PARALLEL DISTRIBUTING COMPUTING

***————————————————786**
————————————————*

Name: Muhammad Sharjeel Akhtar

Roll No: 20p-0101

Assignment No: 02

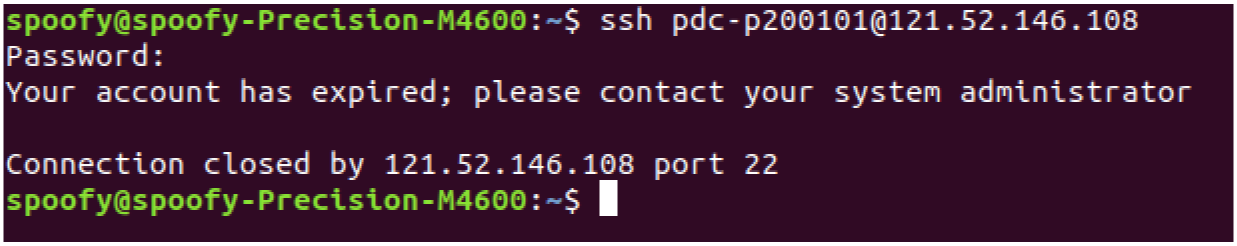
Submitted To Respected
Sir: Dr Omer Usman Khan

Section: BCS-7F

Task 1: Getting Ready

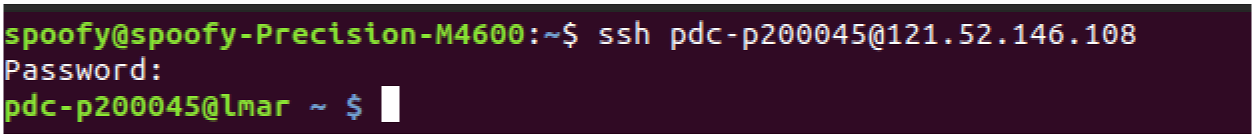
P.S: Getting Login Issue With My Personal Account Which Was Working Fine Few Days Back So Using Friend Account For Testing Of Codes For The Assignment

Error:



```
spoofy@spoofy-Precision-M4600:~$ ssh pdc-p200101@121.52.146.108
Password:
Your account has expired; please contact your system administrator

Connection closed by 121.52.146.108 port 22
spoofy@spoofy-Precision-M4600:~$ 
```

Doing SSH on Another Account:

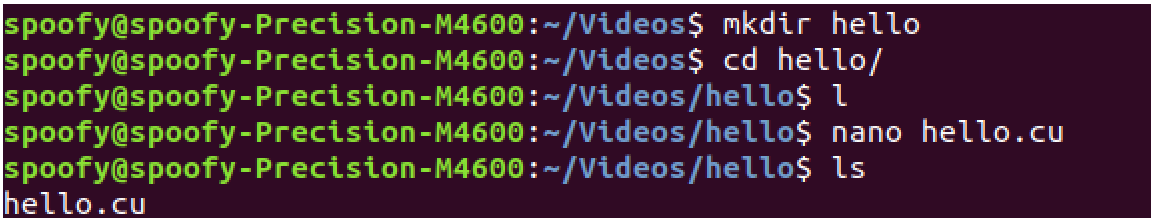


```
spoofy@spoofy-Precision-M4600:~$ ssh pdc-p200045@121.52.146.108
Password:
pdc-p200045@lmar ~ $ 
```

SUCESSFULLY LOGGED IN

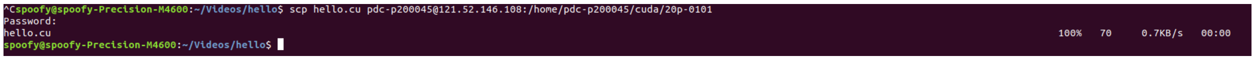
Task 2: Hello World

Initially create file locally in my computer,



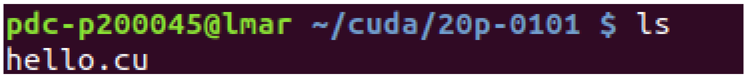
```
spoofy@spoofy-Precision-M4600:~/Videos$ mkdir hello
spoofy@spoofy-Precision-M4600:~/Videos$ cd hello/
spoofy@spoofy-Precision-M4600:~/Videos/hello$ l
spoofy@spoofy-Precision-M4600:~/Videos/hello$ nano hello.cu
spoofy@spoofy-Precision-M4600:~/Videos/hello$ ls
hello.cu
```

Sending it to cloud pc,



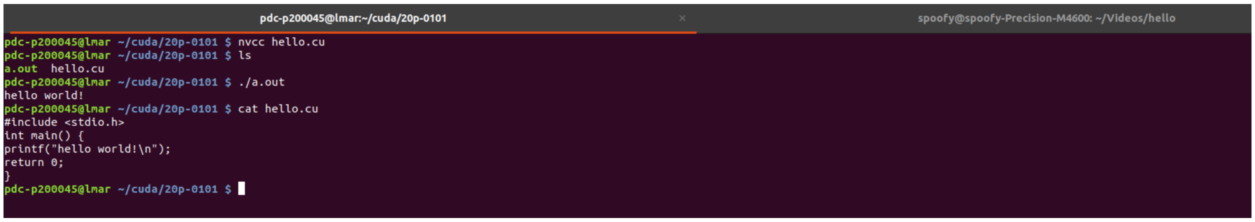
```
^Cspoofy@spoofy-Precision-M4600:~/Videos/hello$ scp hello.cu pdc-p200045@121.52.146.108:/home/pdc-p200045/cuda/20p-0101
Password:
hello.cu                                                                                              100%   70     0.7KB/s   00:00
spoofy@spoofy-Precision-M4600:~/Videos/hello$
```

VERIFICATION ON CLOUD:



```
pdc-p200045@lmar ~/cuda/20p-0101 $ ls
hello.cu
```

RUNNING:


pdc-p200045@lmar:~/cuda/20p-0101
spoofy@spoofy-Precision-M4600: ~/Videos/hello

```
pdc-p200045@lmar ~/cuda/20p-0101 $ nvcc hello.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ ls
a.out  hello.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
hello world!
pdc-p200045@lmar ~/cuda/20p-0101 $ cat hello.cu
#include <stdio.h>
int main() {
printf("hello world!\n");
return 0;
}
pdc-p200045@lmar ~/cuda/20p-0101 $
```

Task 5: Playing with 1D GPU indices

Dummy Code that WE ARE GOING to USE:

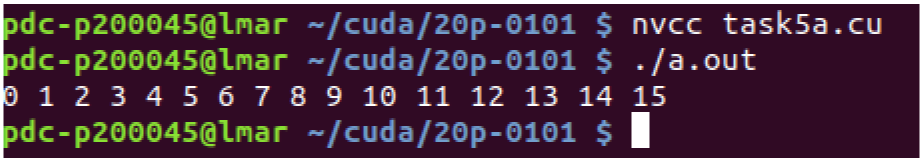
```
/* Name: task5.cu
*/
#include <stdio.h>
__global__ void myHelloOnGPU(int *array) {
// Position 1: To write Code here later
}
int main() {
int N = 16;
int *cpuArray = (int*)malloc(sizeof(int)*N);
int *gpuArray;
cudaMalloc((void **)&gpuArray, sizeof(int)*N);
// Position 2: To write Code here later
cudaMemcpy(cpuArray, gpuArray, sizeof(int)*N,
cudaMemcpyDeviceToHost);
int i;
for (i = 0; i < N; i++) {
```

```
printf("%d ", cpuArray[i]);
}
printf("\n");
return 0;
}
```

1. Task 5a

```
Position 2: myHelloOnGPU<<<N, 1>>>(gpuArray);
Position 1: array[blockIdx.x] = blockIdx.x
```

Result:

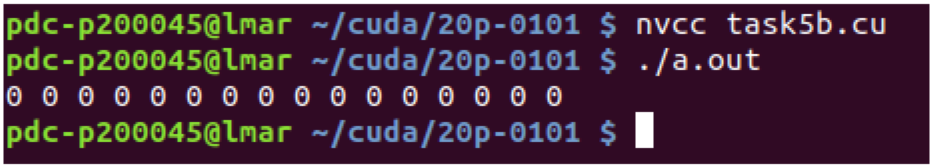


```
pdc-p200045@lmar ~/cuda/20p-0101 $ nvcc task5a.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
pdc-p200045@lmar ~/cuda/20p-0101 $ 
```

2. Task 5b

```
Position 2: myHelloOnGPU<<<N, 1>>>(gpuArray);
Position 1: array[blockIdx.x] = threadIdx.x
```

Result:

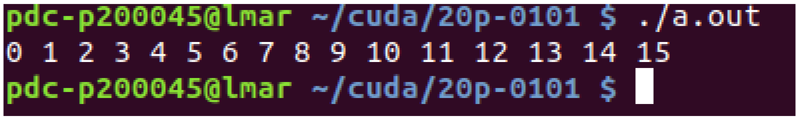


```
pdc-p200045@lmar ~/cuda/20p-0101 $ nvcc task5b.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
pdc-p200045@lmar ~/cuda/20p-0101 $ 
```

3.Task 5c

```
Position 2: myHelloOnGPU<<<1, N>>>(gpuArray);
Position 1: array[threadIdx.x] = threadIdx.x
```

Result:

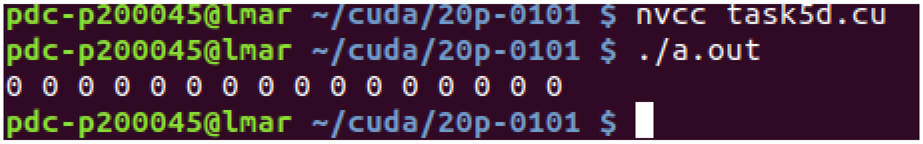


```
pdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
pdc-p200045@lmar ~/cuda/20p-0101 $ 
```

4. Task 5d

```
Position 2: myHelloOnGPU<<<1, N>>>(gpuArray);
Position 1: array[threadIdx.x] = blockIdx.x
```

Result:

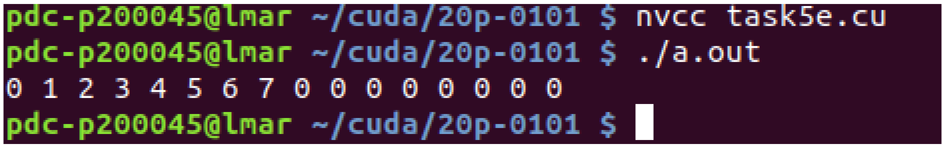


```
pdc-p200045@lmar ~/cuda/20p-0101 $ nvcc task5d.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
pdc-p200045@lmar ~/cuda/20p-0101 $ 
```

5. Task 5e

```
Position 2: myHelloOnGPU<<<1, N/2>>>(gpuArray);
    Position 1: array[threadIdx.x] = threadIdx.x
```

Result:

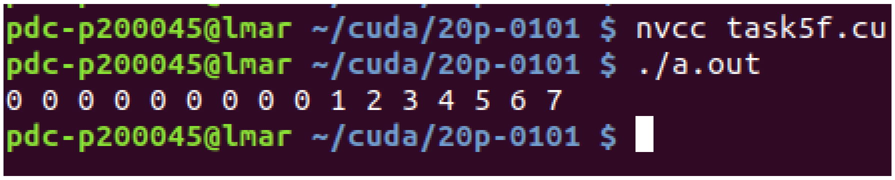


```
pdc-p200045@lmar ~/cuda/20p-0101 $ nvcc task5e.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
0 1 2 3 4 5 6 7 0 0 0 0 0 0 0 0
pdc-p200045@lmar ~/cuda/20p-0101 $ 
```

6. Task 5f

```
Position 2: myHelloOnGPU<<<1, N/2>>>(gpuArray);
Position 1: array[threadIdx.x + blockDim.x] = threadIdx.x
```

Result:

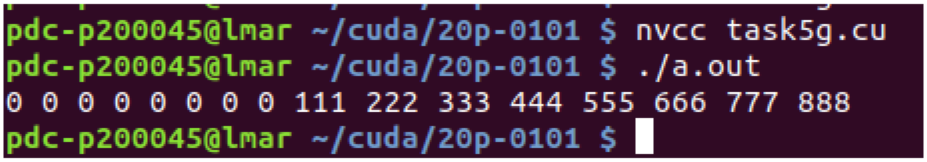


```
pdc-p200045@lmar ~/cuda/20p-0101 $ nvcc task5f.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 7
pdc-p200045@lmar ~/cuda/20p-0101 $
```

7. Task 5g

```
Position 2: myHelloOnGPU<<<N/2, 1>>>(gpuArray);
Position 1: array[blockIdx.x + gridDim.x] = 111 *(blockIdx.x + 1
```

Result:

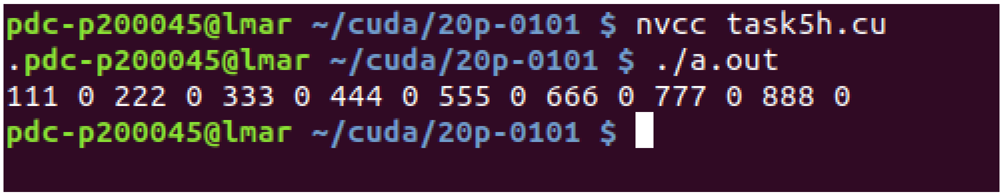


```
pdc-p200045@lmar ~/cuda/20p-0101 $ nvcc task5g.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
0 0 0 0 0 0 0 0 111 222 333 444 555 666 777 888
pdc-p200045@lmar ~/cuda/20p-0101 $
```

8. Task 5h: What should be Position 1 and 2 in order to obtain the following output:

```
Position 2: myHelloOnGPU<<<N/2, 1>>>(gpuArray);
Position 1: array[blockIdx.x * 2] = 111 *(blockIdx.x + 1);
```

Result:

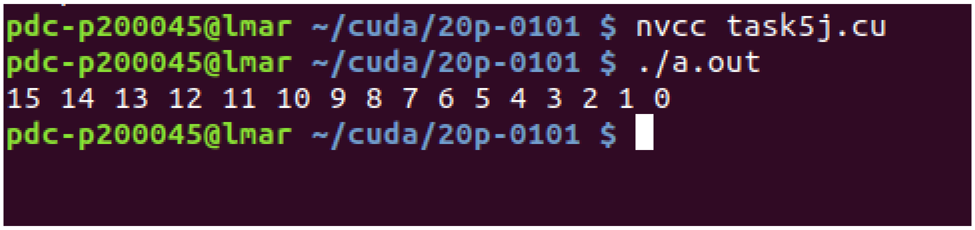


```
pdc-p200045@lmar ~/cuda/20p-0101 $ nvcc task5h.cu
.pdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
111 0 222 0 333 0 444 0 555 0 666 0 777 0 888 0
pdc-p200045@lmar ~/cuda/20p-0101 $
```

9. Task 5j

```
Position 2: myHelloOnGPU<<<N, 1>>>(gpuArray);
Position 1: array[blockIdx.x] = gridDim.x - blockIdx.x - 1;
```

Result:

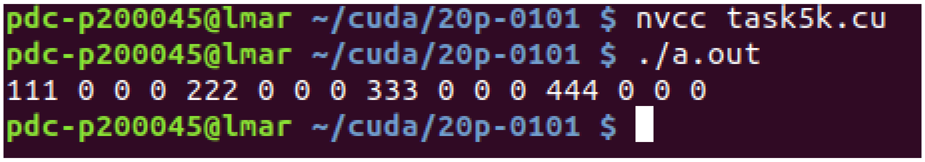


```
pdc-p200045@lmar ~/cuda/20p-0101 $ nvcc task5j.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
pdc-p200045@lmar ~/cuda/20p-0101 $
```

10. Task 5k

```
Position 2: myHelloOnGPU<<<N/4, N/4>>>(gpuArray);
Position 1: array[blockIdx.x * blockDim.x] = 111*(blockIdx.x +
```

Result:

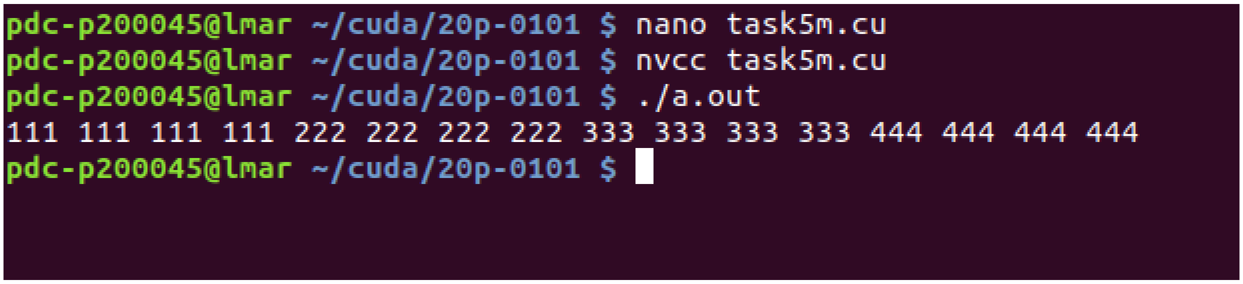


```
pdc-p200045@lmar ~/cuda/20p-0101 $ nvcc task5k.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
111 0 0 0 222 0 0 0 333 0 0 0 444 0 0 0
pdc-p200045@lmar ~/cuda/20p-0101 $
```

11. Task 5m

```
Position 2: myHelloOnGPU<<<N/4, N/4>>>(gpuArray);
Position 1: array[blockIdx.x * blockDim.x + threadIdx.x] =
111*(blockIdx.x + 1);
```

Result

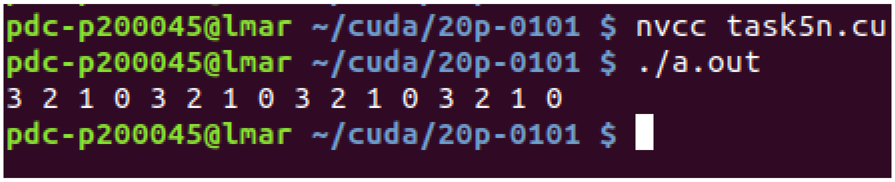


```
pdc-p200045@lmar ~/cuda/20p-0101 $ nano task5m.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ nvcc task5m.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
111 111 111 111 222 222 222 222 333 333 333 333 444 444 444 444
pdc-p200045@lmar ~/cuda/20p-0101 $
```

Task 5n: What should be Position 1 and 2 in order to obtain the following output:

```
Position 2: myHelloOnGPU<<<N/4, N/4>>>(gpuArray);
Position 1: array[blockIdx.x * blockDim.x + threadIdx.x] = blocl
threadIdx.x – 1;
```

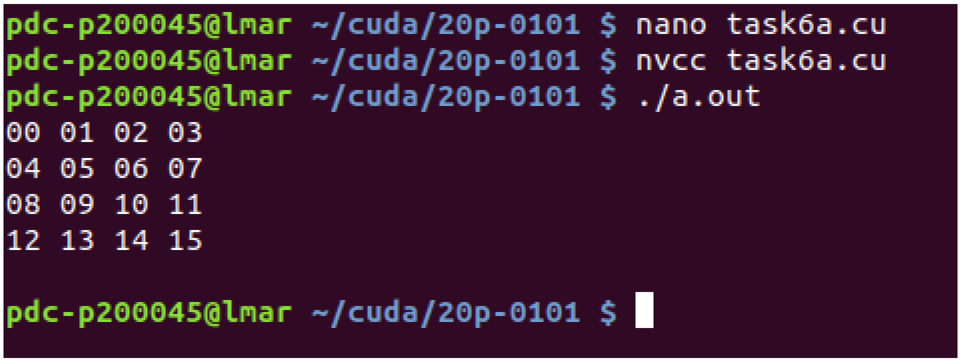
Result:



```
pdc-p200045@lmar ~/cuda/20p-0101 $ nvcc task5n.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
3 2 1 0 3 2 1 0 3 2 1 0 3 2 1 0
pdc-p200045@lmar ~/cuda/20p-0101 $ 
```

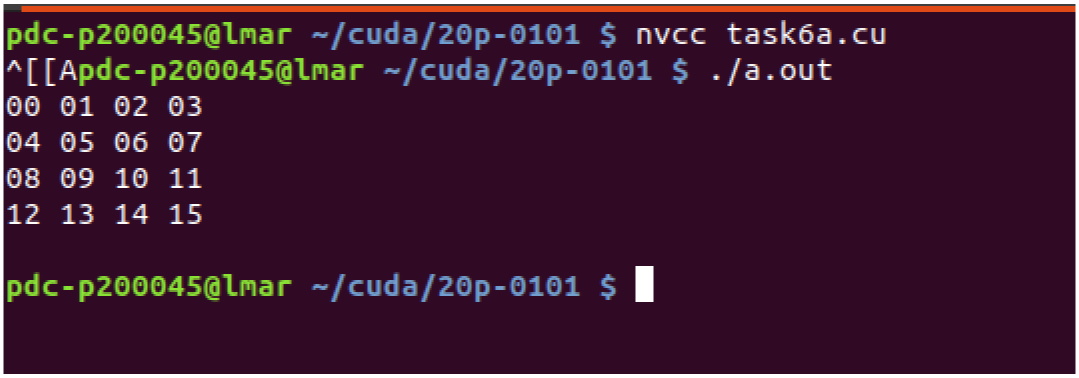
Task 6: Playing with 2D GPU indices.

1. Task 6a:



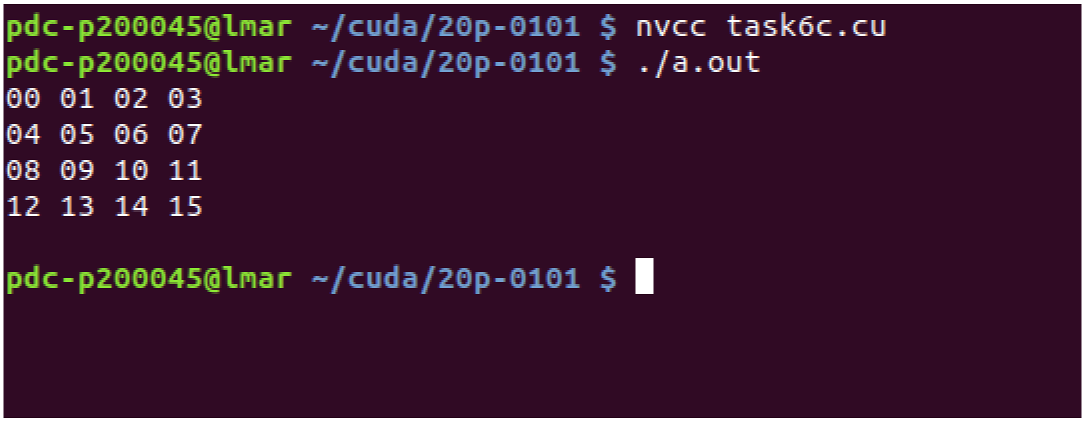
```
pdc-p200045@lmar ~/cuda/20p-0101 $ nano task6a.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ nvcc task6a.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
00 01 02 03
04 05 06 07
08 09 10 11
12 13 14 15

pdc-p200045@lmar ~/cuda/20p-0101 $ 
```

2. Task 6b:



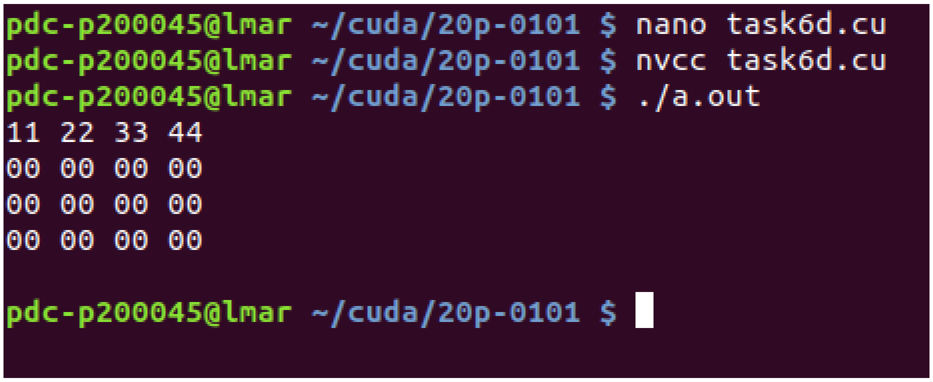
```
pdc-p200045@lmar ~/cuda/20p-0101 $ nvcc task6a.cu
^[[Apdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
00 01 02 03
04 05 06 07
08 09 10 11
12 13 14 15

pdc-p200045@lmar ~/cuda/20p-0101 $ 
```

3. Task 6c:



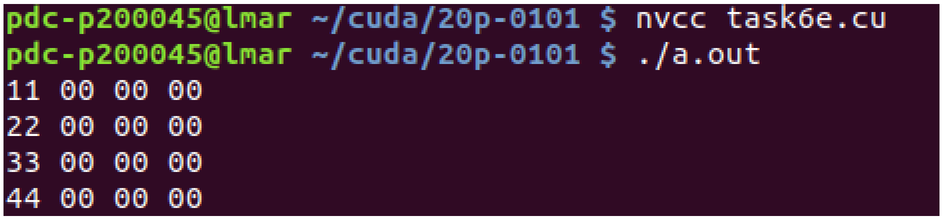
```
pdc-p200045@lmar ~/cuda/20p-0101 $ nvcc task6c.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
00 01 02 03
04 05 06 07
08 09 10 11
12 13 14 15

pdc-p200045@lmar ~/cuda/20p-0101 $ 
```

4. Task 6d:



```
pdc-p200045@lmar ~/cuda/20p-0101 $ nano task6d.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ nvcc task6d.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
11 22 33 44
00 00 00 00
00 00 00 00
00 00 00 00

pdc-p200045@lmar ~/cuda/20p-0101 $ 
```

5. Task 6e:

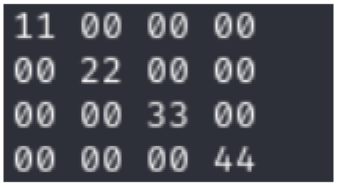


```
pdc-p200045@lmar ~/cuda/20p-0101 $ nvcc task6e.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
11 00 00 00
22 00 00 00
33 00 00 00
44 00 00 00
```

6. Task 6f:

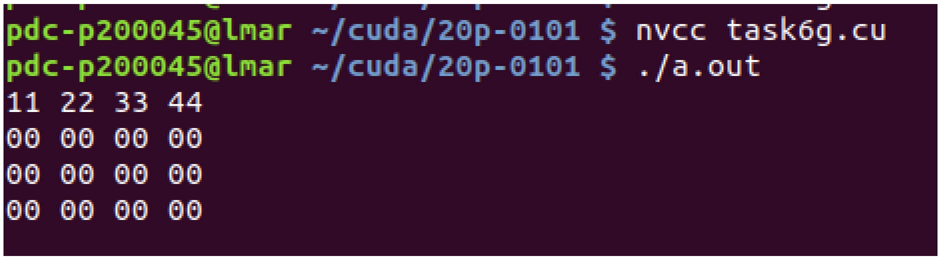
```
Position1 : int index = threadIdx.x * blockDim.x;
array[index] = (index % 5== 0) ? (index / 5+1)* 11: 0;
Position2: dim3 dimGrid(4,1,1); dim3 dimBlock(4,1,1);
```

Result:



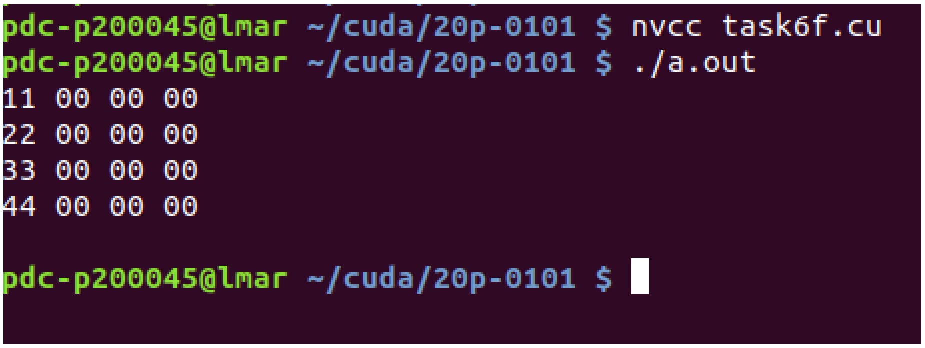
```
11 00 00 00
00 22 00 00
00 00 33 00
00 00 00 44
```

7. Task 6g:



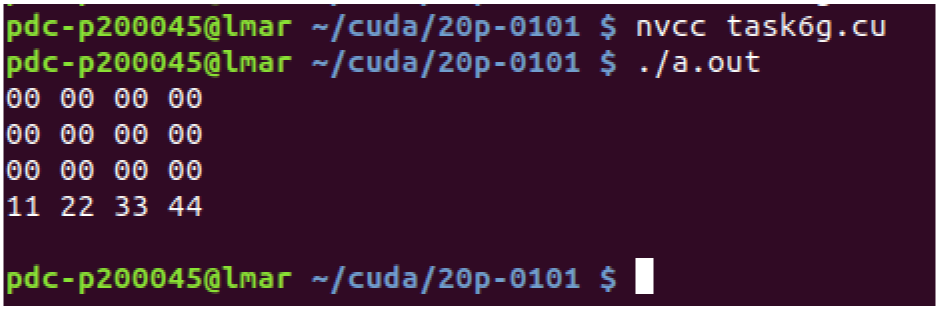
```
pdc-p200045@lmar ~/cuda/20p-0101 $ nvcc task6g.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
11 22 33 44
00 00 00 00
00 00 00 00
00 00 00 00
```

8. Task 6f:



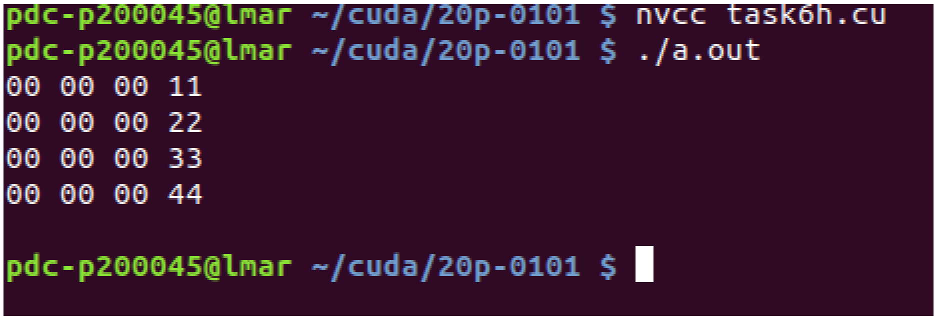
```
pdc-p200045@lmar ~/cuda/20p-0101 $ nvcc task6f.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
11 00 00 00
22 00 00 00
33 00 00 00
44 00 00 00

pdc-p200045@lmar ~/cuda/20p-0101 $ 
```

9. Task 6g:



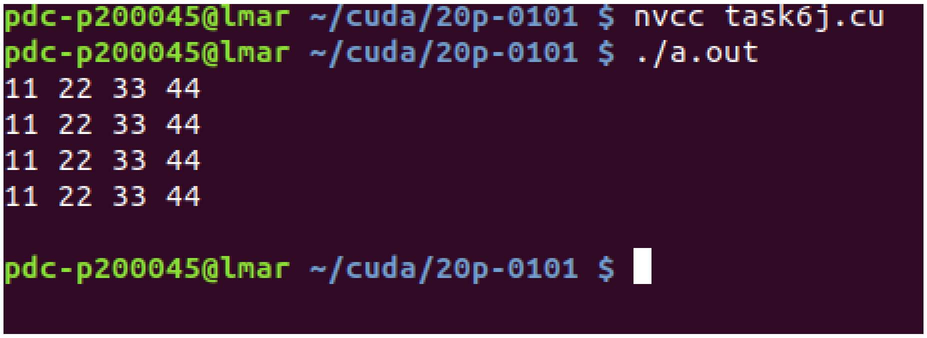
```
pdc-p200045@lmar ~/cuda/20p-0101 $ nvcc task6g.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
00 00 00 00
00 00 00 00
00 00 00 00
11 22 33 44

pdc-p200045@lmar ~/cuda/20p-0101 $ 
```

10. Task 6h:



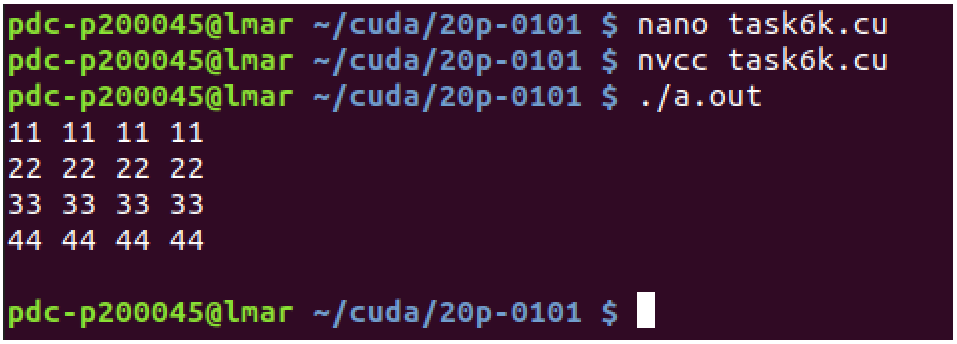
```
pdc-p200045@lmar ~/cuda/20p-0101 $ nvcc task6h.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
00 00 00 11
00 00 00 22
00 00 00 33
00 00 00 44

pdc-p200045@lmar ~/cuda/20p-0101 $ 
```

11. Task 6j:



```
pdc-p200045@lmar ~/cuda/20p-0101 $ nvcc task6j.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
11 22 33 44
11 22 33 44
11 22 33 44
11 22 33 44

pdc-p200045@lmar ~/cuda/20p-0101 $ 
```

12. Task 6k:

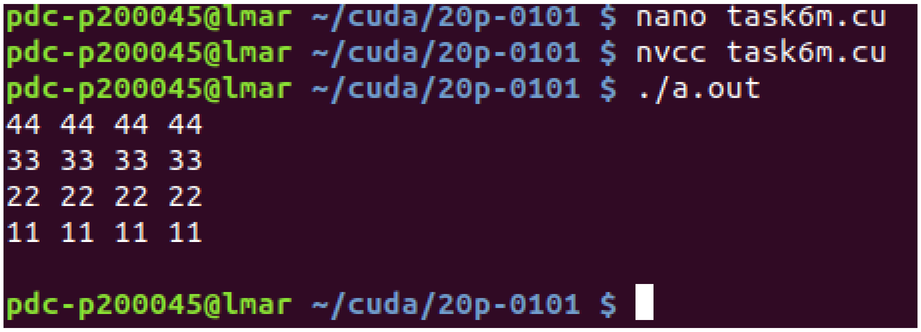


```
pdc-p200045@lmar ~/cuda/20p-0101 $ nano task6k.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ nvcc task6k.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
11 11 11 11
22 22 22 22
33 33 33 33
44 44 44 44

pdc-p200045@lmar ~/cuda/20p-0101 $ 
```

13. Task 6m:

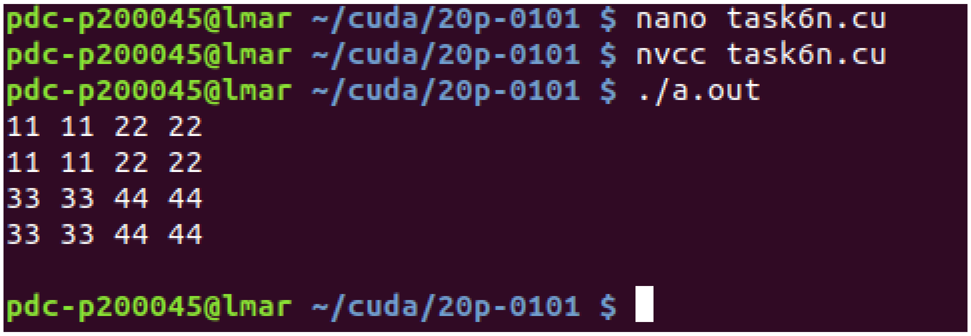
```
Position2: dim3 dimGrid(N/4,1,1) ; dim3 dimBlock(N/4 , 1 , 1);
Position1: int index = threadIdx.x + blockIdx.x * blockDim.x;
array[index] = (4- blockIdx.x) * 11;
```

Result:



```
pdc-p200045@lmar ~/cuda/20p-0101 $ nano task6m.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ nvcc task6m.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
44 44 44 44
33 33 33 33
22 22 22 22
11 11 11 11

pdc-p200045@lmar ~/cuda/20p-0101 $ 
```

14. Task 6n:



```
pdc-p200045@lmar ~/cuda/20p-0101 $ nano task6n.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ nvcc task6n.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
11 11 22 22
11 11 22 22
33 33 44 44
33 33 44 44

pdc-p200045@lmar ~/cuda/20p-0101 $ 
```

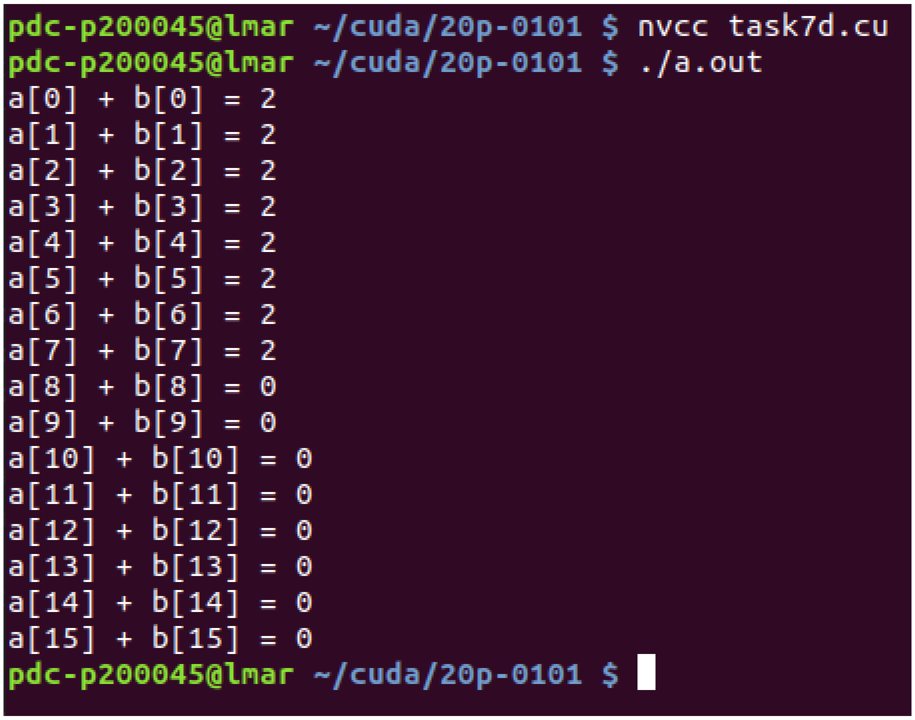
15. Task 6o:

```
Position 1:
int i = threadIdx.x + blockIdx.x * blockDim.x;
    if (i < 16) {
        array[i] = (i / 4 + 1) * 11;
    }
Position 2:
dim3 dimBlock(16); dim3 dimGrid(1);
```

Task 7: Matrix Addition

```
#include <stdio.h>
#include <stdlib.h>
#define N 16
__global__ void add(int *a, int *b, int *c) {
    int tid = blockIdx.x * blockDim.x + threadIdx.x; // Thread I

    // Position 1: Write Code here for vector addition
    if (tid < N) {
        c[tid] = a[tid] + b[tid];
    }
}

int main() {
    int *a, *b, *c, *da, *db, *dc, i;

    a = (int*)malloc(sizeof(int)*N); // allocate host mem
    b = (int*)malloc(sizeof(int)*N); // and assign random memory
    c = (int*)malloc(sizeof(int)*N); // for the result

    // Write code to initialize both a and b to 1’s.
    for (i = 0; i < N; i++) {
        a[i] = 1;
        b[i] = 1;
    }
    cudaMalloc((void **)&da, sizeof(int)*N);
    cudaMalloc((void **)&db, sizeof(int)*N);
    cudaMalloc((void **)&dc, sizeof(int)*N);

    cudaMemcpy(da, a, sizeof(int)*N, cudaMemcpyHostToDevice);
    cudaMemcpy(db, b, sizeof(int)*N, cudaMemcpyHostToDevice);

    dim3 dimGrid(N/8, 1, 1);
    dim3 dimBlock(N/4, 1, 1);

    add<<<dimGrid,dimBlock>>>(da, db, dc);

    cudaMemcpy(c, dc, sizeof(int)*N, cudaMemcpyDeviceToHost);

    for (i = 0; i < N; i++) {
        printf("a[%d] + b[%d] = %d\n", i, i, c[i]);
    }

    // Free allocated memory
    free(a);
        free(b);
    free(c);
    cudaFree(da);
    cudaFree(db);
    cudaFree(dc);

    return 0;
}
```

Result:



```
pdc-p200045@lmar ~/cuda/20p-0101 $ nvcc task7d.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
a[0] + b[0] = 2
a[1] + b[1] = 2
a[2] + b[2] = 2
a[3] + b[3] = 2
a[4] + b[4] = 2
a[5] + b[5] = 2
a[6] + b[6] = 2
a[7] + b[7] = 2
a[8] + b[8] = 0
a[9] + b[9] = 0
a[10] + b[10] = 0
a[11] + b[11] = 0
a[12] + b[12] = 0
a[13] + b[13] = 0
a[14] + b[14] = 0
a[15] + b[15] = 0
pdc-p200045@lmar ~/cuda/20p-0101 $
```

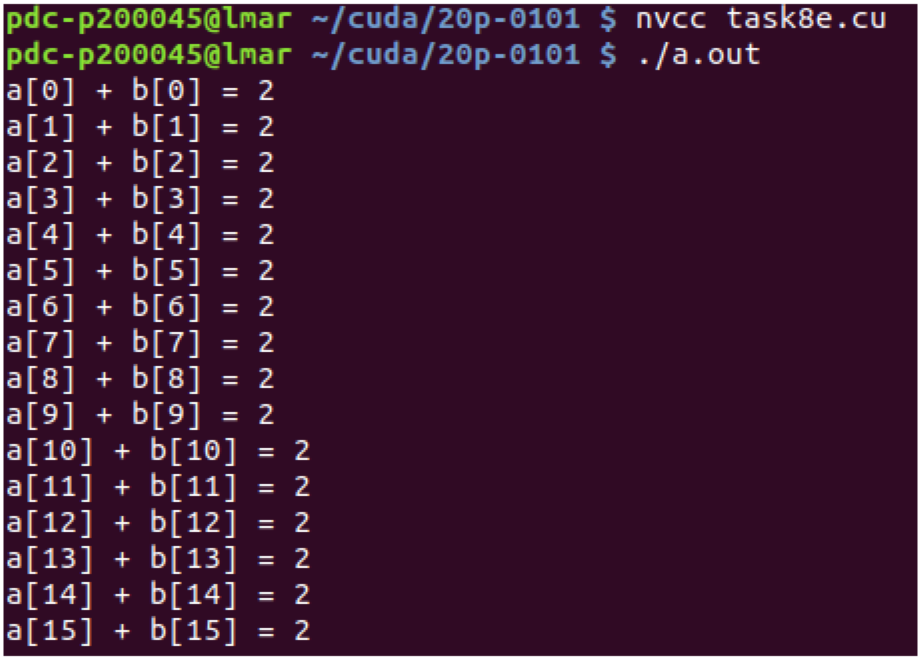
Task 8: Matrix Addition Slightly Complicated

```
#include <stdio.h>
#include <stdlib.h>

_global_ void add(int *a, int *b, int *c, int N) {
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    int row = blockIdx.y * blockDim.y + threadIdx.y;

    int index=row*blockDim.x*gridDim.x+col;
    c[index] = a[index] + b[index];
}

int main() {
    int *a, *b, *c, *da, *db, *dc, N = 16, i, j;

    a = (int*)malloc(sizeof(int)*N*N); // allocate host mem
    b = (int*)malloc(sizeof(int)*N*N); // and assign random memo
    c = (int*)malloc(sizeof(int)*N*N); // for the result
```

```
// Write code to initialize both a and b to 1’s.
for (i = 0; i < N*N; i++) {
    a[i] = 1;
    b[i] = 1;
}

cudaMalloc((void **)&da, sizeof(int)*N*N);
cudaMalloc((void **)&db, sizeof(int)*N*N);
cudaMalloc((void **)&dc, sizeof(int)*N*N);

cudaMemcpy(da, a, sizeof(int)*N*N, cudaMemcpyHostToDevice);
cudaMemcpy(db, b, sizeof(int)*N*N, cudaMemcpyHostToDevice);

dim3 dimGrid(N/8, N/8, 1);
dim3 dimBlock(N/8, N/8, 1);

add<<<dimGrid, dimBlock>>>(da, db, dc, N);

cudaMemcpy(c, dc, sizeof(int)*N*N, cudaMemcpyDeviceToHost);

for (j = 0; j < N; j++) {
    for (i = 0; i < N; i++) {
        printf("a[%d] + b[%d] = %d\n", j*N+i, j*N+i, c[j*N+i
    }
    printf("\n");
}

// Free allocated memory
free(a);
free(b);
free(c);
cudaFree(da);
cudaFree(db);
cudaFree(dc);
```

```
    return 0;
}
```

Result:



```
pdc-p200045@lmar ~/cuda/20p-0101 $ nvcc task8e.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
a[0] + b[0] = 2
a[1] + b[1] = 2
a[2] + b[2] = 2
a[3] + b[3] = 2
a[4] + b[4] = 2
a[5] + b[5] = 2
a[6] + b[6] = 2
a[7] + b[7] = 2
a[8] + b[8] = 2
a[9] + b[9] = 2
a[10] + b[10] = 2
a[11] + b[11] = 2
a[12] + b[12] = 2
a[13] + b[13] = 2
a[14] + b[14] = 2
a[15] + b[15] = 2
```

Task 9: Measurements

Title: To measure anything in CUDA, we can use the following from the CUDA Events API's:

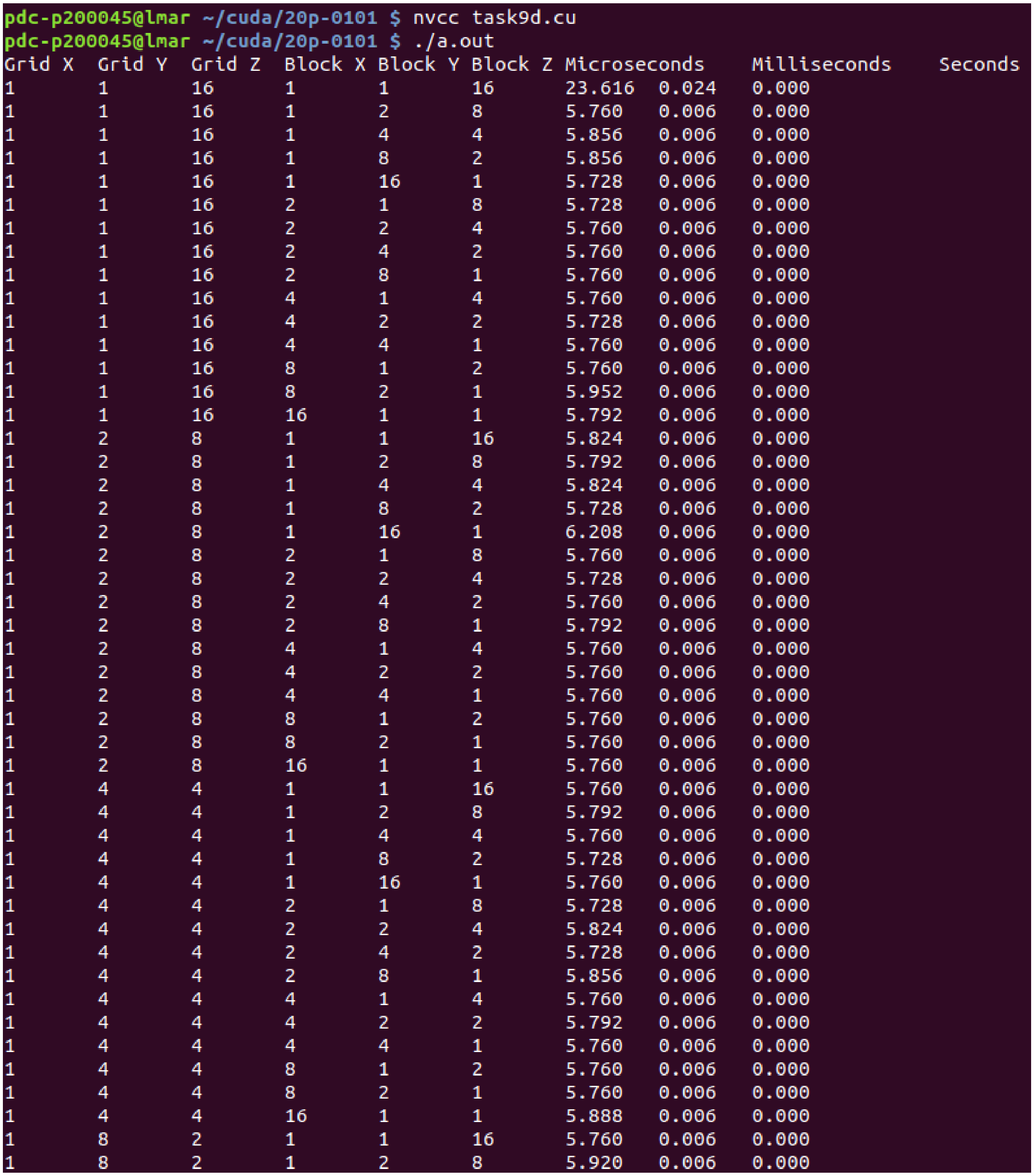
```
#include <stdio.h>
#include <cuda_runtime.h>

//Kernel Function for element-wise addition
__global__ void elementWiseAddition(int *a, int *b, int *result,
    int tid=blockIdx.x*blockDim.x+threadIdx.x;
    if(tid<N){
        result[tid]=a[tid]+b[tid];
    }
```

```
}

int main() {
    const int N=16;
    cudaEvent_t start,stop;
    float elapsed;

    //Host arrays
    int h_a[N],h_b[N],h_result[N];
    //Initialize host arrays
    for(int i=0;i<N;i++){
        h_a[i]=i;
        h_b[i]=2*i;
    }
    //Device arrays
    int *d_a,*d_b,*d_result;

    cudaMalloc((void **)&d_a,N*sizeof(int));
    cudaMalloc((void **)&d_b,N*sizeof(int));
    cudaMalloc((void **)&d_result,N*sizeof(int));

    //Copy data from host to device
    cudaMemcpy(d_a,h_a,N*sizeof(int),cudaMemcpyHostToDevice);
    cudaMemcpy(d_b,h_b,N*sizeof(int),cudaMemcpyHostToDevice);
    printf("Grid X\tGrid Y\tGrid Z\tBlock X\tBlock Y\tBlock Z\tN
    for(int gridX=1;gridX <= N;gridX*=2){
        for(int gridY=1;gridY<=N/gridX;gridY*=2){
            int gridZ=N/(gridX*gridY);

            for(int blockX=1;blockX<=N;blockX*=2){
                for(int blockY=1;blockY<=N/blockX;blockY*=2){
                    int blockZ=N/(blockX*blockY);

                    dim3 gridSize(gridX,gridY,gridZ);
                    dim3 blockSize(blockX,blockY,blockZ);
```

```
                   cudaEventCreate(&start);
                   cudaEventCreate(&stop);

                   cudaEventRecord(start,0);

                   //call the Kernel
                   elementWiseAddition<<<gridSize,blockSize>>>(
                   cudaEventRecord(stop,0);
                   cudaEventSynchronize(stop);
                   cudaEventElapsedTime(&elapsed,start,stop);

                   //Copy result from device to host
                   cudaMemcpy(h_result,d_result,N*sizeof(int),c

                   // Print the configuration and timing infor
                   printf("%d\t%d\t%d\t%d\t%d\t%d\t%0.3f\t%0.3
                          gridX, gridY, gridZ, blockX, blockY,
                          elapsed * 1000, elapsed, elapsed / 1

                   // Reset device memory for the next iteratio
                   cudaMemset(d_result, 0, N * sizeof(int));
               }
           }
       }
   }
// Free allocated memory on the device
   cudaFree(d_a);
   cudaFree(d_b);
   cudaFree(d_result);

   // Destroy CUDA events
   cudaEventDestroy(start);
   cudaEventDestroy(stop);
```

```
    return 0;
}
```

Result:



```
pdc-p200045@lmar ~/cuda/20p-0101 $ nvcc task9d.cu
pdc-p200045@lmar ~/cuda/20p-0101 $ ./a.out
Grid X  Grid Y  Grid Z  Block X Block Y Block Z Microseconds    Milliseconds    Seconds
1       1       16      1       1       16      23.616  0.024   0.000
1       1       16      1       2       8       5.760   0.006   0.000
1       1       16      1       4       4       5.856   0.006   0.000
1       1       16      1       8       2       5.856   0.006   0.000
1       1       16      1       16      1       5.728   0.006   0.000
1       1       16      2       1       8       5.728   0.006   0.000
1       1       16      2       2       4       5.760   0.006   0.000
1       1       16      2       4       2       5.760   0.006   0.000
1       1       16      2       8       1       5.760   0.006   0.000
1       1       16      4       1       4       5.760   0.006   0.000
1       1       16      4       2       2       5.728   0.006   0.000
1       1       16      4       4       1       5.760   0.006   0.000
1       1       16      8       1       2       5.760   0.006   0.000
1       1       16      8       2       1       5.952   0.006   0.000
1       1       16      16      1       1       5.792   0.006   0.000
1       2       8       1       1       16      5.824   0.006   0.000
1       2       8       1       2       8       5.792   0.006   0.000
1       2       8       1       4       4       5.824   0.006   0.000
1       2       8       1       8       2       5.728   0.006   0.000
1       2       8       1       16      1       6.208   0.006   0.000
1       2       8       2       1       8       5.760   0.006   0.000
1       2       8       2       2       4       5.728   0.006   0.000
1       2       8       2       4       2       5.760   0.006   0.000
1       2       8       2       8       1       5.792   0.006   0.000
1       2       8       4       1       4       5.760   0.006   0.000
1       2       8       4       2       2       5.760   0.006   0.000
1       2       8       4       4       1       5.760   0.006   0.000
1       2       8       8       1       2       5.760   0.006   0.000
1       2       8       8       2       1       5.760   0.006   0.000
1       2       8       16      1       1       5.760   0.006   0.000
1       4       4       1       1       16      5.760   0.006   0.000
1       4       4       1       2       8       5.792   0.006   0.000
1       4       4       1       4       4       5.760   0.006   0.000
1       4       4       1       8       2       5.728   0.006   0.000
1       4       4       1       16      1       5.760   0.006   0.000
1       4       4       2       1       8       5.728   0.006   0.000
1       4       4       2       2       4       5.824   0.006   0.000
1       4       4       2       4       2       5.728   0.006   0.000
1       4       4       2       8       1       5.856   0.006   0.000
1       4       4       4       1       4       5.760   0.006   0.000
1       4       4       4       2       2       5.792   0.006   0.000
1       4       4       4       4       1       5.760   0.006   0.000
1       4       4       8       1       2       5.760   0.006   0.000
1       4       4       8       2       1       5.760   0.006   0.000
1       4       4       16      1       1       5.888   0.006   0.000
1       8       2       1       1       16      5.760   0.006   0.000
1       8       2       1       2       8       5.920   0.006   0.000
```

END.