

Name: Siddharth Salunkhe

PRN No: 21510111

Batch : B5

High Performance Computing Lab

Practical No. 11

Title of practical: Understanding concepts of CUDA Programming

Problem Statement 1:

Execute the following program and check the properties of your GPGPU.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int deviceCount;
    cudaGetDeviceCount(&deviceCount);
    if (deviceCount == 0)
    {
        printf("There is no device supporting CUDA\n");
    }
    int dev;
    for (dev = 0; dev < deviceCount; ++dev)
    {
        cudaDeviceProp deviceProp;
        cudaGetDeviceProperties(&deviceProp, dev);
        if (dev == 0)
        {
            if (deviceProp.major < 1)
            {
```

```

        printf("There is no device supporting CUDA.\n");
    }
    else if (deviceCount == 1)
    {
        printf("There is 1 device supporting CUDA\n");
    }
    else
    {
        printf("There are %d devices supporting CUDA\n",
deviceCount);
    }
}

printf("\nDevice %d: \"%s\"\n", dev, deviceProp.name);
printf("  Major revision number:          %d\n", deviceProp.major);
printf("  Minor revision number:           %d\n", deviceProp.minor);
printf("  Total amount of global memory:      %d bytes\n",
deviceProp.totalGlobalMem);
printf("  Total amount of constant memory:     %d bytes\n",
deviceProp.totalConstMem);
printf("  Total amount of shared memory per block:  %d bytes\n",
deviceProp.sharedMemPerBlock);
printf("  Total number of registers available per block: %d\n",
deviceProp.regsPerBlock);
printf("  Warp size:                            %d\n", deviceProp.warpSize);
printf("  Multiprocessor count:
%d\n",deviceProp.multiProcessorCount );

printf("  Maximum number of threads per block:      %d\n",
deviceProp.maxThreadsPerBlock);

printf("  Maximum sizes of each dimension of a block:  %d x %d x %d\n",
deviceProp.maxThreadsDim[0],deviceProp.maxThreadsDim[1],
deviceProp.maxThreadsDim[2]);

printf("  Maximum sizes of each dimension of a grid:   %d x %d x %d\n",
deviceProp.maxGridSize[0], deviceProp.maxGridSize[1], deviceProp.maxGridSize[2]);

```

```

        printf(" Maximum memory pitch:           %d bytes\n",
deviceProp.memPitch);

        printf(" Texture alignment:             %d bytes\n",
deviceProp.textureAlignment);

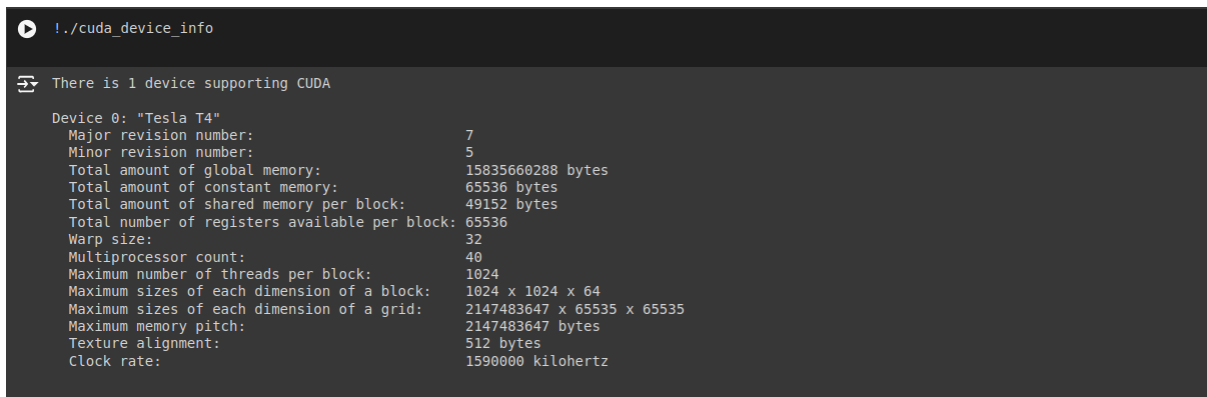
        printf(" Clock rate:                     %d kilohertz\n",
deviceProp.clockRate);

    }

}

```

Output :



```

!./cuda_device_info

There is 1 device supporting CUDA

Device 0: "Tesla T4"
Major revision number:      7
Minor revision number:      5
Total amount of global memory: 15835660288 bytes
Total amount of constant memory: 65536 bytes
Total amount of shared memory per block: 49152 bytes
Total number of registers available per block: 65536
Warp size: 32
Multiprocessor count: 40
Maximum number of threads per block: 1024
Maximum sizes of each dimension of a block: 1024 x 1024 x 64
Maximum sizes of each dimension of a grid: 2147483647 x 65535 x 65535
Maximum memory pitch: 2147483647 bytes
Texture alignment: 512 bytes
Clock rate: 1590000 kilohertz

```

Problem Statement 2:

Write a program to where each thread prints its thread ID along with hello world.
Launch the kernel with one block and multiple threads.

```
# Q2
%%writefile hello_cuda.cu
#include <stdio.h>

// CUDA kernel
__global__ void helloFromThreads() {
    int tid = threadIdx.x;
    printf("Hello World from thread %d\n", tid);
}

int main() {
    int numThreads = 10;
    helloFromThreads<<<1, numThreads>>>();
    cudaDeviceSynchronize();

    return 0;
}
```

Writing hello_cuda.cu

[6] !nvcc hello_cuda.cu -o hello_cuda

[7] !./hello_cuda

Hello World from thread 0
Hello World from thread 1
Hello World from thread 2
Hello World from thread 3
Hello World from thread 4
Hello World from thread 5
Hello World from thread 6
Hello World from thread 7
Hello World from thread 8
Hello World from thread 9

Problem Statement 3:

Write a program to where each thread prints its thread ID along with hello world.
Launch the kernel with multiple blocks and multiple threads.

```
# Q3
%%writefile hello_multi_cuda.cu
#include <stdio.h>

// CUDA kernel
__global__ void helloFromThreads() {
    // Calculate the global thread ID
    int threadID = blockIdx.x * blockDim.x + threadIdx.x;

    // Print Hello World along with the thread ID and block index
    printf("Hello World from block %d, thread %d (Global thread ID: %d)\n", blockIdx.x, threadIdx.x, threadID);
}

int main() {
    // Define the number of threads per block and the number of blocks
    int threadsPerBlock = 4; // Example: 4 threads per block
    int numBlocks = 3;      // Example: 3 blocks

    // Launch the kernel with multiple blocks and threads per block
    helloFromThreads<<<numBlocks, threadsPerBlock>>>();

    // Synchronize to ensure all threads finish before program exits
    cudaDeviceSynchronize();

    return 0;
}
```

```
[9] !nvcc hello_multi_cuda.cu -o hello_multi_cuda
```

```
!./hello_multi_cuda
```

```
⌘ Hello World from block 2, thread 0 (Global thread ID: 8)
Hello World from block 2, thread 1 (Global thread ID: 9)
Hello World from block 2, thread 2 (Global thread ID: 10)
Hello World from block 2, thread 3 (Global thread ID: 11)
Hello World from block 0, thread 0 (Global thread ID: 0)
Hello World from block 0, thread 1 (Global thread ID: 1)
Hello World from block 0, thread 2 (Global thread ID: 2)
Hello World from block 0, thread 3 (Global thread ID: 3)
Hello World from block 1, thread 0 (Global thread ID: 4)
Hello World from block 1, thread 1 (Global thread ID: 5)
Hello World from block 1, thread 2 (Global thread ID: 6)
Hello World from block 1, thread 3 (Global thread ID: 7)
```

Problem Statement 4:

Write a program where each thread prints its thread ID along with hello world.
Launch the kernel with 2D blocks and 2D threads.

```
# Q4
%%writefile hello_2d_cuda.cu
#include <stdio.h>

// CUDA kernel
__global__ void helloFrom2DThreads() {
    // Get the thread ID in the 2D grid
    int threadID_x = threadIdx.x;
    int threadID_y = threadIdx.y;

    int blockID_x = blockIdx.x;
    int blockID_y = blockIdx.y;

    // Print Hello World along with the thread and block IDs
    printf("Hello World from block (%d, %d), thread (%d, %d)\n", blockID_x, blockID_y, threadID_x, threadID_y);
}

int main() {
    // Define the block and grid dimensions
    dim3 threadsPerBlock(4, 4); // 4x4 threads per block (2D thread configuration)
    dim3 numBlocks(2, 2);      // 2x2 blocks (2D block configuration)

    // Launch the kernel with 2D grid and 2D block dimensions
    helloFrom2DThreads<<<numBlocks, threadsPerBlock>>>>();

    // Synchronize to ensure all threads finish before program exits
    cudaDeviceSynchronize();

    return 0;
}
```

```
[12] !nvcc hello_2d_cuda.cu -o hello_2d_cuda
```

```
▶ !./hello_2d_cuda
```

```
⇌ Hello World from block (0, 1), thread (0, 0)  
Hello World from block (0, 1), thread (1, 0)  
Hello World from block (0, 1), thread (2, 0)  
Hello World from block (0, 1), thread (3, 0)  
Hello World from block (0, 1), thread (0, 1)  
Hello World from block (0, 1), thread (1, 1)  
Hello World from block (0, 1), thread (2, 1)  
Hello World from block (0, 1), thread (3, 1)  
Hello World from block (0, 1), thread (0, 2)  
Hello World from block (0, 1), thread (1, 2)  
Hello World from block (0, 1), thread (2, 2)  
Hello World from block (0, 1), thread (3, 2)  
Hello World from block (0, 1), thread (0, 3)  
Hello World from block (0, 1), thread (1, 3)  
Hello World from block (0, 1), thread (2, 3)  
Hello World from block (0, 1), thread (3, 3)  
Hello World from block (0, 0), thread (0, 0)  
Hello World from block (0, 0), thread (1, 0)  
Hello World from block (0, 0), thread (2, 0)  
Hello World from block (0, 0), thread (3, 0)  
Hello World from block (0, 0), thread (0, 1)  
Hello World from block (0, 0), thread (1, 1)  
Hello World from block (0, 0), thread (2, 1)  
Hello World from block (0, 0), thread (3, 1)  
Hello World from block (0, 0), thread (0, 2)  
Hello World from block (0, 0), thread (1, 2)  
Hello World from block (0, 0), thread (2, 2)  
Hello World from block (0, 0), thread (3, 2)  
Hello World from block (0, 0), thread (0, 3)  
Hello World from block (0, 0), thread (1, 3)  
Hello World from block (0, 0), thread (2, 3)  
Hello World from block (0, 0), thread (3, 3)  
Hello World from block (1, 1), thread (0, 0)  
Hello World from block (1, 1), thread (1, 0)  
Hello World from block (1, 1), thread (2, 0)  
Hello World from block (1, 1), thread (3, 0)  
Hello World from block (1, 1), thread (0, 1)
```