National University of Computer and Emerging Sciences

Islamabad Campus

High Performance Computing with GPUs (CS4110)

Course Instructor(s):

Dr. Imran Ashraf

Section(s): A, B, C

Sessional-I Exam

Total Time (Hrs): 1

Total Marks: 50

Total Questions: 4

Date: Sep 24, 2025

_____ _____ _____

Roll No **Course Section** **Student Signature**

Do not write below this line.

Attempt all the questions.

A bonus mark will be awarded for solving all the questions and all parts in order.

CLO 1: Describe the terms related to HPC systems, GPU architecture and GPU programming models. **[Understand, C2]**

Q1: **[3+6+3+3 = 15 marks]**

Solve the following to demonstrate your **understanding** of CUDA kernel launch configuration.

A) A vector addition kernel is launched with <<<16, 64>>>. How many threads are generated?

B) Suppose you need to process an array of size 4096 elements. Write two different kernel launch configurations (blocks, threads) that can cover all elements.

C) A 2D grid is launched with dim3 grid(4, 8); dim3 block(16, 16);. How many total threads are generated?

D) You want to launch 1 million threads. If the maximum block size is 1024 threads, how many blocks do you need?

Solution

A) <<<16, 64>>> → 16×64=1024

B) Array of size 4096 elements

We need at least 4096 threads. Possible launch configurations:

<<<16, 256>>> → 16×256=4096

<<<64, 64>>> → 64×64=4096

C) 2D grid and block

Grid = dim3(4, 8) → total blocks = 4×8=32

Block = dim3(16, 16) → threads per block = 16×16=256

Total threads = 32×256=8192

D) Launching 1,000,000 threads, max 1024/block

Blocks = 1,000,000 / 1024

1,000,000÷1024≈976.56

Round up → 977 blocks

CLO 3: Develop data-parallel solutions using appropriate programming model.**[Create, C6]**

Q2: **[5+3+2+2+3 = 15 marks]**

You are asked to **develop** a CUDA-accelerated image processing filter. The goal is to apply a grayscale-to-negative transformation on a given image.

A) Write a CUDA kernel that takes as input an array representing the image pixels in grayscale (values from 0–255). Each thread should process one pixel and transform it using the rule:

$$\text{negative}(x)=255-x$$

B) Assume the image size is 4096 × 4096 pixels. Propose a **suitable** grid and block configuration for launching your kernel. Justify your choice.

C) Modify your kernel to handle the case when the number of pixels is not exactly divisible by the total number of threads launched.

D) Provide the code to allocate device memory.

E) Provide the code which will handle data transfers between host and device.

Solution

A)

```
__global__ void negativeFilter(unsigned char *d_in, unsigned char
*d_out, int N) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < N) {
        d_out[idx] = 255 - d_in[idx];
    }
}
```

B) Grid and Block Configuration

- Image size = 4096 × 4096 = 16,777,216 pixels
- Choose threads per block = 256 (warp-friendly)
- Blocks = (N + 255) / 256 = 65,536
- Launch:

```
negativeFilter<<<65536, 256>>>(d_in, d_out, N);
```

ensures all pixels are covered.

C) Kernel Handling Non-Divisible Case

Already handled with boundary check:
```
if (idx < N) { ... }
```

This implies, threads beyond N do nothing.

D) Device Memory Allocation

```
unsigned char *d_in, *d_out;
int N = 4096 * 4096;
size_t size = N * sizeof(unsigned char);

cudaMalloc((void**)&d_in, size);
cudaMalloc((void**)&d_out, size);
```

E) Data Transfer (Host ↔ Device)

```
// Copy input from host to device
cudaMemcpy(d_in, h_in, size, cudaMemcpyHostToDevice);

// Copy result back from device to host
cudaMemcpy(h_out, d_out, size, cudaMemcpyDeviceToHost);
```

CLO 4: Analyze performance of an application running on an HPC system to improve compute and memory performance. **[Evaluate, C5]**

Q3: **[3+3+2+3 = 11 marks]**

Analyze the peak performance of a CPU and GPU with the following specifications. A CPU has:

- 8 cores,
- each core can execute 8 single-precision FLOPs per cycle,
- base clock frequency = 3.0 GHz.

A GPU has:

- 64 Streaming Multiprocessors (SMs),
- each SM contains 128 CUDA cores,
- each CUDA core can execute 2 single-precision FLOPs per cycle,
- base clock frequency = 1.5 GHz.

A) Calculate the theoretical peak performance (in GFLOPs) of the CPU.

B) Calculate the theoretical peak performance (in TFLOPs) of the GPU.

C) Based on these results list two reasons why GPUs generally achieve higher FLOP rates than CPUs, despite lower clock speeds.

D) Despite the higher GPU FLOPS, is it possible for an application to perform better on CPU? List at least 3 reasons to support your answer.

Solution

A) **CPU theoretical peak (GFLOPs)**

- Cores = 8
- FLOPs/core/cycle = 8 (SP)
- Clock = 3.0 GHz = 3.0×10^9
- FLOPs/sec = $8\times8\times3.0\times10^9$
- Total SP FLOPS = 192 GLOPS

B) **GPU theoretical peak (TFLOPs)**

- SMs = 64, cores/SM = 128 → total CUDA cores = 64×128
- FLOPs/core/cycle = 2 (SP)
- Clock = 1.5 GHz = 1.5×10^9
- FLOPs/sec = $8192\times2\times1.5\times10^9$
- Total SP FLOPS = 24.576 TFLOPS

C) **Two reasons GPUs achieve higher FLOP rates than CPUs (despite lower clocks)**

1. **Much wider parallel hardware:** thousands of simple arithmetic units (many more cores) designed for throughput, so aggregate FLOPs per cycle are far larger.
2. **Architectural emphasis on throughput:** GPU pipelines, SIMD/SIMT execution and very high parallelism (many cores + many threads) trade single-thread latency for massive simultaneous FLOP execution.

D) **Can a CPU beat the GPU despite higher GPU FLOPs? Why?**

Yes, due to

- Memory-bound workloads
- Small problem sizes, which typically results in more overheads
- Control-heavy or divergent code

CLO 1: Describe the terms related to HPC systems, GPU architecture and GPU programming models. **[Understand, C2]**

Q4: **[3+3+3 = 9 marks]**

Contrast with examples

A) Theoretical Performance vs Measure Performance

B) Grid vs Block

C) Device vs Host memory

Solution

Theoretical Performance	**Measured Performance**
Theoretical performance: The maximum possible performance based on hardware specs (e.g., FLOPs × frequency).	Measured performance: The actual performance achieved in practice, which is usually lower due to memory bottlenecks, branching, or inefficiencies.
Example: A GPU may have a theoretical peak of 24 TFLOPs, based on core count and clock.	Example: The same GPU might only achieve 10 TFLOPs in a real matrix multiplication benchmark.
Grid	**Block**
A collection of blocks that make up the full launch configuration.	A group of threads (1D, 2D, or 3D) that execute together and can share fast shared memory.
gridDim(64,64) with each block (16×16) → processes a 1024×1024 image.	blockDim(16,16) creates 256 threads per block.
Host Memory	**Device**
Memory located in the CPU's address space (RAM). It is accessed by CPU cores.	Memory located on the GPU (global memory). Threads on the GPU access it during kernel execution.
Example: An image array stored in malloc on CPU.	Example: Using cudaMalloc to store the same image on GPU before processing.