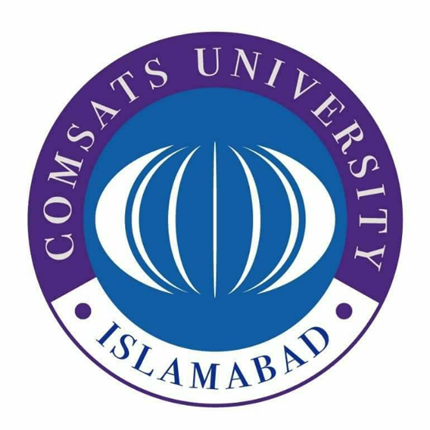
**COMSATS UNIVERSITY ISLAMABAD**

**LAHORE CAMPUS**

**ASSIGNMENT NO#2**

**NAME: ZOHA NISAR GILL**

**ROLL NUMBER: SP23-BCS-130**

**SECTION: C**

**Q1**

**Conceptual Question**

**Why block size not a multiple of 32 wastes resources?**

* On NVIDIA GPUs, **1 warp = 32 threads**.
* If you choose block sizes that are not multiples of 32 (e.g., 33, 50, etc.), the GPU still allocates **whole warps**.
* Example: 33 threads → requires **2 warps (64 threads)**, but only 33 are active and **31 are idle**.
* Result: **underutilization** because some warp slots do no work.

**How occupancy of an SM depends on block size and threads per block?**

* **Occupancy = (active warps per SM) / (maximum warps per SM)**.
* Each block launches a certain number of warps.
* Too small a block → not enough warps to fill the SM.
* Too large a block → fewer blocks can fit on the SM, which may reduce total active warps.
* Best performance usually comes when the block size is a multiple of 32 and big enough to keep SMs busy but not so big that it reduces parallelism.

**Q2**

**Practical / Coding Question**

**Which configuration runs fastest and why?**

When the block size is set to **(8,8) = 64 threads**, there are too few threads per block, which leads to underutilization of the GPU since the hardware is not kept fully busy. On the other hand, with **(32,32) = 1024 threads**, the block is very large and consumes more registers and shared memory per block, which limits how many blocks can run simultaneously on an SM and reduces occupancy. The **(16,16) = 256 threads** configuration provides the best balance between parallelism and resource usage, keeping enough threads active without overloading hardware resources, which makes it the fastest overall.

**Q3**

**Analysis Question**

**If Case B is fastest, explain why neither the smallest nor the largest block size gave optimal performance.**

Case B, with 256 threads per block, is the fastest configuration. The smallest block size, 64 threads, does not perform as well because there are too few threads to fully utilize the GPU, leaving many cores idle. The largest block size, 1024 threads, also underperforms since it uses too many registers and shared memory per block, which limits how many blocks can run concurrently on each Streaming Multiprocessor and reduces occupancy. Case B strikes the optimal balance, providing enough threads to keep the GPU busy while not exceeding resource limits, resulting in the best overall performance.

**Q4**

**Discussion Question**

**Why does increasing the number of threads per block not always improve performance? Consider register pressure, shared memory limits, and scheduling**

### Why Increasing Threads per Block Does Not Always Improve Performance

**1. Register Pressure**

* Each thread uses registers.
* Larger blocks consume more registers per block, which can limit the number of blocks that fit on a Streaming Multiprocessor (SM), lowering occupancy.

**2. Shared Memory Limits**

* Blocks may also require shared memory.
* If one block uses too much shared memory, fewer blocks can run simultaneously on an SM, reducing parallelism.

**3. Scheduling Constraints**

* An SM can only run a fixed number of threads and blocks at once.
* If a block already consumes the maximum resources, other blocks cannot be scheduled, causing GPU underutilization.