**High Performance Computing Lab**

**Assignment No. 11**

**Name : Sourabh Shankar Patil**

**PRN : 21510045**

**Batch : B2**

**Title:** Understanding concepts of CUDA Programming

**Problem Statement 1:**

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

**Code:**

%%writefile cudaProgram1.cu

#include <stdio.h>

#include <stdlib.h>

#include <cuda\_runtime.h> // Include CUDA runtime header

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: %zu bytes\n", deviceProp.totalGlobalMem); // Use %zu for size\_t

printf(" Total amount of constant memory: %zu bytes\n", deviceProp.totalConstMem); // Use %zu for size\_t

printf(" Total amount of shared memory per block: %zu bytes\n", deviceProp.sharedMemPerBlock); // Use %zu for size\_t

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: %zu bytes\n", deviceProp.memPitch); // Use %zu for size\_t

printf(" Texture alignment: %zu bytes\n", deviceProp.textureAlignment); // Use %zu for size\_t

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

}

}

**Ans:**

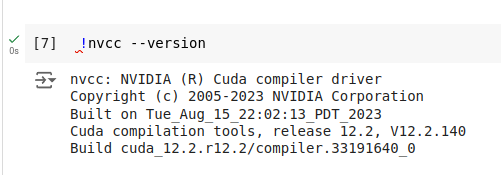
**GPGPU** stands for **"General-Purpose Graphics Processing Unit"**.

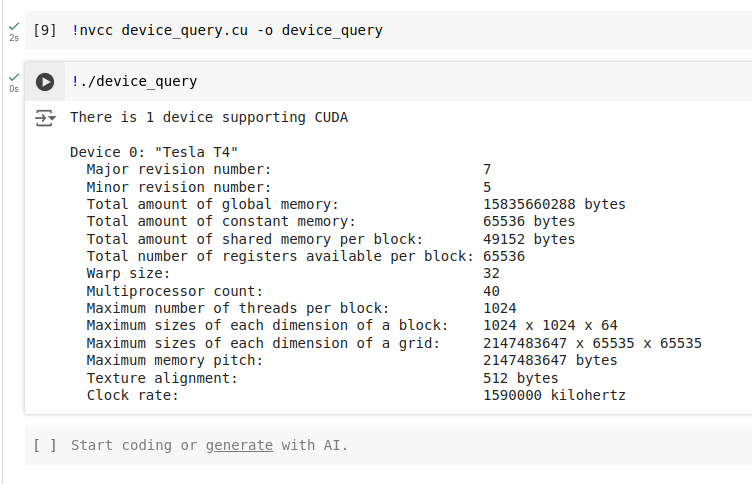
It's a technology that uses a graphics processing unit (GPU) to perform tasks beyond traditional graphics rendering.

GPGPUs are increasingly common and can be used for a variety of purposes, including:

1. Scientific computing
2. Accelerating parts of an application
3. Creating faster, high-performance applications

GPGPUs are efficient parallel processors because of their large number of cores. These cores operate at lower frequencies than CPUs, but are better suited for data that's in graphical form. GPUs can run multiple operations faster than a CPU, making them an ideal choice for many applications.

**Screenshots:**This program queries your system for available CUDA-capable devices using cudaGetDeviceCount() and cudaGetDeviceProperties(). It prints out various properties of each device, such as the number of CUDA cores, memory size, warp size, clock rate, etc.

****

**Problem Statement 2:**

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

**Ans:**

**Code:**

%%writefile hello\_world.cu

#include <stdio.h>

#include <cuda\_runtime.h> // Include CUDA runtime header

\_\_global\_\_ void helloWorld() {

int threadId = threadIdx.x;

printf("Hello World from thread %d\n", threadId);

}

int main() {

helloWorld<<<1, 10>>>();

cudaDeviceSynchronize(); // Ensure that the kernel completes

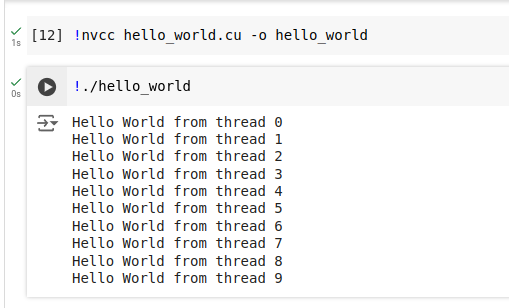
return 0;

}

#### Explanation:

* helloWorld is a CUDA kernel where each thread prints its ID along with "Hello World!".
* The kernel is launched with one block (<<<1, 10>>>) and 10 threads.
* **cudaDeviceSynchronize()** ensures that the CPU waits for the GPU to complete execution before exiting.

**Screenshots:**



**Problem Statement 3:**

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

**Ans:**

**Code:**

%%writefile hello\_world\_blocks.cu

#include <stdio.h>

#include <cuda\_runtime.h> // Include CUDA runtime header

\_\_global\_\_ void helloWorld() {

int threadId = threadIdx.x + blockIdx.x \* blockDim.x;

printf("Hello World from thread %d (Block %d)\n", threadId, blockIdx.x);

}

int main() {

helloWorld<<<5, 10>>>();

cudaDeviceSynchronize();

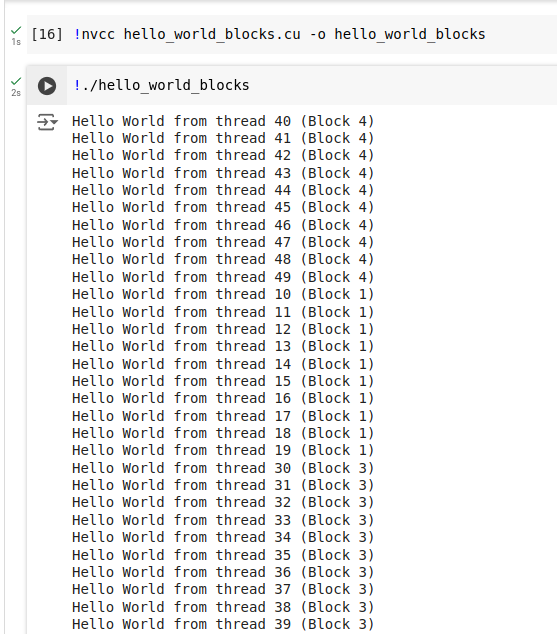
return 0;

}

#### Explanation:

* The kernel is launched with 5 blocks, each having 10 threads (<<<5, 10>>>).
* Each thread prints its unique ID (calculated as threadIdx.x + blockIdx.x \* blockDim.x) and the block it belongs to.

**Screenshots:**

****



**Problem Statement 4:**

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

**Ans:**

**Code:**

%%writefile hello\_world\_2D.cu

#include <stdio.h>

#include <cuda\_runtime.h> // Include CUDA runtime header

\_\_global\_\_ void helloWorld() {

int threadIdX = threadIdx.x + blockIdx.x \* blockDim.x;

int threadIdY = threadIdx.y + blockIdx.y \* blockDim.y;

printf("Hello World from thread (%d, %d)\n", threadIdX, threadIdY);

}

int main() {

dim3 grid(2, 2); // 2D grid (2x2 blocks)

dim3 block(4, 4); // 2D block (4x4 threads)

helloWorld<<<grid, block>>>();

cudaDeviceSynchronize();

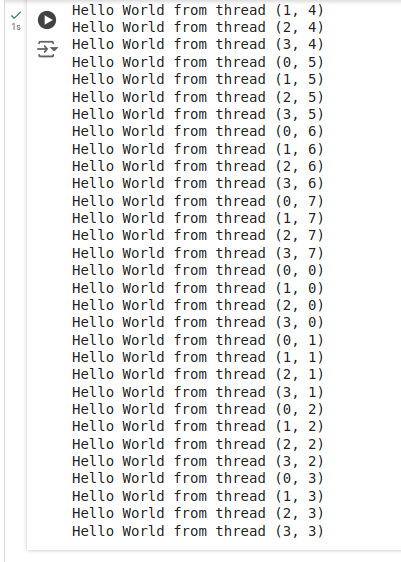
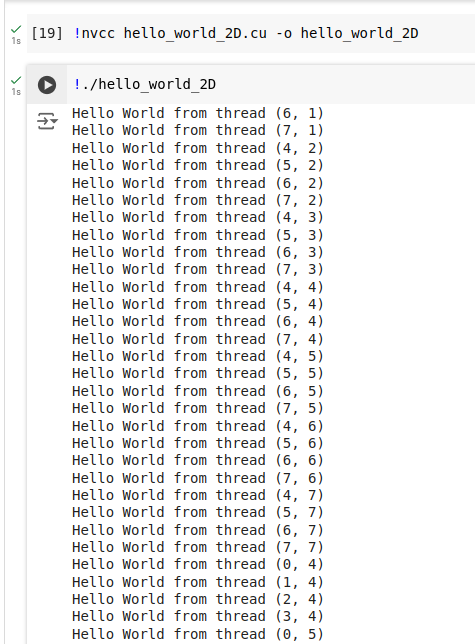
return 0;

}

**Explaination:**

* This kernel uses 2D blocks and 2D threads to assign unique 2D coordinates (threadIdX, threadIdY) for each thread.
* dim3 grid(2, 2) creates a 2x2 grid of blocks, and dim3 block(4, 4) creates 4x4 threads per block.

**Screenshots:**



**Github Link:**

<https://github.com/sourabh-patil-7/HPC-Lab-Assignments/tree/main/Practical%2011>