

LAb Assignment 3

Name: Urwah Muhammad

Roll NO.: SP23-BCS-123

Section: SP23-BCS-C

Subject: Parallel and Distributed Computing

Submitted to: Sir Akhzar Nazir



**Answers**

# 3. Streams

So, if you run the two kernels on different streams and they both try to save their results to the same array, it's a total mess. It’s a race condition. It's like two people trying to write an answer on the same spot on a whiteboard at the exact same time—the final result is just gibberish.

To fix this, you can either just tell them to write to different arrays (which is the easy way), or you have to make them get in line by putting them back on the same stream so one has to finish before the other starts.

# 4. Synchronization

Using cudaDeviceSynchronize() is like telling your computer's brain (the CPU) to stop and wait for the graphics card (GPU) to actually finish all the math. When the GPU is done, the CPU gets the final, correct answers.

If you don't use it, the CPU is way too impatient. It asks the GPU to do the work and then immediately asks for the answer without waiting. The GPU is still busy, so the CPU just grabs whatever random junk is in the memory at that moment and shows you garbage numbers.

# 5. Thread Hierarchy

The <<<1, N>>> way (one big block): This is like having one huge team with N workers. To find a worker, you just need their ID number (threadIdx.x). The team number (blockIdx.x) is always just 0 because there's only one team.

The <<<N/32, 32>>> way (lots of small blocks): This is the normal way. It's like having a bunch of smaller teams (blocks) with 32 workers each. To find a specific worker, you need to know their team number (blockIdx.x) AND their ID number within that team (threadIdx.x). You have to combine them with the formula idx = blockIdx.x \* 32 + threadIdx.x to find out which piece of the array they're supposed to work on.