Final Year B.Tech. (CSE) – VII [2024-25]

6CS452: High Performance Computing Lab

Assignment No: 11

Date: 14/10/2024

Understanding concepts of CUDA Programming

PRN: 2020BTECS00010 **Name: Abhijeet Kamalekar**

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:

```
[7]  !nvcc --version

nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2023 NVIDIA Corporation
Built on Tue_Aug_15_22:02:13_PDT_2023
Cuda compilation tools, release 12.2, V12.2.140
Build cuda_12.2.r12.2/compiler.33191640_0
```

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.

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

!./device_query

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

[ ]  Start coding or generate with AI.
```

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:

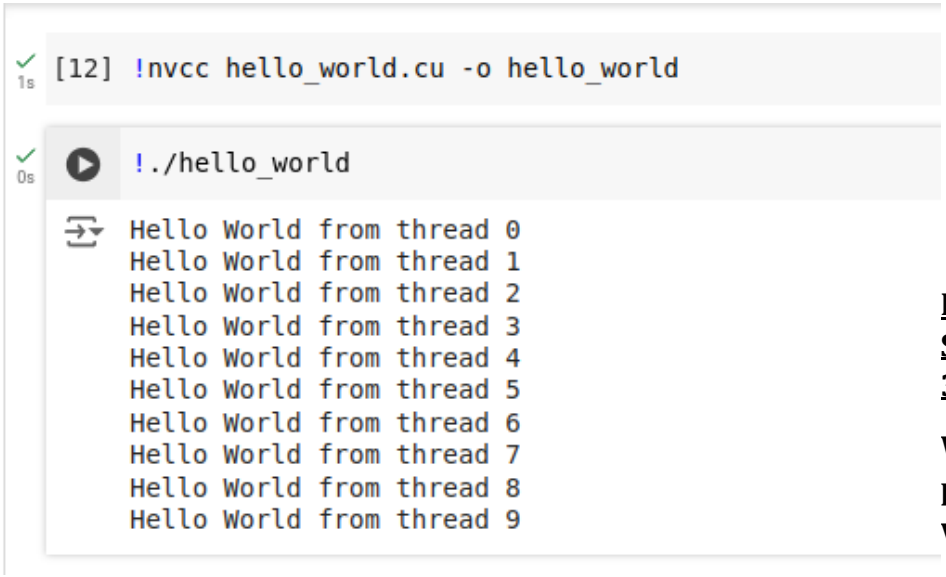
Code:

```
%%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:

```
[16] !nvcc hello_world_blocks.cu -o hello_world_blocks
```

```
!./hello_world_blocks
```

```
Hello World from thread 40 (Block 4)
Hello World from thread 41 (Block 4)
Hello World from thread 42 (Block 4)
Hello World from thread 43 (Block 4)
Hello World from thread 44 (Block 4)
Hello World from thread 45 (Block 4)
Hello World from thread 46 (Block 4)
Hello World from thread 47 (Block 4)
Hello World from thread 48 (Block 4)
Hello World from thread 49 (Block 4)
Hello World from thread 10 (Block 1)
Hello World from thread 11 (Block 1)
Hello World from thread 12 (Block 1)
Hello World from thread 13 (Block 1)
Hello World from thread 14 (Block 1)
Hello World from thread 15 (Block 1)
Hello World from thread 16 (Block 1)
Hello World from thread 17 (Block 1)
Hello World from thread 18 (Block 1)
Hello World from thread 19 (Block 1)
Hello World from thread 30 (Block 3)
Hello World from thread 31 (Block 3)
Hello World from thread 32 (Block 3)
Hello World from thread 33 (Block 3)
Hello World from thread 34 (Block 3)
Hello World from thread 35 (Block 3)
Hello World from thread 36 (Block 3)
Hello World from thread 37 (Block 3)
Hello World from thread 38 (Block 3)
Hello World from thread 39 (Block 3)
```

```
Hello World from thread 38 (Block 3)
Hello World from thread 39 (Block 3)
Hello World from thread 20 (Block 2)
Hello World from thread 21 (Block 2)
Hello World from thread 22 (Block 2)
Hello World from thread 23 (Block 2)
Hello World from thread 24 (Block 2)
Hello World from thread 25 (Block 2)
Hello World from thread 26 (Block 2)
Hello World from thread 27 (Block 2)
Hello World from thread 28 (Block 2)
Hello World from thread 29 (Block 2)
Hello World from thread 0 (Block 0)
Hello World from thread 1 (Block 0)
Hello World from thread 2 (Block 0)
Hello World from thread 3 (Block 0)
Hello World from thread 4 (Block 0)
Hello World from thread 5 (Block 0)
Hello World from thread 6 (Block 0)
Hello World from thread 7 (Block 0)
Hello World from thread 8 (Block 0)
Hello World from thread 9 (Block 0)
```

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:

```
[19] !nvcc hello_world_2D.cu -o hello_world_2D
```

```
!./hello_world_2D
```

```
Hello World from thread (6, 1)
Hello World from thread (7, 1)
Hello World from thread (4, 2)
Hello World from thread (5, 2)
Hello World from thread (6, 2)
Hello World from thread (7, 2)
Hello World from thread (4, 3)
Hello World from thread (5, 3)
Hello World from thread (6, 3)
Hello World from thread (7, 3)
Hello World from thread (4, 4)
Hello World from thread (5, 4)
Hello World from thread (6, 4)
Hello World from thread (7, 4)
Hello World from thread (4, 5)
Hello World from thread (5, 5)
Hello World from thread (6, 5)
Hello World from thread (7, 5)
Hello World from thread (4, 6)
Hello World from thread (5, 6)
Hello World from thread (6, 6)
Hello World from thread (7, 6)
Hello World from thread (4, 7)
Hello World from thread (5, 7)
Hello World from thread (6, 7)
Hello World from thread (7, 7)
Hello World from thread (0, 4)
Hello World from thread (1, 4)
Hello World from thread (2, 4)
```

```
Hello World from thread (1, 4)
Hello World from thread (2, 4)
Hello World from thread (3, 4)
Hello World from thread (0, 5)
Hello World from thread (1, 5)
Hello World from thread (2, 5)
Hello World from thread (3, 5)
Hello World from thread (0, 6)
Hello World from thread (1, 6)
Hello World from thread (2, 6)
Hello World from thread (3, 6)
Hello World from thread (0, 7)
Hello World from thread (1, 7)
Hello World from thread (2, 7)
Hello World from thread (3, 7)
Hello World from thread (0, 0)
Hello World from thread (1, 0)
Hello World from thread (2, 0)
Hello World from thread (3, 0)
Hello World from thread (0, 1)
Hello World from thread (1, 1)
Hello World from thread (2, 1)
Hello World from thread (3, 1)
Hello World from thread (0, 2)
Hello World from thread (1, 2)
Hello World from thread (2, 2)
Hello World from thread (3, 2)
Hello World from thread (0, 3)
Hello World from thread (1, 3)
Hello World from thread (2, 3)
Hello World from thread (3, 3)
```