum dimension 1 of grid: 65535
        Maximum dimension 2 of grid: 65535
        Number of multiprocessors:13
```

Observation: Through the above output, we can see that the number of devices are 1 where the max no. of threads per block is 1024 and max dimension of a grid is 65535. The number of multiprocessors is 13.

Q2. Hello world program. Record the result and write the observation.
SOLUTION:

Output:

(continued...)

```
[14] %%cu
     #include<stdio.h>
     #include<cuda.h>
     __global__ void helloworld(void)
     {
     printf("Hello World from GPU\n");
     }
     int main() {
     helloworld<<<1,10>>>();
     printf("Hello World\n");
     return 0;
     }

     Hello World
```

Observations: We see that only the “Hello world” of the main program has been obtained; we get nothing from the function call. This happens because CUDA kernels are asynchronous. The CPU will continue executing the main program and will not wait for the device function to complete its execution. Due to this, the program ends before the device function is executed and we get only one “Hello world”.

Q3. Program to perform c[i] =a[i]+b[i]; Here, c[i] is calculated for all i. But results are displayed only for a few c[i]. Explain your observation. Run the program for the following and note down the time.

SOLUTION:

(continued...)

(a) vecAdd<<<1,100>>>(d_a, d_b, d_c, n);

Output:

```
[15] c[0]=2.000000
     c[10]=22.000000
     c[20]=42.000000
     c[30]=62.000000
     c[40]=82.000000
     c[50]=102.000000
     c[60]=122.000000
     c[70]=142.000000
     c[80]=162.000000
     c[90]=182.000000
     Time to generate: 0.1270000000 ms
```

(b) vecAdd<<<1,50>>>(d_a, d_b, d_c, n);

Output:

```
[17] c[0]=2.000000
     c[10]=22.000000
     c[20]=42.000000
     c[30]=62.000000
     c[40]=82.000000
     c[50]=0.000000
     c[60]=0.000000
     c[70]=0.000000
     c[80]=0.000000
     c[90]=0.000000
     Time to generate: 0.1820000000 ms
```

(c) vecAdd<<<2,50>>>(d_a, d_b, d_c, n);

Output:

```
[16] c[0]=2.000000
     c[10]=22.000000
     c[20]=42.000000
     c[30]=62.000000
     c[40]=82.000000
     c[50]=102.000000
     c[60]=122.000000
     c[70]=142.000000
     c[80]=162.000000
     c[90]=182.000000
     Time to generate: 0.1220000000 ms
```

Observations:

Execution Times

(num_blocks, num_threads)	(1,100)	(1,50)	(2,50)
Time (in ms)	0.127	0.182	0.122

We see that in (a) and (c) , we have a total of 1*100 and 2* 50 both equal to a total of 100 threads, sufficient to compute the whole host array h_c with 100 elements, but in (b) we only have 50 threads from ids 0-49 which are not fully sufficient, that's why only half of the array sums have been calculated (from indices 0-49). In both cases (a) and (c), the thread ids will range from 0-99 even though the block ids are different. This happens because the number of threads per block is different and so we get thread ids ranging from 0-99 itself. Whereas in (b), thread ids will range from 0-49 only as we only have one block containing 50 threads.

THANK YOU