Assignment 4: Cuda Practice

For this task, you will require access to CUDA enabled machine. If your system or laptop has this logo, it should work fine (just ensure that the nvidia cuda) is available on your machine.

If your laptop does not have an NVIDIA card, then you can connect to the GPU machines available with campus. These are the same machines you used for hadoop and spark. So your accounts are already created.

Task 0: Getting Used to the Environment

Create a text file (hello.cu) containing the following text:

```
#include <stdio.h>

int main() {
     printf("hello world!\n");
     return 0;
}
```

If you have created this file on your local machine, you will need to carry it over to the GPU using your credentials mentioned. The exact command could be something like this:

```
scp hello.cu bda-pXXYYYY@121.52.146.108:/home/bda-pXXYYYY
```

You would need to send the file every-time you make any change to it. If you are on windows, you can use the utility Windows SCP client http://winscp.net/eng/index.php.

Now, from the HPC system, run the command:

```
nvcc hello.cu
```

It can be run as:

```
./a.out
```

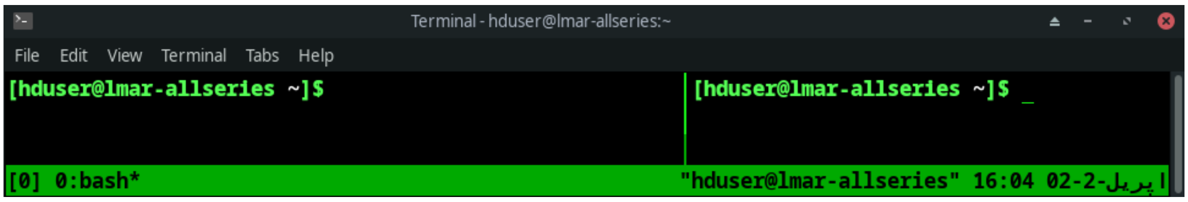
To make any changes, you will need to copy, compile, and run the file again and again. If you want to avoid it, you can work directly on the HPC system by using nano, vim, or any other editor of your choice. However, still you would have to exit from the editor environment to shell, and from shell to editor environment. This comes out to be a nuisance. For this, it would be better to create two shell sessions. In the first shell session, you can open the editor. While in the second session, you can continue compiling and running commands. The remaining description in this section guides you on some useful tools that you can use. You can skip them as they are not relevant to the lab per say.

Optional:

A useful tool is tmux emulator environment. To start it, run:

```
tmux
```

You will notice that a green line appears at the bottom and thats it. Once the green lined terminal window appears, type **CTRL + b** followed by **SHIFT + 5**. You should see the following window:


Terminal - hduser@lmar-allseries:~
File Edit View Terminal Tabs Help

```
[hduser@lmar-allseries ~]$                              [hduser@lmar-allseries ~]$ _
[0] 0:bash*                                "hduser@lmar-allseries" 16:04 02-2-اپریل
```

To move to the right shell, type **CTRL + b** followed by **right arrow** on numeric keypad.

To move to the left shell, type **CTRL + b** followed by **left arrow** on numeric keypad.

To change the size of a particular shell, type **CTRL + b + left** or **CTRL + b + right** repeatedly.

If you don't like the vertical arrangment, you can type **CTRL + b** followed by **SHIFT + "** for a horizontal arrangement.

You can then launch a text editor in any of the window you want as follows, and type in your code:

```
vim hello.cu
```

This is the **vim** editor. You will need the following shortcuts to edit the document:

- To enable editing the document, press **i** and then make changes.
- To disable editing the document, press **ESC**
- To save a document, type **:w**
- To exit the document, type **:q**
- To undo, type **u** in disable edit mode.
- To redo, type **CTRL+r** in disable edit mode.

For full list of these shortcuts, visit https://www.fprintf.net/vimCheatSheet.html

To prevent data loss due to accidental disconnection, you can use the screen utility. To use it, as soon as you login, type:

```
screen
```

You will be back at the same prompt you started with. Then, you may proceed with launching vim or tmux as you want. To use screen properly, continue with the following activities.

Launch your hello.cu file in the editor:

```
vim hello.cu
```

While the file is open, just close the terminal abruptly, and log back in again through another window. You can resume the previous activity by running:

```
screen -r
```

You will see that it is still at the session where your window was accidentally closed. If you have created multiple screens, you may see the following:

```
There are several suitable screens on:
        23592.pts-4.lmar        (Detached)
        23597.pts-4.lmar        (Detached)

Type "screen [-d] -r [pid.]tty.host" to resume one of them.
```

Choose the session you want to resume and then give the following command:

```
screen -r 23597.pts-4.lmar
```

To minimize a screen, you can type **CTRL + a** followed by **d**. Note that launching programs within screen means that you can leave your programs running on LMAR while you safely switch off your laptops.

Task 1: Playing with GPU Indices

For this task, you will need the following as a base code:

```
#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;
}
```

Then, in replacement of Position 1 and Position 2 above, provide the following:

```
Position 2: myHelloOnGPU<<<N, 1>>>(gpuArray);

Position 1: array[blockIdx.x] = blockIdx.x
```

Compile and run the code. You should be able to see the following:

```
nvcc hello.cu && ./a.out

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Now, attempt the following questions:

Question 1. What is the output if you use the following:

```
Position 2: myHelloOnGPU<<<N, 1>>>(gpuArray);

Position 1: array[blockIdx.x] = threadIdx.x
```

Question 2. What is the output if you use the following:

```
Position 2: myHelloOnGPU<<<1, N>>>(gpuArray);

Position 1: array[threadIdx.x] = threadIdx.x
```

Question 3. What is the output if you use the following:

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

Question 4. What is the output if you use the following:

```
Position 2: myHelloOnGPU<<<1, N/2>>>(gpuArray);

Position 1: array[threadIdx.x] = threadIdx.x
```

Question 5. What is the output if you use the following:

```
Position 2: myHelloOnGPU<<<1, N/2>>>(gpuArray);

Position 1: array[threadIdx.x + blockDim.x] = threadIdx.x
```

Question 6. What is the output if you use the following:

```
Pos. 2: myHelloOnGPU<<<N/2, 1>>>(gpuArray);

Pos. 1: array[blockIdx.x+gridDim.x] = 111*(blockIdx.x+1);
```

Question 7. What value of Position 1 and 2 would be required to get the following output?

```
111 0 222 0 333 0 444 0 555 0 666 0 777 0 888 0
```

Question 8. What is the output if you use the following:

```
Position 2: myHelloOnGPU<<<N, 1>>>(gpuArray);

Position 1: array[blockIdx.x] = gridDim.x - blockIdx.x - 1;
```

Question 9. What is the output if you use the following:

```
Pos. 2: myHelloOnGPU<<<N/4, N/4>>>(gpuArray);

Pos. 1: array[blockIdx.x*blockDim.x]=111*(blockIdx.x + 1);
```

Question 10. What is the output if you use the following:

```
Pos. 2: myHelloOnGPU<<<N/4, N/4>>>(gpuArray);

Pos. 1: array[blockIdx.x * blockDim.x + threadIdx.x] =
111*(blockIdx.x + 1);
```

Question 11. What value of Position 1 and 2 would be required to get the following output?

```
3 2 1 0 3 2 1 0 3 2 1 0 3 2 1 0
```

Task 2: Playing with 2D GPU indices

Type in the following as base code.

```
/* Name: task6.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
    myHelloOnGPU<<<dimGrid, dimBlock>>>(gpuArray);

    cudaMemcpy(cpuArray, gpuArray, sizeof(int)*N,
                                   cudaMemcpyDeviceToHost);

    int i, j;
    for (j = 0; j < N/4; j++) {
         for (i = 0; i < N/4; i++) {
              printf("%2.2d ", cpuArray[j*N/4+i]);
         }
         printf("\n");
    }
    printf("\n");

    return 0;
}
```

Once done, attempt the following:

1. Task 6a: What output would you see if you use the following:

```
Position 2: dim3 dimGrid(N, 1, 1); dim3 dimBlock(1, 1, 1);

Position 1: array[blockIdx.x] = blockIdx.x;
```

2. Task 6b: What output would you see if you use the following:

```
Position 2: dim3 dimGrid(N, 1, 1); dim3 dimBlock(1, 1, 1);

Position 1: array[blockIdx.y] = blockIdx.y;
```

3. Task 6c: What output would you see if you use the following:

```
Position 2: dim3 dimGrid(1, N, 1); dim3 dimBlock(1, 1, 1);

Position 1: array[blockIdx.y] = blockIdx.y;
```

4. Task 6d: What output would you see if you use the following:

```
Position 2: dim3 dimGrid(N/4, 1, 1); dim3 dimBlock(1, 1, 1);

Position 1: array[blockIdx.x] = 11 * (blockIdx.x + 1);
```

5. Task 6e: What output would you see if you use the following:

```
Position 2: dim3 dimGrid(N/4, 1, 1); dim3 dimBlock(1, 1, 1);

Position 1: array[blockIdx.x * gridDim.x] = 11 * (blockIdx.x + 1);
```

6. Task 6f: What should be Position 1 and 2 in order to obtain the following output:

```
bda-p101234@lmar ~/cuda $ nvcc task6.cu && ./a.out
11 00 00 00
00 22 00 00
```

```
00 00 33 00
00 00 00 44
```

7. Task 6g: What output would you see if you use the following:

```
Position 2: dim3 dimGrid(1, 1, 1); dim3 dimBlock(N/4, 1, 1);

Position 1: array[threadIdx.x] = 11 * (threadIdx.x + 1);
```

8. Task 6f: What output would you see if you use the following:

```
Position 2: dim3 dimGrid(1, 1, 1); dim3 dimBlock(N/4, 1, 1);

Position 1: array[threadIdx.x*blockDim.x] = 11*(threadIdx.x + 1);
```

9. Task 6g: What output would you see if you use the following:

```
Position 2: dim3 dimGrid(1, 1, 1); dim3 dimBlock(N/4, 1, 1);

Position 1: array[threadIdx.x + blockDim.x * (blockDim.x-1)] =
11*(threadIdx.x + 1);
```

10. Task 6h: What output would you see if you use the following:

```
Position 2: dim3 dimGrid(1, 1, 1); dim3 dimBlock(N/4, 1, 1);

Position 1: array[threadIdx.x * blockDim.x + (blockDim.x-1)] =
11*(threadIdx.x + 1);
```

11. Task 6j: What output would you see if you use the following (Important):

```
Position 2: dim3 dimGrid(N/4, 1, 1); dim3 dimBlock(N/4, 1, 1);

Position 1: array[blockIdx.x * blockDim.x + threadIdx.x] =
11*(threadIdx.x + 1);
```

12. Task 6k: What output would you see if you use the following:

```
Position 2: dim3 dimGrid(N/4, 1, 1); dim3 dimBlock(N/4, 1, 1);

Position 1: array[blockIdx.x * blockDim.x + threadIdx.x] =
11*(blockIdx.x + 1);
```

13. Task 6m: What should be Position 1 and 2 in order to obtain the following output:

```
bda-p101234@lmar ~/cuda $ nvcc task6.cu && ./a.out
44 44 44 44
33 33 33 33
22 22 22 22
11 11 11 11
```

14. Task 6n: What output would you see if you use the following (This is Fishy; watch closely):

```
Position 2: dim3 dimGrid(N/8, N/8, 1); dim3 dimBlock(N/8, N/8, 1);

Position 1: int index_x = blockIdx.x * blockDim.x + threadIdx.x;
            int index_y = blockIdx.y * blockDim.y + threadIdx.y;
            array[index_y * blockDim.x * blockDim.y + index_x] =
               11 * ((blockIdx.y * gridDim.x + blockIdx.x) + 1);
```

Showing output:

```
bda-p101234@lmar ~/cuda $ nvcc task6.cu && ./a.out
11 11 22 22
11 11 22 22
33 33 44 44
33 33 44 44
```

15. Task 6o: What should be Position 1 and 2 in order to obtain the following output:

```
bda-p101234@lmar ~/cuda $ nvcc task6.cu && ./a.out
44 44 33 33
44 44 33 33
22 22 11 11
22 22 11 11
```

Submission

Send in your code/jupyter notebooks and answer to your questions to get graded for this task.