CS 250: Foundations of Computer Systems
[Graphics Processing Units GPUs]

"We have trained the A.I. art generator so well that it now feels too insecure about its work to make any art."
Sarah Kempa, January 11, New Yorker.

Shrideep Pallickara
Computer Science
Colorado State University

1

Frequently asked questions from the previous class survey

- Are B-Tree keys always ints?
  - No! They can be any data type … but they need to be linearly ordered
- How do you find your block size?
  - Depends on the OS. Some have GUIs (e.g., Windows Systems Information Components > Storage > Disks); some have CLIs blockdev on Linux
- If the block size is 4KB why have a fan-out of only 512?

2

Topics covered in this lecture

- GPUs
  - History
  - Contrasting with CPUs
  - Differences in caching schemes

The broader journey today …

How did hardware built to draw triangles become the engine behind games, scientific computing, cryptocurrency, and modern AI?

But I've read this script and the costume fits, so I'll play my part.
Cleopatra, The Lumineers.

GPUs

The early days of GPU

- Graphics processing units (GPUs) are present in most PCs
- GPUs provide several basic operations to the CPU, such as rendering an image in memory and then displaying that image onto the screen
- A GPU will typically process a complex set of polygons (a map of the scene) to be rendered
  - Applies **textures** to the polygons
  - Performs **shading** and **lighting** calculations

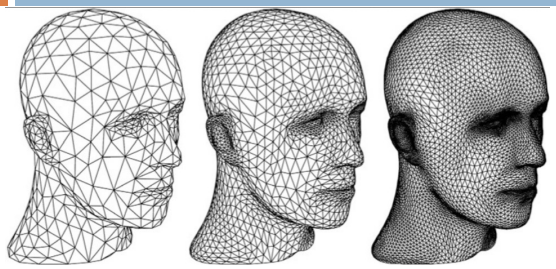
The path to GPGPU programming

- One of the important steps was the development of **programmable shaders**
  - Effectively little programs that the GPU ran to calculate different effects
  - Rendering was no longer fixed in the GPU; through downloadable shaders, it could be manipulated
- This was the first evolution of general-purpose graphics processor unit (GPGPU) programming
  - The design had taken its first steps in moving away from *fixed-function* units

The trick: one spell, millions of targets

- Graphics workloads are naturally **parallel**
  - Apply similar operations to many vertices, triangles, fragments, and pixels
- Each tiny operation is simple; the number of operations is enormous
- This is the pattern GPUs were built for:
  - Same program
  - Many data items
  - Limited coordination
- Once GPUs became programmable, people realized:
  - "Maybe the triangles were just the opening act"

Any smooth surface can be convincingly approximated by **triangles**



- The example above is three successive triangulations of a head
- The more triangles we use and the smaller we make them, the better the approximation becomes

Avatar (2009)

- Level of polygonal detail became more extravagant
- At director James Cameron's insistence, animators used about a **million polygons** to render *each* plant on the imaginary world of Pandora
- Given that the movie took place in a lush virtual jungle, that amounted to a lot of plants . . . and a lot of polygons
- No wonder Avatar cost $300 million to produce
- It was the first movie to use polygons by the billions!

And finally … gaming

- In 2022, Top Gun Maverick was released
  - First blockbuster in a while for Hollywood
  - Made $1 Billion in about a month
- Call of Duty (COD) Black Ops 6
  - $1 Billion in 10 days!

CPU

When I look into your eyes
I can see a love restrained
But darlin' when I hold you
Don't you know I feel the same?

Nothin' lasts forever
And we both know hearts can change
And it's hard to hold a candle
In the cold November rain

November Rain, Guns N' Roses

GPU

CPUs vs GPUs

The biggest player in the GPU market

- NVIDIA
  - Most folks still mispronounce its name calling it "en-vidia"
    - It's actually "nuh-vidia"
      - Similar to the wor-chest-er-shire (wrong!) versus WOOS-ter-sher (correct!) issue
- Founded in 1993
  - Made ~$98 Billion in profit for 2025
    - Profit margins are around 70%

GPUs (and NVIDIA) eating the CPU's lunch

- As of 2024, NVIDIA controls about 95% of the market for specialist AI chips
  - Also accounts for 80% of the gaming GPUs
- GPUs have found wider use beyond gaming and AI
  - Cryptocurrency and self driving cars
- Two other strengths that NVIDIA has
  - CUDA
  - High-performance networking (via purchase of Mellanox for $7bn in 2019)

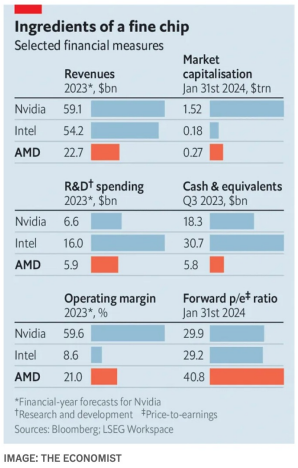


IMAGE: THE ECONOMIST

Looking at CPU & GPU servers in the computer science department

- 2 CPU server (AMD EPYC 74F3)
  - Number of cores/CPU: 24
  - Clock speed: 3.2 GHz, Turbo: 4 GHz
  - Size of the RAM: 1TB
  - Approximate cost: $6,000
- GPU Server with 4 GPUs (NVIDIA A100); specs per GPU listed below
  - Number of CUDA cores (or streaming processors): 6912
  - Clock speed: 1.410 GHz (boost)
  - Size of RAM (GPU memory): 80GB
  - Approximate cost: $56,000

CPUs vs GPUs

- Traditional CPUs are aimed at **serial code execution** and are *extremely good at it*
  - They contain special hardware such as **branch prediction** units, **multiple caches**, etc., all of which target serial code execution
- GPUs are **not designed for this serial execution flow**
  - Only achieve their peak performance when fully utilized in a parallel manner

GPUs are not magic, which is rude … but true

- **GPUs struggle when** programs have:
  - Lots of serial dependencies
  - Heavy branching where threads take different paths
  - Small workloads with little parallelism
  - Frequent communication among far-apart threads
  - Too much data movement between CPU and GPU
- The question is not "*Can I run this on a GPU?*" The question is "*Is there enough parallel work to keep the beast fed?*"

Effect of large cache sizes in the CPU

- As cache sizes grow, so does the physical size of the silicon used to make the processors
- The **larger** the chip, the *more expensive* it is to manufacture
  - And the higher the likelihood that it will contain an error and be discarded during the manufacturing process
- Sometimes these faulty devices are sold cheaply as either triple- or dual-core devices, with the faulty cores disabled
- However, the effect of larger, progressively more inefficient caches ultimately results in higher costs to the end user

GPUs working in tandem with the CPU

- A GPU card, currently, must operate **in conjunction** with a CPU-based host
- The GPU cards can broadly be considered as an **accelerator** or a **coprocessor**

The GPU is fast; but the trip to the GPU is not free

- The CPU usually controls the overall program
- Data often must move from CPU memory to GPU memory
- The CPU launches a GPU kernel
- The GPU performs massively parallel work
- Results may need to move back to the CPU
- GPU speedups are best when the computation is **large enough** to justify the trip

2007 was the year that GPU programming went mainstream

- NVIDIA brought GPUs into the mainstream by adding an easier-to-use programming interface
  - **CUDA**, or Compute Unified Device Architecture
- Opened up the possibility to program GPUs
  - Without having to learn complex shader languages
  - Without thinking only in terms of graphics primitives

CUDA in brief [1/2]

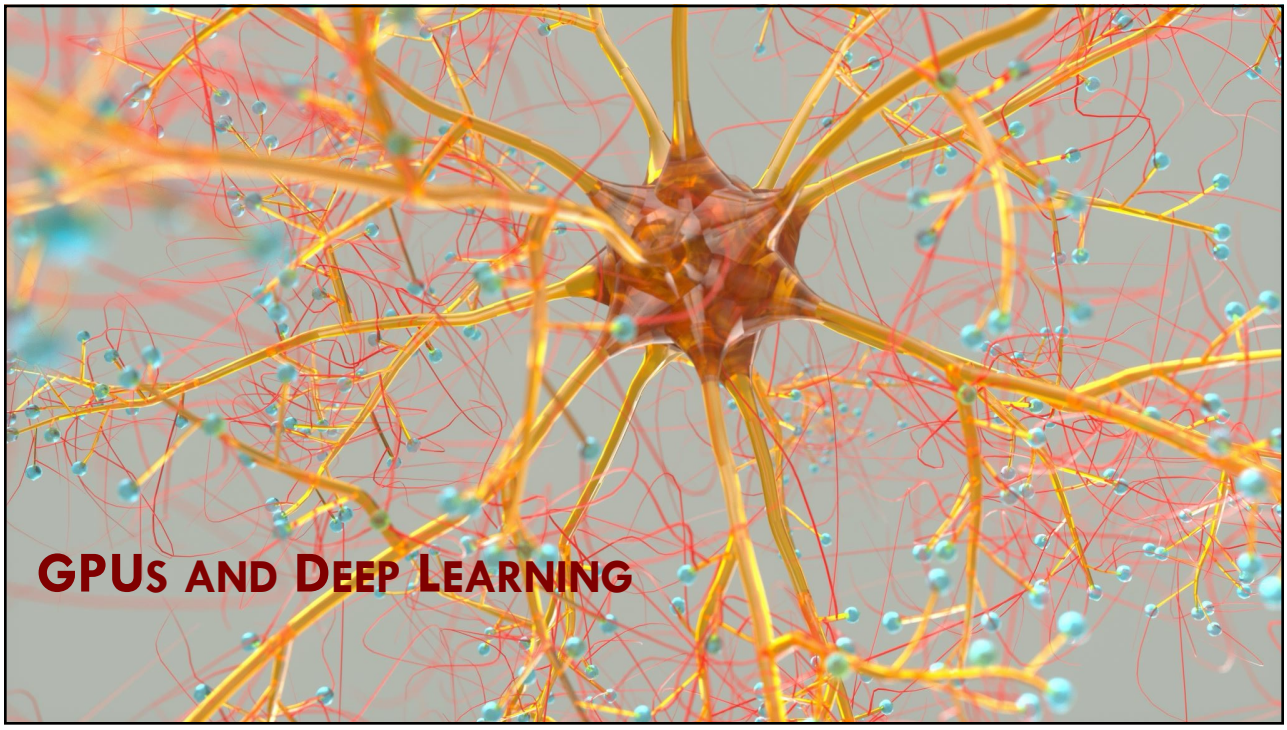
- CUDA is an **extension to the C language** that allows GPU code to be written in regular C
- The code is either targeted for
  - The **host** processor (the CPU) or
  - At the **device** processor (the GPU)

CUDA in brief [2/2]

- The host processor spawns multithread tasks (or kernels as they are known in CUDA) onto the GPU device
  - **CUDA kernel** is a *function* that gets executed on the GPU
- The GPU has its **own internal scheduler** that will then allocate the kernels to whatever GPU hardware is present

CUDA and compatibility

- The CUDA compilation model applies the same principle as used in Java: **runtime compilation of a virtual instruction set**
- Allows modern GPUs to execute code from even the oldest generation GPUs
- However, executions benefit from the original programmer reworking the program for the features of the newer GPUs



25

GPUs and Deep learning

- From a computational perspective, a major breakthrough for deep learning occurred in the late 2000s with the adoption of graphics processing units (GPUs) by the deep learning community to speed up training
- A neural network can be understood as a **sequence of matrix multiplications** that are interspersed with the application of *nonlinear activation functions*
  - GPUs are optimized for *very fast* matrix multiplications
- So, GPUs are ideal hardware to speed up neural network training, and their use has made a significant contribution to the development of the field

26

Some history

- In 2004, Oh and Jung reported a twentyfold performance increase using a GPU implementation of a neural network
- The following year two further papers were published that demonstrated the potential of GPUs to speed up the training of neural networks:
  - Steinkraus et al. (2005) used GPUs to train a two-layer neural network
  - Chellapilla et al. (2006) used GPUs to train a CNN (convolutional neural network)

A note about these early efforts

- Circa 2004-2006, there were significant programming challenges to using GPUs for training networks
  - The training algorithm had to be implemented as a sequence of graphics operations
  - So, the initial adoption of GPUs by neural network researchers was relatively slow

These programming challenges were significantly reduced in 2007 with the release of **CUDA**

- CUDA was specifically designed to facilitate the use of GPUs for general computing tasks
- In the years following the release of CUDA, the use of GPUs to speed up neural network training became standard

A very, very brief history of what powered deep learning

- Improved weight initialization methods
- New activation functions
- The speedup in computing power
- The massive increase in dataset sizes

GETTING BACK TO GPUS

The GPU hardware consists of a number of key blocks [1/2]

- Streaming multiprocessors (SMs)
- Streaming processors (SPs)
- Memory (global, constant, shared)

The GPU hardware consists of a number of key blocks [2/2]

- Each GPU device contains a set of SMs
  - Each of which contain a set of SPs or CUDA cores
- The SPs execute work as **parallel sets** of up to 32 units
- CPUs need a lot of the complex circuitry to achieve high-speed serial execution through **instruction-level parallelism** (ILP)
  - GPUs eliminate this
  - GPUs replace ILP with a *programmer-specified* explicit parallelism model
    - Allowing more compute capacity to be squeezed onto the same area of silicon

ILP: Cooking a meal without watching the water boil

- Imagine a chef preparing one full-course meal from a single recipe
- Some steps depend on earlier ones:
  - You cannot sauté onions before chopping them, and you cannot plate the pasta before boiling it
  - But other steps are independent:
    - You can preheat the oven, boil water, chop herbs, and mix a sauce without waiting for each one to finish

ILP: Cooking a meal without watching the water boil

- Without ILP:
  - The chef follows the recipe like a rigid checklist: chop vegetables, then boil water, then preheat the oven, then mix the sauce
    - Even when the water is boiling or the oven is preheating, the chef just waits
    - The kitchen has resources, but they sit idle
- With ILP:
  - The chef keeps the kitchen busy
    - While the water boils, chop vegetables
    - While the oven preheats, mix the sauce
    - While something simmers, prepare the next independent step
- The chef is not breaking the recipe; the chef is simply noticing which steps do not depend on each other and overlapping them

In a CPU …

- ILP is the CPU's way of refusing to stand around while the water boils
- ILP means a CPU finds independent instructions *within* a single **program thread** and executes them at the same time or overlaps them using different processor resources
  - However, instructions that *depend* on earlier results still have to wait

Overall throughput of GPUs is largely determined by …

- The **number of SPs** present
- The **bandwidth** to global memory
- How well **the programmer** makes use of the parallel architecture they are working with

To use a military analogy for how CUDA splits problems

- We have an *army* (a **grid**) of *soldiers* (**threads**)
- The army is split into a number of *units* (**blocks**), each commanded by a lieutenant
- The unit is split into *squads* of 32 soldiers (a **warp**), each commanded by a sergeant

Coordination across threads

- To perform some action, central command (the kernel/host program) must provide some action plus some data
- Each soldier (thread) works on their individual part of the problem
- Threads may from time-to-time swap data with one another under the coordination of either the sergeant (the warp) or the lieutenant (the block)
- However, any coordination with other units (blocks) must be performed by central command (the kernel/host program)
- When you think about how a CUDA program will implement concurrency
  - Think of orchestrating **thousands of threads in this very hierarchical manner**

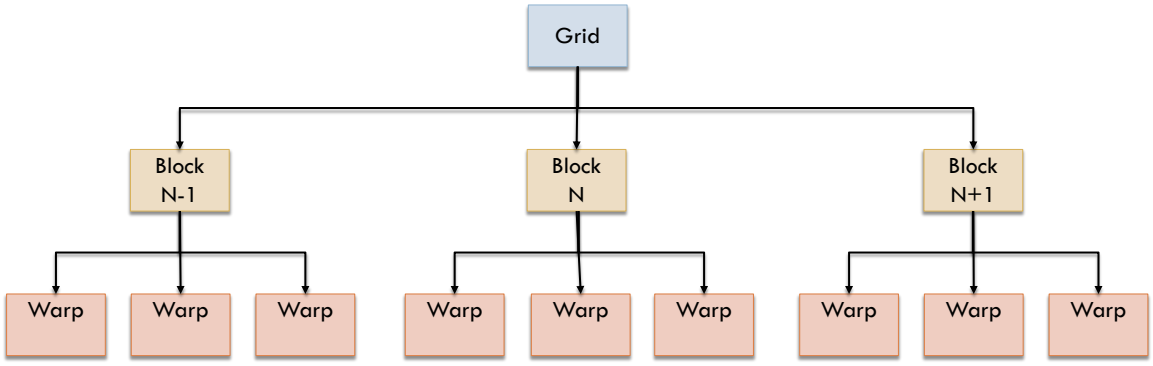
CUDA splits problems into grids of blocks, each containing multiple threads [1/2]

- The **blocks** may run in any order
- Only a subset of the blocks will ever execute at any one point in time
- A block must execute from start to completion and may be run on one of the N SMs (streaming multiprocessors)

CUDA splits problems into grids of blocks, each containing multiple threads [2/2]

- Blocks are allocated from the grid of blocks to any SM that has free slots
- Initially this is done on a round-robin basis, so each SM gets an equal distribution of blocks
- For most kernels, the **number of blocks** needs to be in the order of *eight or more times the number of physical SMs* on the GPU
  - Recall that CUDA kernel is a function that gets executed on the GPU

GPU-based view of threads



The contents of this slide-set are based on the following references

- Shane Cook. *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs (Applications of GPU Computing)*. ISBN-10/ISBN-13: 0124159338/978-0124159334. 1st Edition. Morgan Kaufmann. [Chapters 2,3]
- Kelleher, John D.. Deep Learning (MIT Press Essential Knowledge series). The MIT Press. [Chapter 4]