CS 181AI
Lecture 13

Matrix Multiplication on GPU: Behind the Scenes

Arthi Padmanabhan

Mar 1, 2023

Feedback: Course Webpage -> Resources

- Please assess how helpful each of the following are to your learning:
  - Paper reading
  - Lectures
  - In-class demos
  - Assignments
  - Office hours
- What can I do differently to make this course a better learning experience?
- Particular things I'd like feedback on:
  - Paper reading – do you think you're developing critical academic paper reading skills? What can be changed so that you better develop these skills?
  - Demos in class – are you able to follow along and is the pace reasonable? Is there anything I can do during demos to make sure you're getting the most out of it?

Last Time

- Review of ML operations
- Matrix multiplication in 3 ways:
  - Python loop
  - Numpy
  - GPU

Important Concepts from Last Time

- True or False: If we try to run an operation but half our data is in GPU memory and other half is in CPU memory, CUDA will take care of moving everything to GPU
- True or False: Some operations that could be run in parallel are still faster when run on the CPU

Today

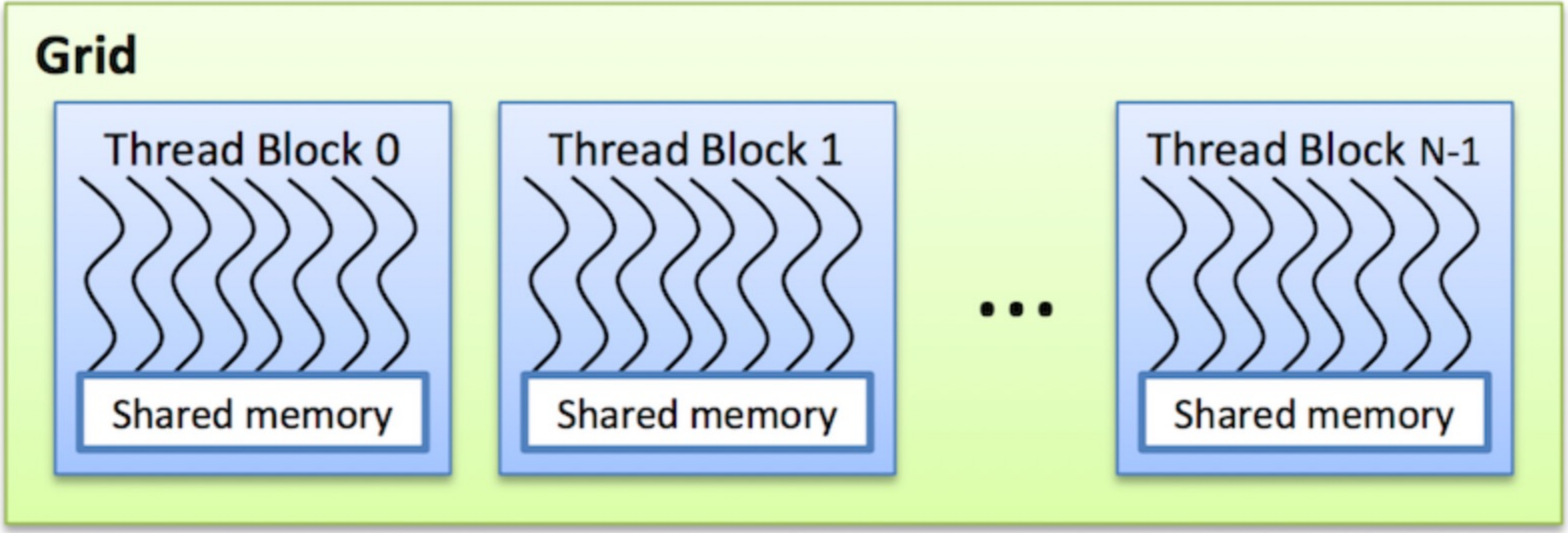
- Thread organization in GPU
- How matrix multiplication is parallelized
- With the above concepts, we can understand how when we issue a matrix multiplication to run on GPU, it is divided into threads

Kernel

- Function that is meant to be executed in parallel on the GPU

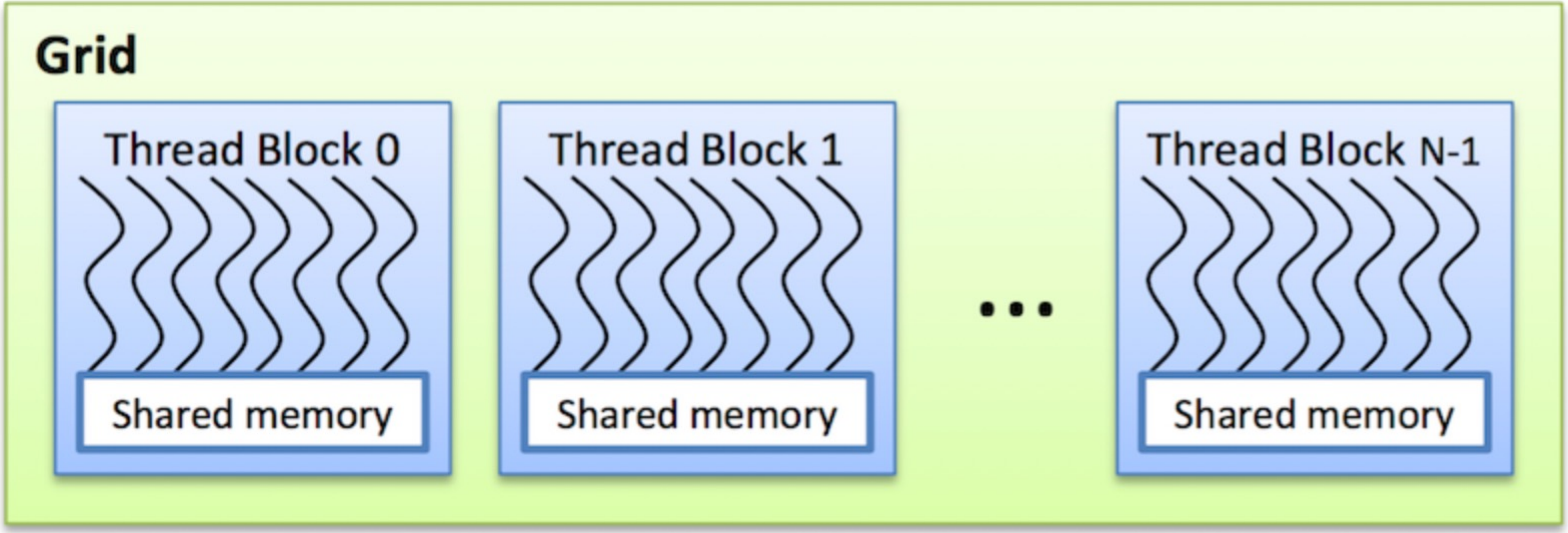
Thread Arrangement

- Threads are arranged as a grid of thread blocks
- Different kernels can have different threads per block and blocks per grid



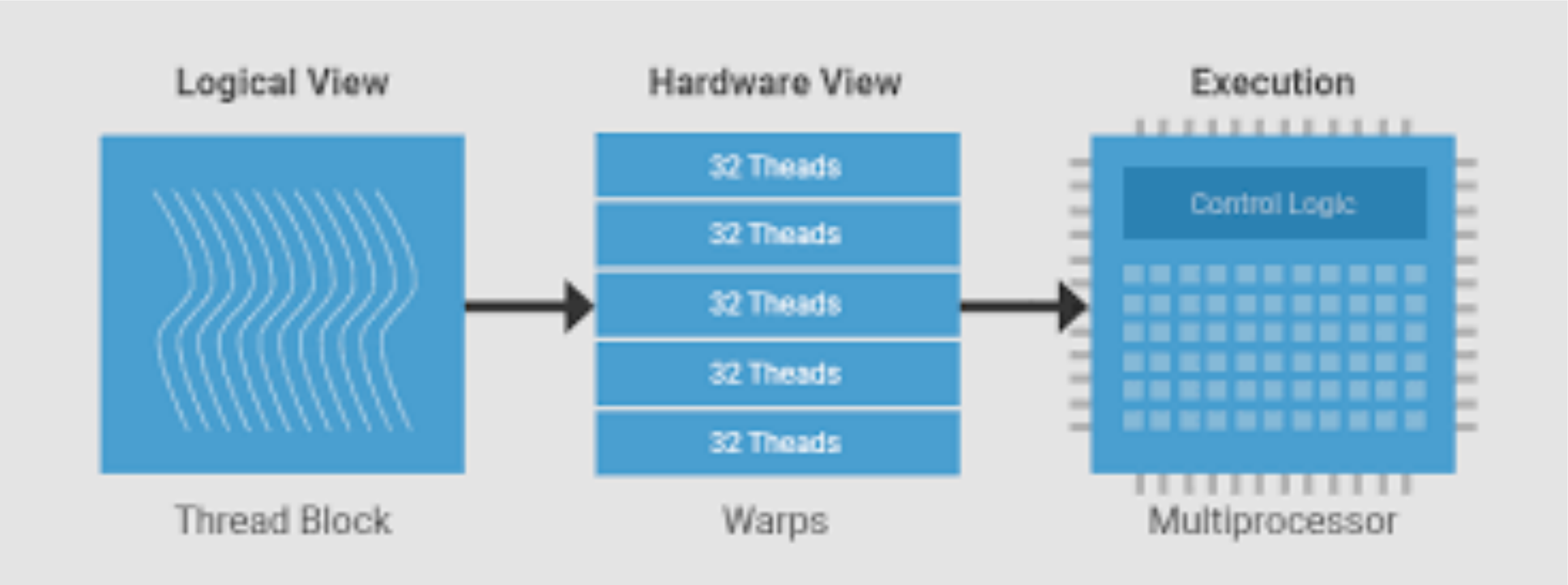
Thread Arrangement

- One block per core, or SM -> all threads in a block have access to shared memory



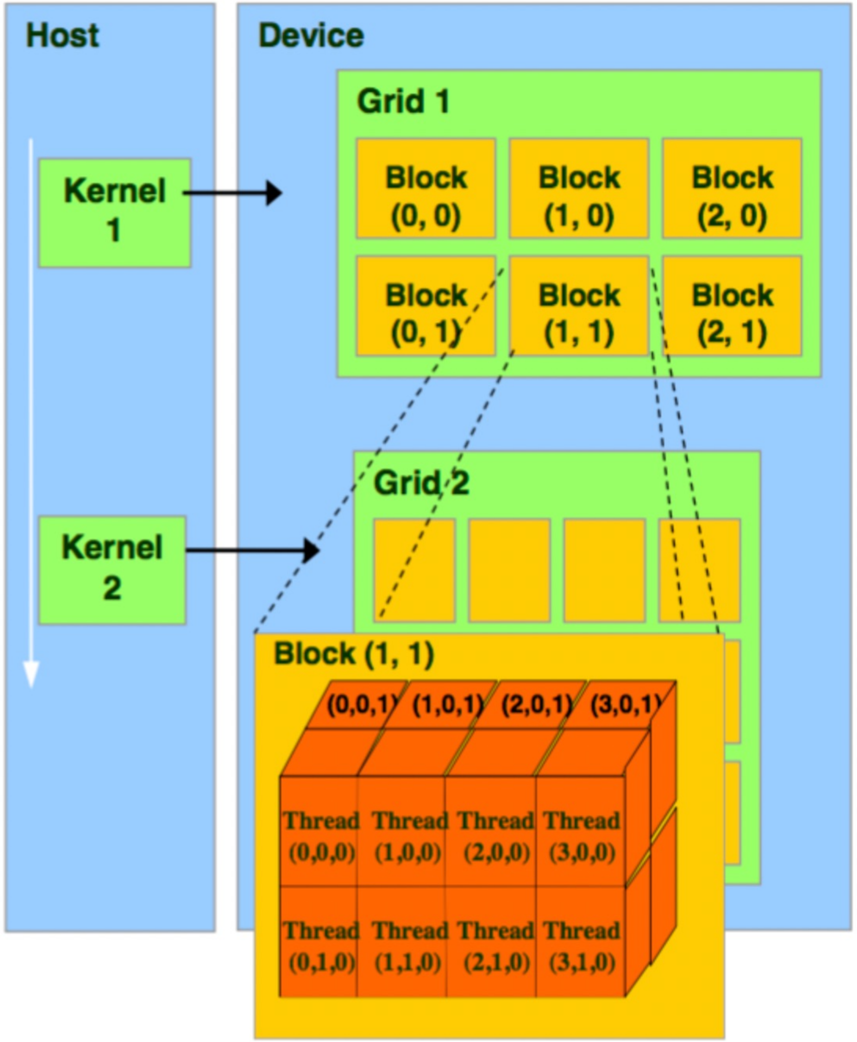
Warps

- Warp = 32 threads; basic unit of execution
- We want the number of threads per block to be a multiple of 32 so that we don't waste threads (threads can only be allocated in warps)

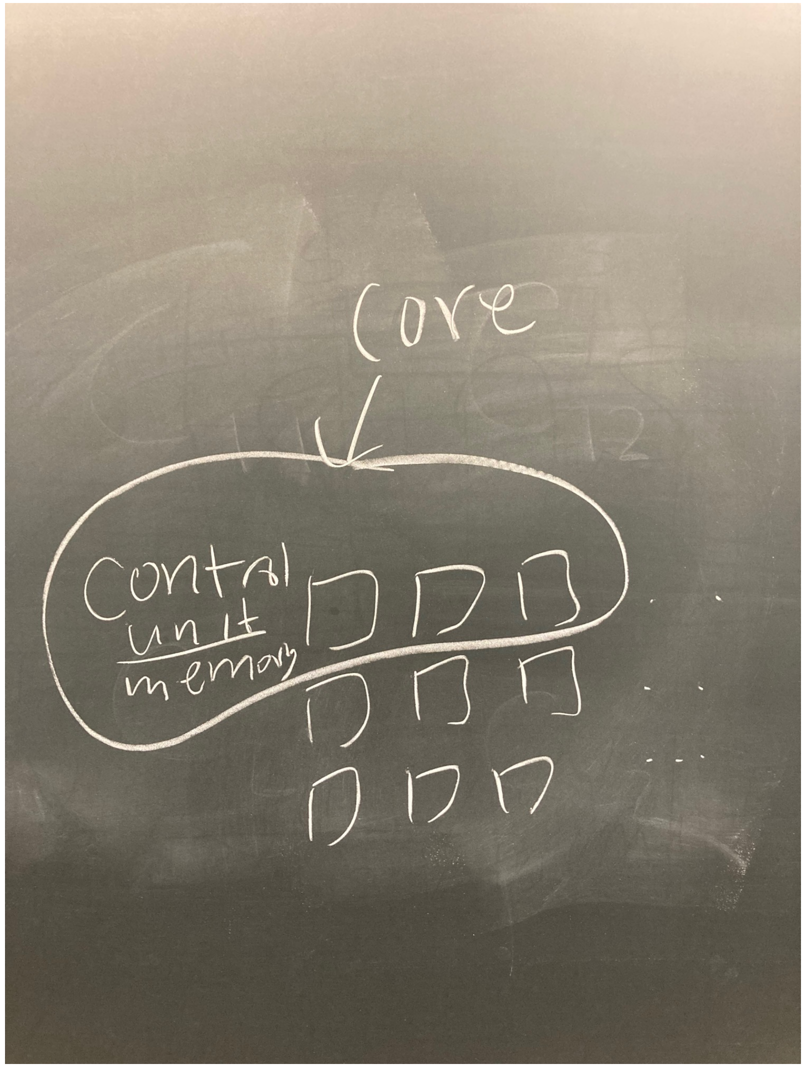


Structural Organization

- The blocks within a grid and the threads within a block can be arranged in a 1D, 2D, or 3D way
- Each thread can access its thread and block index and can use this to determine which piece of data to run on

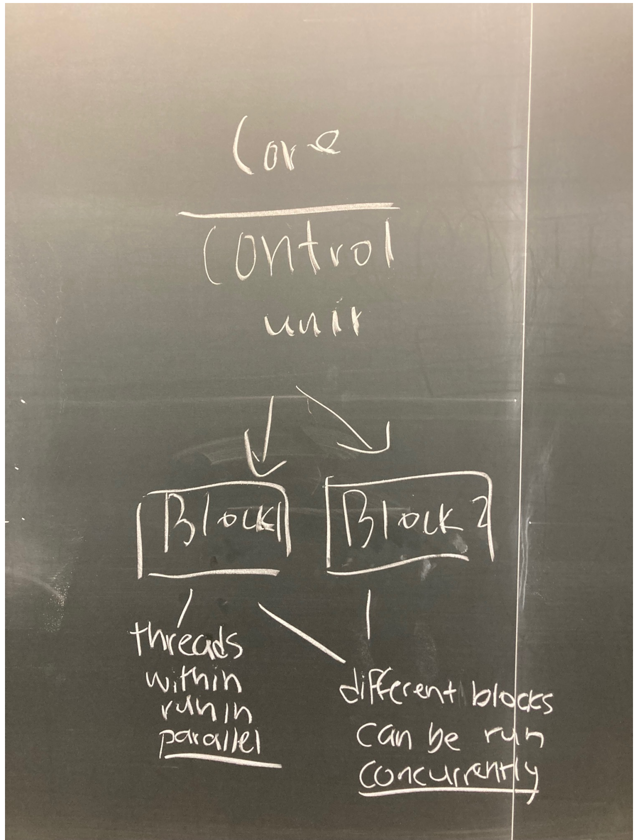


One core per SM



Parallelism vs Concurrency

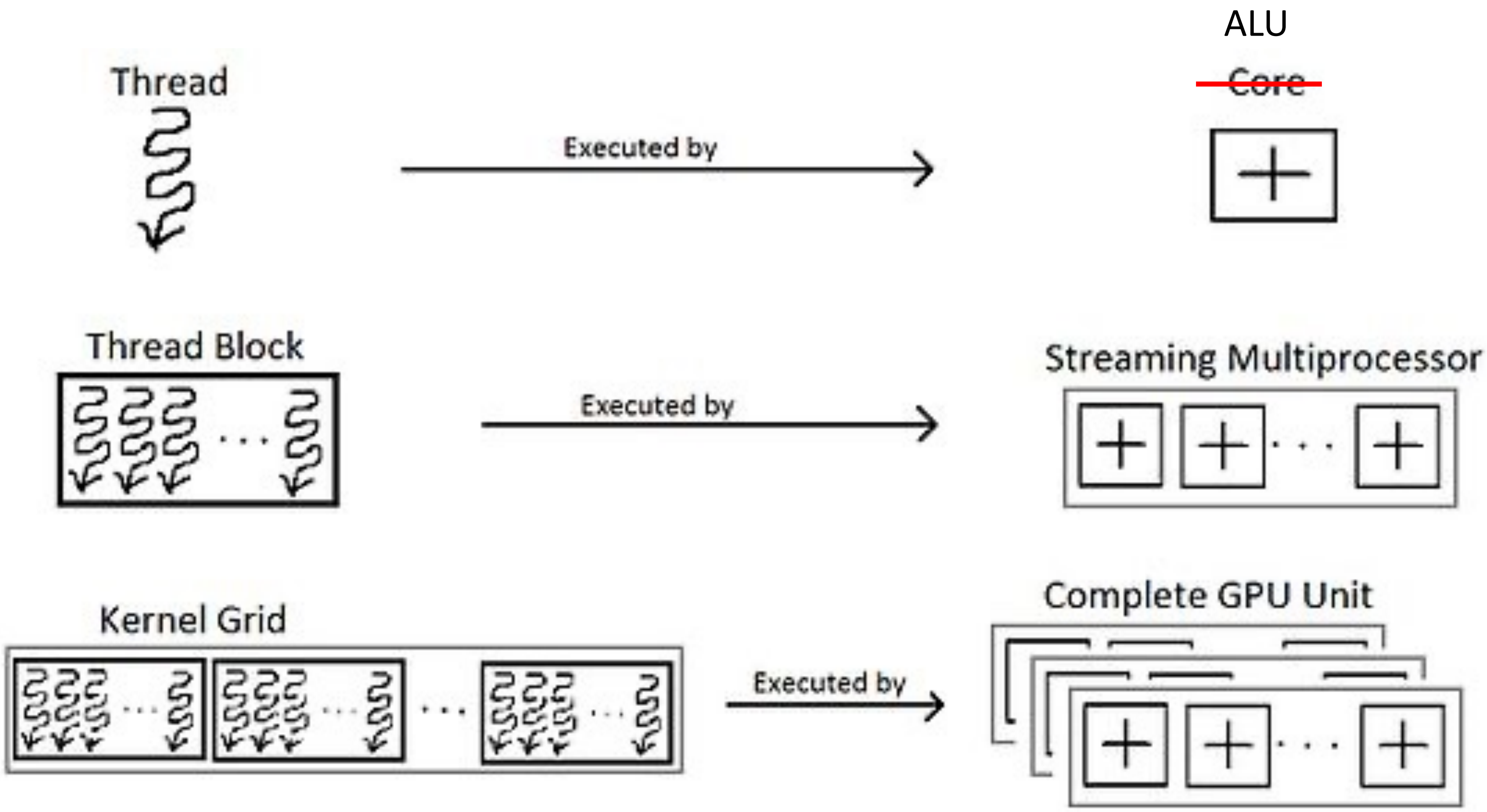
- Threads within a block run in parallel
- Different thread blocks can run concurrently on one SM



Kernel Launch

- When a kernel is launched, the following occur:
  - Blocks of the grid are enumerated and distributed to cores (SMs)
  - Threads within a thread block execute in parallel on different parts of the data (using different ALUs)
  - Different thread blocks can run concurrently on one core (they will pause at the same time)

Full Picture



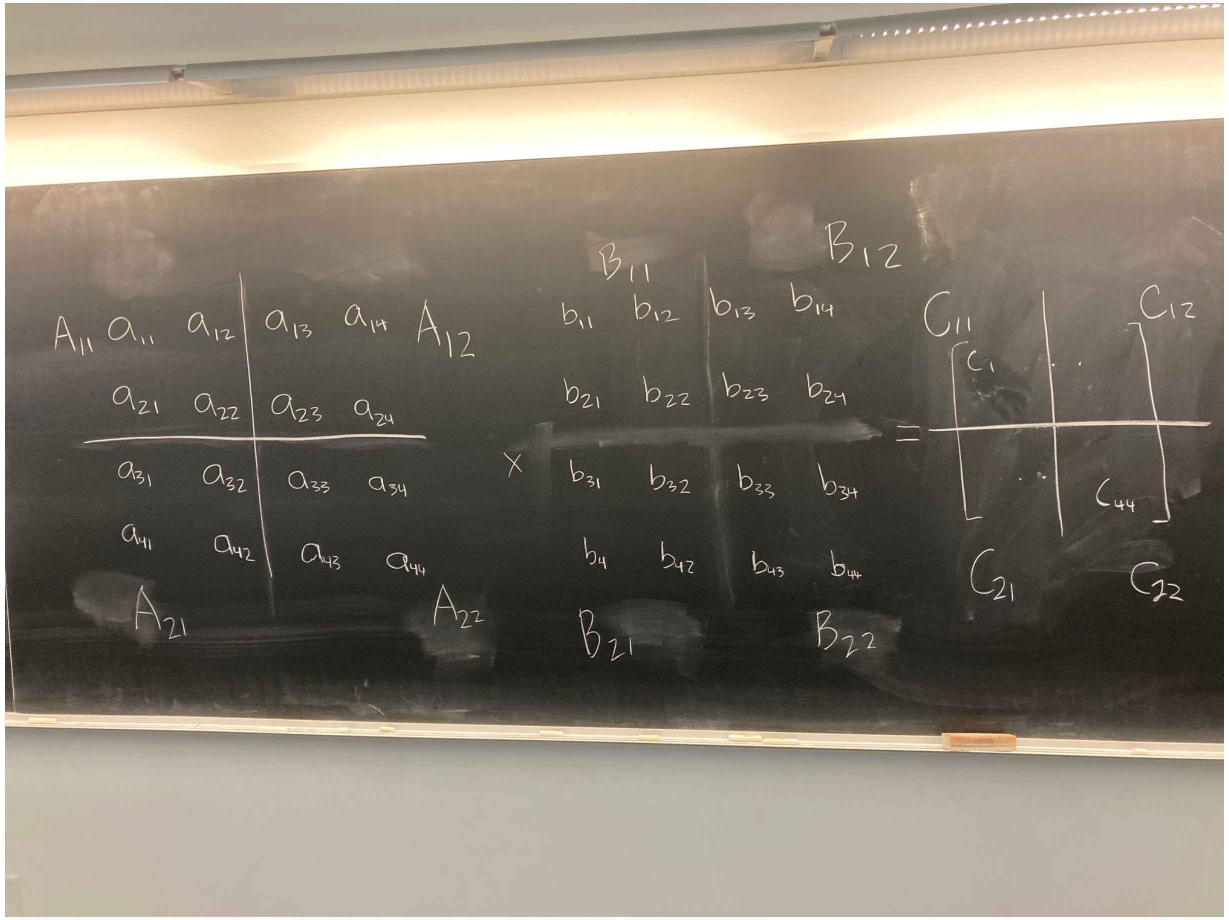
Configurations?

- How is the threads/block and blocks/grid decided?
- Depends on both your computation and the device limitations (amount of shared memory, limits on active thread, etc)
- CUDA does this for you!

Simplest Solution

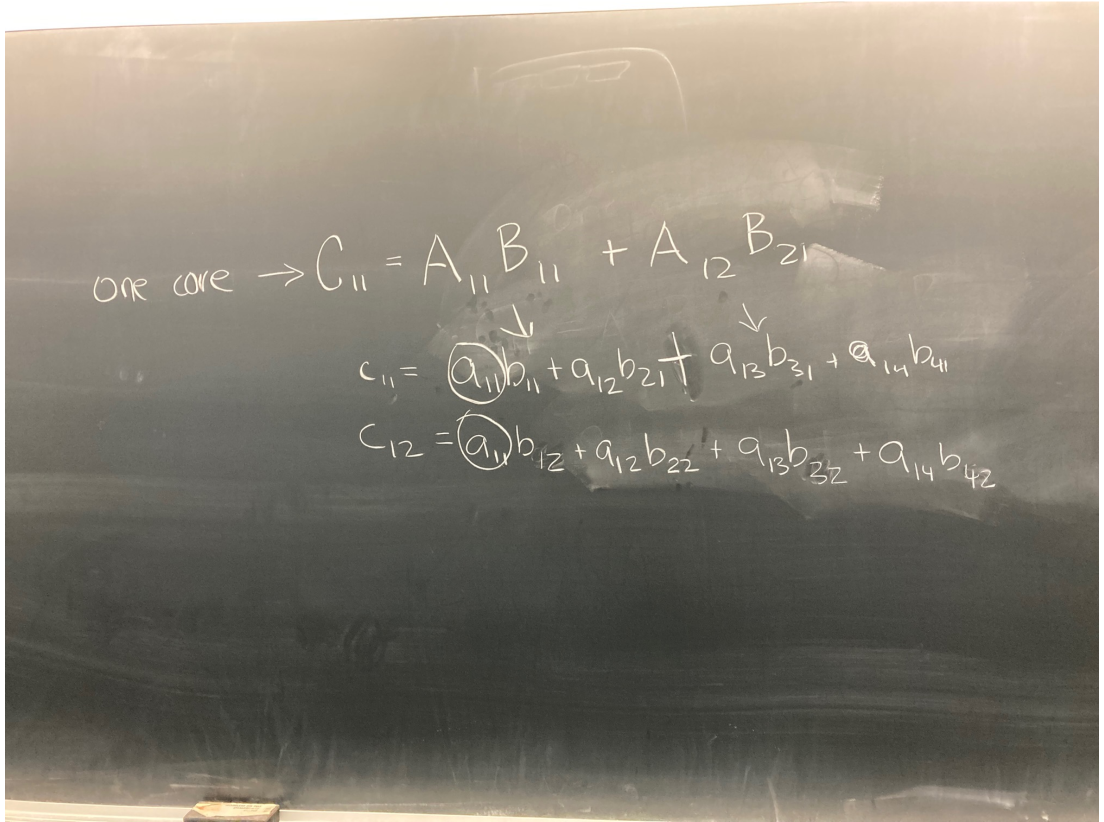
- Assign one SM per element in output
  - Need to load one row of A and one column of B per SM

Block Multiplication



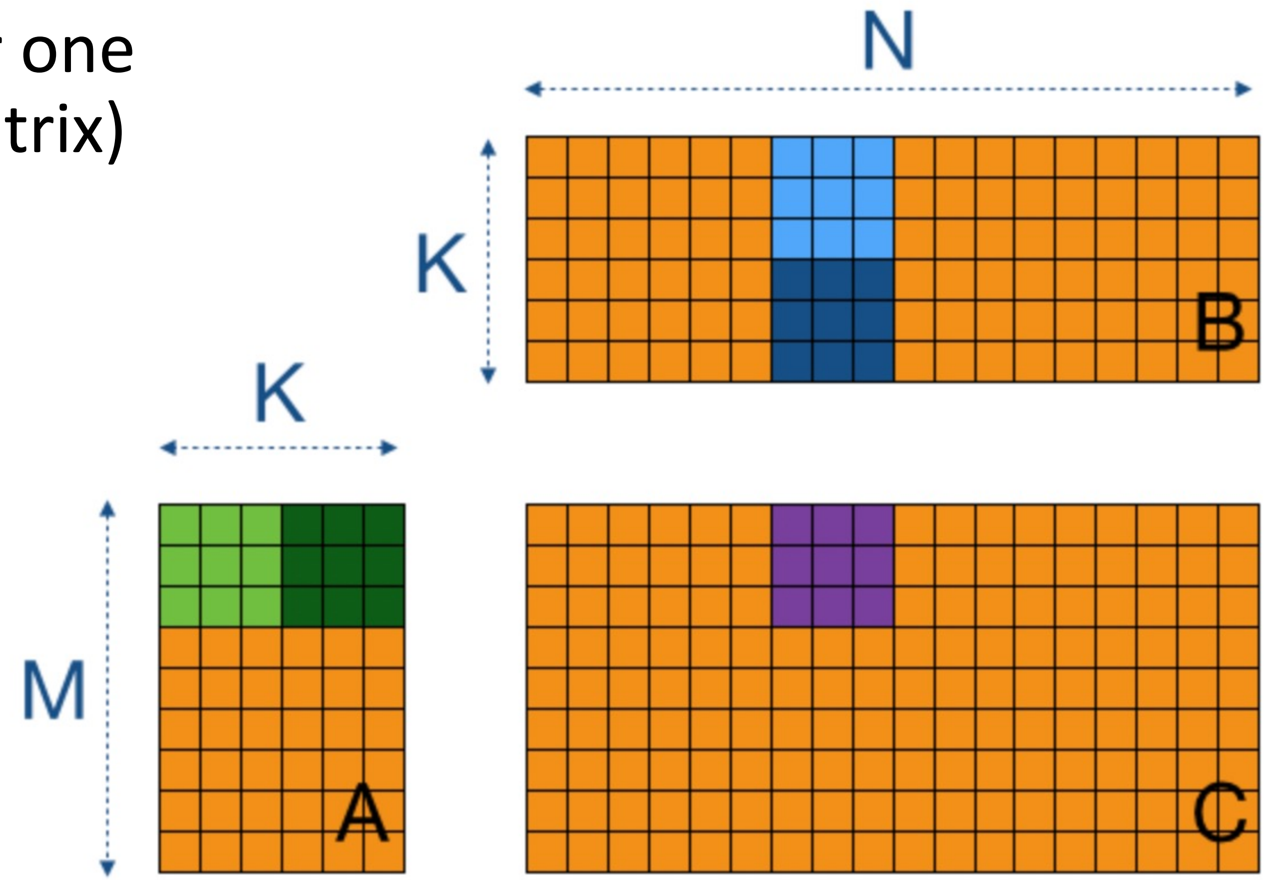
Block Multiplication

- One core would be responsible for C_{11}
- Each element fetched from memory is used twice instead of once



Block Multiplication

- Each core is responsible for one block in C (the resulting matrix)



Next Time

- Look at memory and energy usage of ML models
- Next Wednesday:
  - Go over ideas for final project