Walk Through Assignment for CUDA

Task 1: Getting Ready

Login using the following credentials to the LMAR HPC facility. Your user-accounts are already created on this machine:

- Username: **cdc-p101234** (substitute your roll number here)
- Password: **p101324** (substitute your roll number here)
- IP Address: **121.52.146.108**

You can log into **LMAR** as follows:

1. **From Linux:**

   Just give the following command from any terminal.

```
ssh 121.52.146.108 -l cdc-p101234 -XY
```

   from any terminal

2. **From Windows:**

   1. Download and run/install the Putty SSH client from: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
   2. Specify the login credentials and connect.

Note: For first time login, you will be prompted to change the password (See below). Enter (1) your current password, followed by (2) your new password. You will not see any movement of cursor on screen when you do that.

```
You are required to change your password immediately (root enforced)
Changing password for cdc-p101234.
(current) UNIX password:
New password:
Retype new password:
cdc-p101234@lmar ~ $
```

Create a directory to contain your cuda code using the command:

```
cdc-p101234@lmar ~ $ mkdir cuda
```

Check to see whether the folder is created by using any of the following:

```
cdc-p101234@lmar ~ $ ls
cuda

cdc-p101234@lmar ~ $ ls -lh
drwxr-xr-x 2 cdc-p101234 cdc-p101234 4.0K May  2 11:24 cuda
```

Task 2: Hello World

Create a code (hello.cu) with the following text on your local machine:

```
/* Name: hello.cu
 */
```

```
#include <stdio.h>

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

You now need to transfer this code from your local machine to the server.

1. **From Linux:**

   Just give the following command from terminal of your **local** machine, where:
   Scp → Secure file transfer utility
   hello.cu → Is your code on local machine
   cdc-p101234 → Is your username on the remote machine
   @121.52.146.108 → Is the destination IP
   /home/cdc-p101234/cuda → Is the destination folder

   ```
   scp hello.cu cdc-p101234@121.52.146.108:/home/cdc-p101234/cuda
   ```

2. **From Windows:**

   1. Download the Windows SCP client from: http://winscp.net/eng/index.php
   2. Launch it and provide the login credentials.
   3. Browse to the right location and simply drag and drop the files.

Once your files are copied, run the following commands on LMAR to ensure everything is okay:

```
cdc-p101234@lmar ~ $ cd cuda

cdc-p101234@lmar ~/cuda $ ls -lh
-rw-r--r-- 1 cdc-p101234 cdc-p101234 95 May  2 11:26 hello.cu
```

Compile the code as follows:

```
cdc-p101234@lmar ~/cuda $ nvcc hello.cu
```

Run the code as follows:

```
cdc-p101234@lmar ~/cuda $ ./a.out
hello world
```

Note: If you want the executable to be named something other than a.out, compile and run as follows

```
cdc-p101234@lmar ~/cuda $ nvcc hello.cu -o hello
cdc-p101234@lmar ~/cuda $ ./hello
hello world
```

From this point onward, make changes to the code on your local machine, then transfer it to LMAR, and then run it on remote machine **EVERYTIME**.

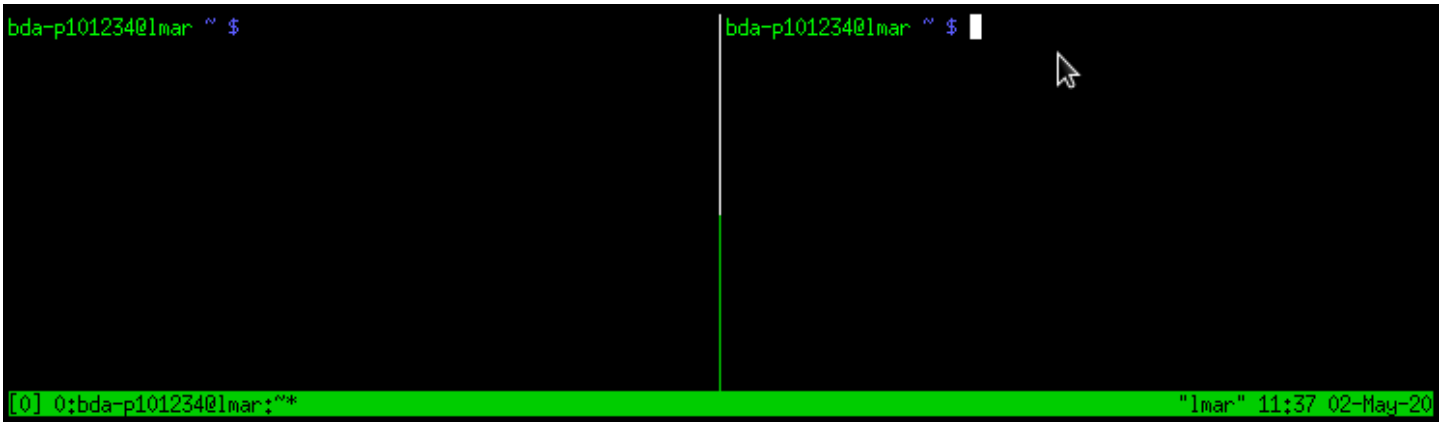
Task 3: Hello World in Another Way (Optional)

If you don’t want to copy the code everytime you make some changes, follow through with the following instructions.

Start a terminal emulator environment called tmux as follows:

```
cdc-p101234@lmar ~ $ 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:



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.

Launch a text editor in any of the window you want as follows, and type in your code:

```
cdc-p101234@lmar ~/cuda $ vim hello.cu
```

This is the **vim** editor. Note the following shortcuts for editing 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

Task 4: Prevent Data Loss in accidental Disconnection (Optional)

Sometimes you just have to reboot your system, or there is a network outage which results in disconnection to LMAR and loss of whatever you were doing. To prevent this, do the following as soon as you login:

```
cdc-p101234@lmar ~ $ screen
```

You will be back at the same prompt you started with. Then, you may proceed with launching vim or tmux as you want.

Try to do some stuff like changing the hello.cu file

```
cdc-p101234@lmar ~/cuda $ vim hello.cu
```

While the file is open, just close the terminal abruptly, and log back in again. This time, resume the screen as follows:

```
cdc-p101234@lmar ~ $ 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:

```
cdc-p101234@lmar ~ $ screen -r

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 which one you want to launch and resume as follows:

```
cdc-p101234@lmar ~ $ 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 5: Playing with 1D GPU indices

Type in the following as base code.

```
/* 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;
}
```

Once done, attempt the following:

1. Task 5a

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

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

   You should be able to see

```
cdc-p101234@lmar ~/cuda $ nvcc task5.cu && ./a.out
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

2. Task 5b: What output would you see if you use the following:

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

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

3. Task 5c: What output would you see if you use the following:

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

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

4. Task 5d: What output would you see if you use the following:

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

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

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

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

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

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

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

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

7. Task 5g: What output would you see if you use the following:

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

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

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

```
cdc-p101234@lmar ~/cuda $ nvcc task5.cu && ./a.out
111 0 222 0 333 0 444 0 555 0 666 0 777 0 888 0
```

9. Task 5j: What output would you see if you use the following:

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

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

10. Task 5k: What output would you see if you use the following:

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

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

11. Task 5m: What output would you see if you use the following:

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

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

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

```
cdc-p101234@lmar ~/cuda $ nvcc task5.cu && ./a.out
3 2 1 0 3 2 1 0 3 2 1 0 3 2 1 0
```

Task 6: 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:

```
cdc-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:

```
cdc-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:

```
cdc-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:

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

Task 7: Matrix Addition

Consider the following code:

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

__global__ void add(int *a, int *b, int *c) {
        // Position 1: To write Code here later
}

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

     a = (int*)malloc(sizeof(int)*N); // allocate host mem
```

```
    b = (int*)malloc(sizeof(int)*N); // and assign random
    c = (int*)malloc(sizeof(int)*N); // memory

    // Write code to initialize both a and b to 1's.

    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]);
    }
}
```

Write code at the point of **Position 1** to perform matrix addition.

Task 8: Matrix Addition Slightly Complicated

Consider the following code:

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

__global__ void add(int *a, int *b, int *c) {
        // Position 1: To write Code here later
}

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

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

    // Write code to initialize both a and b to 1's.

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

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

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

    for (j = 0; j < N/4; j++) {
        for (i = 0; i < N/4; i++) {
            printf("a[%d] + b[%d] = %d\n",     j*N/4+i,
                                               j*N/4+i, c[j*N/4+i]);
        }
        printf("\n");
    }
    printf("\n");
}
```

Write code at the point of **Position 1** to perform matrix addition.

Task 9: Measurements

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

```
cudaEvent_t start, stop;
float       elapsed;


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


// 0 is stream. Not covered in class
cudaEventRecord(start, 0);


// Call the Kernel


// We want all the threads to finish execution, so synchronize them
cudaEventSynchronize(stop);
// 0 is stream. Not covered in class
cudaEventRecord(stop, 0);


cudaEventElapsedTime(&elapsed, start, stop);


cudaEventDestroy(start);
cudaEventDestroy(stop);
```

Then, you can simply print the elapsed time (which is ms by default). For a problem size of N = 16, you will have the following factors:

16 x 1
8 x 2

4 x 4
2 x 8
1 x 16
4 x 2 x 2
2 x 4 x 2
2 x 2 x 4
2 x 2 x 2 x 2

You can place any of the above factor on any position of Grid (Grid X, Grid Y, Grid Z), and Block (Block X, Block Y, Block Z). Try different combinations to see which is giving you the minimum elapsed time.

Grid Dimensions			Block Dimensions			Elapsed Time		
Grid X	Grid Y	Grid Z	Block X	Block Y	Block Z	Microseconds	Milliseconds	Seconds
16	1	1	1	1	1			
2	2	2	2	1	1			
2	1	2	2	1	2			
And	So	on						

Give evidences of all the above tasks and submit on google classroom.