WIDS Week1 Assignment

Anant Singh

15/12/25

Chapter 1

Task 1: Identify and Analyze a GPU-Accelerable Workload

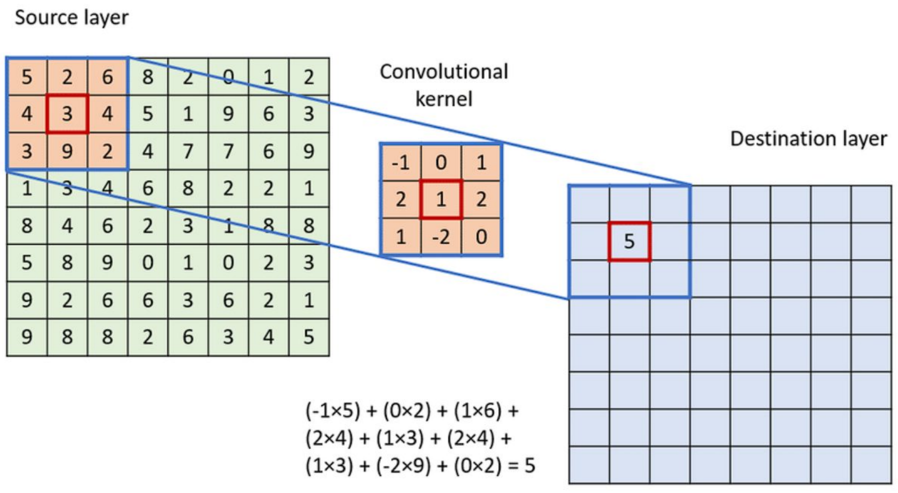
1.1 Operational Breakdown

Work load chosen - Convolution in images. Will extend to CNNs later

In convolution of images, what essentially happens is that we try to extract features from an image (like shadows, corners, curves, etc.) by establishing correlation between pixels in an area of the image.

We multiply an area of the image by a "kernel" of the same size. This operation takes a dot product between the two matrices and calculates the value of one pixel in the destination layer.

The values of a kernel depend on the feature we are trying to detect. Here, the computation is analysed using a fixed kernel.



1.1.1 Mathematics and Pseudocode

```
float K[k][k];
float image[N][N];

for x in 0 : N-k        // Boundary conditions
                         // are handled using zero-padding or clamping
  for y in 0 : N-k
    sum = 0
    for i in 0 : k
      for j in 0 : k
        sum += image[x+i][y+j] * K[i][j]
    D[x][y] = sum
```

1.1.2 Data Shapes

- Image: width × height (e.g., 1024 × 1024)
- Destination layer: width × height (this may be different from the original image)
- Kernel: depends on the application (most common sizes are 3×3, 5×5, etc.)

1.1.3 Parallelism Opportunities

Making the destination layer involves convolving continuous chunks of kernel-sized areas in the image with the kernel. This can be done in parallel; that is, each value in the destination layer can be computed independently and in parallel.

1.2 Compute vs Memory Behaviour

1.2.1 Data Reuse Opportunity

Every pixel in the image participates in multiple kernel operations (mostly k^2 times, unless it is near the boundary). Hence, pixels can be stored in shared memory, where each pixel value can be reused by multiple threads.

1.2.2 Compute-Bound vs Memory-Bound

Since each input pixel is used many times, without caching and appropriate memory usage, the same pixel has to be fetched repeatedly from global memory. This becomes the major contributor to execution time.

Additionally, since the mathematics involved is basic arithmetic (multiplication and addition), the computational load is relatively low. Hence, the workload is memory-bound rather than compute-bound.

1.2.3 Dependencies

There are no dependencies between the calculations of individual values in the destination layer. Therefore, they can be assigned to completely independent parallel threads.

1.3 Expected GPU Mapping

1.3.1 Thread, Block, and Grid Mapping in GPU

The image should be divided into tiles, and each tile should be assigned to a block. All pixels in a tile are assigned individual threads within that block.

1.3.2 Scaling Behaviour

Scaling this workload by increasing image resolution or dimensions does not affect the parallelism strategy. For example, when working with an RGB image, the same process can be applied independently to each channel (R, G, B), as each channel behaves like a separate image. Hence, the approach scales efficiently.

1.3.3 Potential Issues

Limited L2 Cache and Shared Memory Size

Due to the limited size of shared memory per block, large tiles and kernels may face difficulties in being processed completely in parallel.

Similarly, a small L2 cache size results in more frequent accesses to global memory (DRAM), which introduces additional latency.

Boundary Pixels

To handle boundary pixels, images are often padded with zeros at the edges. As a result, boundary pixel computations include extra steps compared to regular pixels.

This difference can hinder parallel processing, as some threads execute additional operations, causing warp divergence (when different threads in a warp follow different execution paths).

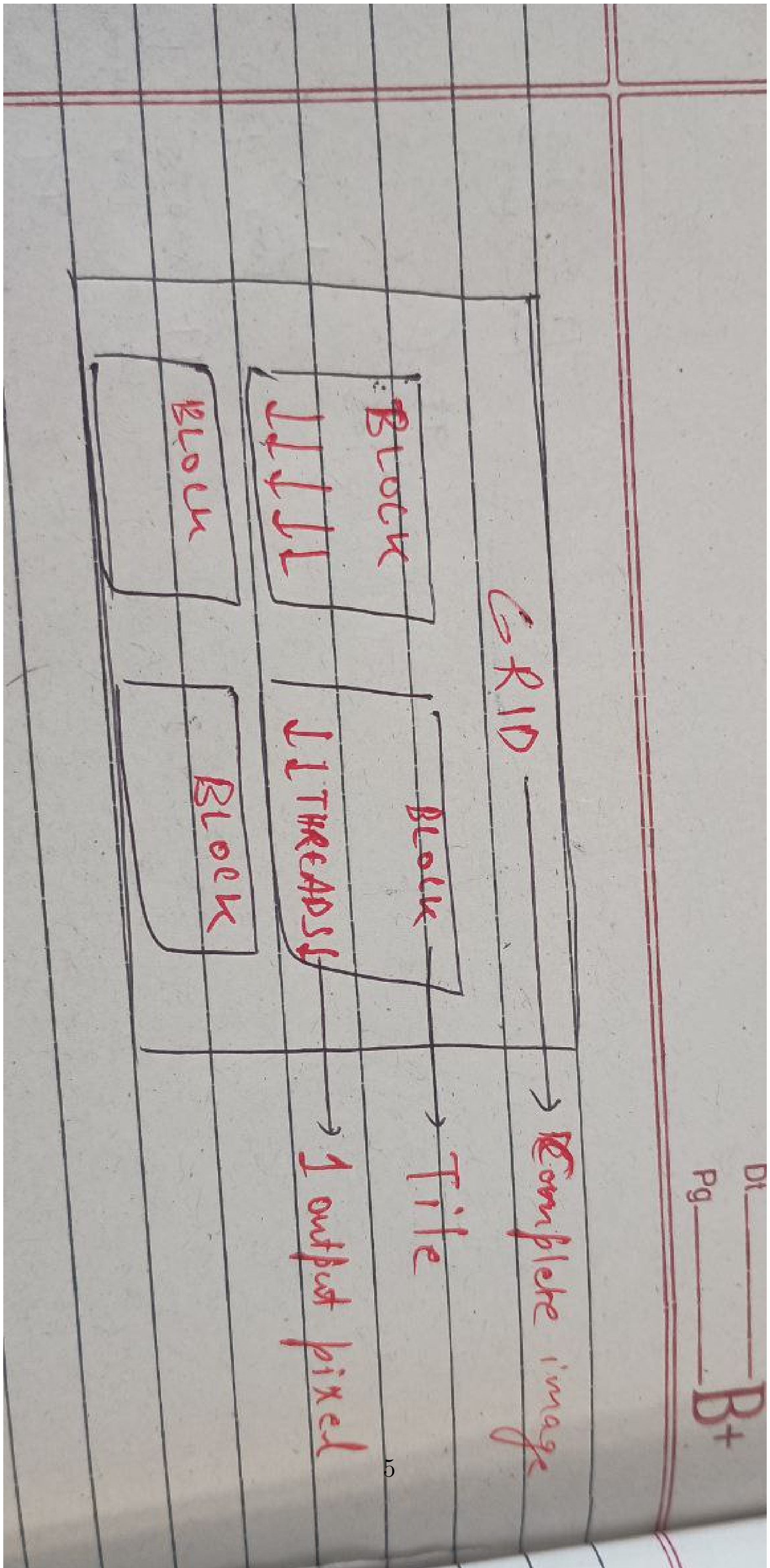
One possible solution is to pad the image on the CPU and then send the already padded image to the GPU.

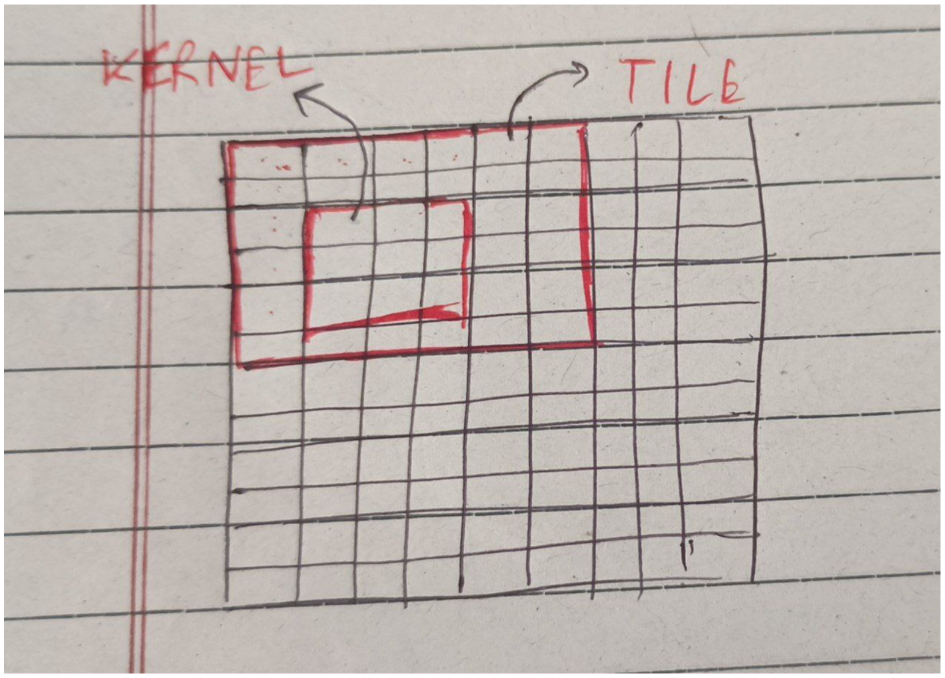
Chapter 2

Task 2: CUDA Execution Model Diagram

The division of the workload on the GPU is as follows:

- Threads: Each pixel or value of the destination layer has a corresponding thread.
- Blocks: Each image tile is assigned one block.
- Grid: The grid covers the entire destination layer.


GRID
Block
Block
Block
Block
THREADS
Complete image
Tile
1 output pixel



2.1 Synchronization Requirement

Synchronization is required within a block when threads cooperatively load data into shared memory before computation.
For example there is a pixel I[x][y] which has to be used by 2 threads. Lets say thread 1 uses the values and calls for the pixel values from the next tile. But thread 2 has not read the data and copies the value from the next tile instead. Hence, here synchronization is vital so that all threads always see the same tile in the shared memory.
No synchronization is required between blocks.

2.2 Memory Bottlenecks

The primary memory bottlenecks arise from repeated global memory accesses, as discussed in the Potential Issues section.

2.3 Conclusion

In subsequent stages after convolution in a convolutional neural network, such as max pooling and fully connected neural networks, the GPU continues to play a crucial and important role.