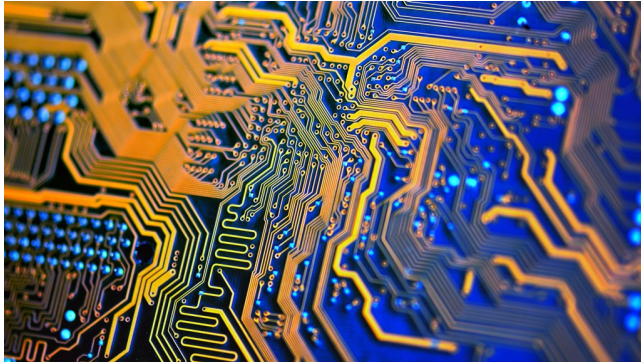


CS 250: FOUNDATION OF COMPUTER SYSTEMS

[**GRAPHICS PROCESSING UNITS GPUs**]



SHRIDEEP PALICKARA
Computer Science
Colorado State University

COMPUTER SCIENCE DEPARTMENT



1

Frequently asked questions from the previous class survey

- How is range scan different from BST or B-Tree searchers?
- Are B-Trees partially resident in memory?
- Pointers for internal nodes?



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.2

2

Topics covered in this lecture

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



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.3

3

But I've read this script and the costume fits, so I'll play my part.
Cleopatra, The Lumineers. 2016.

GPUs

COMPUTER SCIENCE DEPARTMENT



COLORADO STATE UNIVERSITY

4

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



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.5

5

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 graphical processor unit (GPGPU) programming
 - ▣ The design had taken its first steps in moving away from fixed function units




COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.6

6




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

COMPUTER SCIENCE DEPARTMENT



COLORADO STATE UNIVERSITY

7

GPUs (and NVIDIA) eating the CPUs 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)

Ingredients of a fine chip
 Selected financial measures


	Revenues 2023*, \$bn	Market capitalisation Jan 31st 2024, \$trn
Nvidia	59.1	1.52
Intel	54.2	0.18
AMD	22.7	0.27

	R&D [†] spending 2023*, \$bn	Cash & equivalents Q3 2023, \$bn
Nvidia	6.6	18.3
Intel	16.0	30.7
AMD	5.9	5.8

	Operating margin 2023*, %	Forward p/e [‡] ratio Jan 31st 2024
Nvidia	59.6	29.9
Intel	8.6	29.2
AMD	21.0	40.8

*Financial-year forecasts for Nvidia
[†]Research and development. [‡]Price-to-earnings
 Sources: Bloomberg; LSEG Workspace

IMAGE: THE ECONOMIST



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
 COMPUTER SCIENCE DEPARTMENT

GPUs

L28.8

8

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: 4Ghz
 - 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



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.9

9

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



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.10

10

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



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.11

11

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**



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.12

12

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



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.13

13

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**)



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.14

14

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



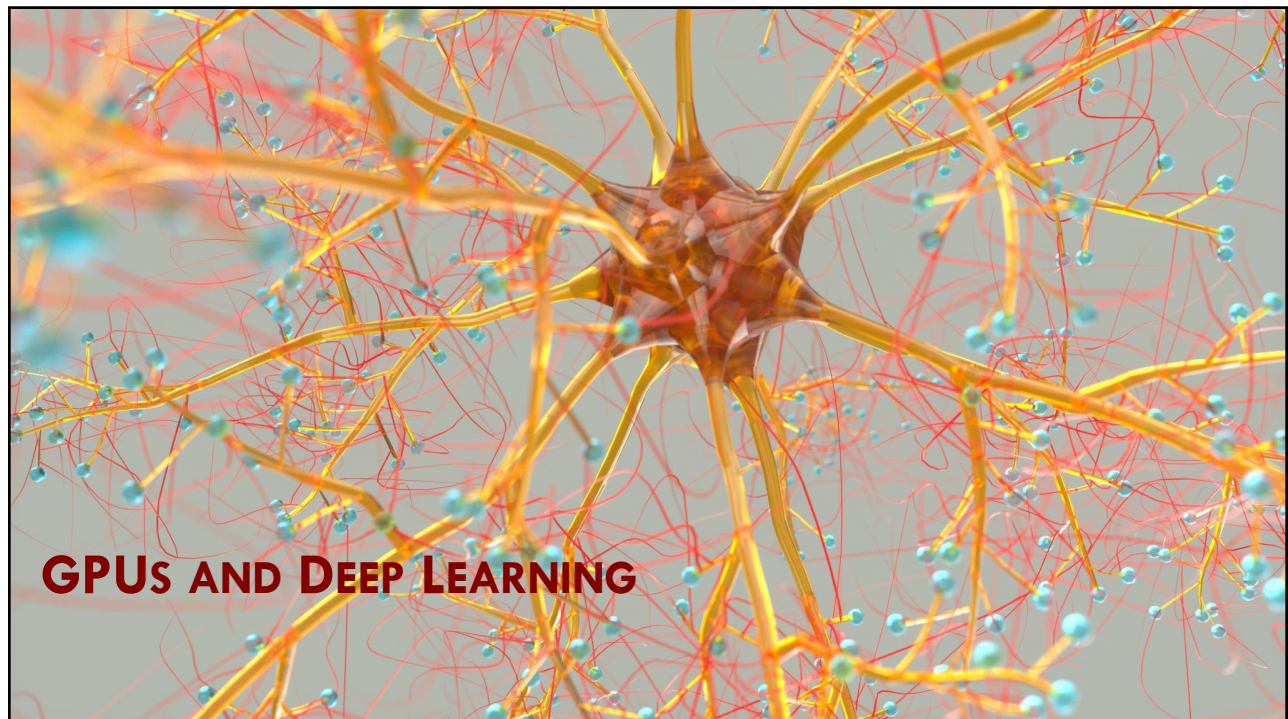
COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.15

15



16

GPUs and Deep learning

- From a computational perspective, a major breakthrough for deep learning occurred in the late 2000s with the adoption of graphical 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
- Consequently, GPUs are ideal hardware to speed up neural network training, and their use has made a significant contribution to the development of the field



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.17

17

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)



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.18

18

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



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.19

19

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



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.20

20

A very, very brief history of what powered deep learning

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



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.21

21

GETTING BACK TO GPUS

COMPUTER SCIENCE DEPARTMENT



COLORADO STATE UNIVERSITY

22

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

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



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.23

23

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



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.24

24

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



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



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



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.27

27

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**



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.28

28

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)

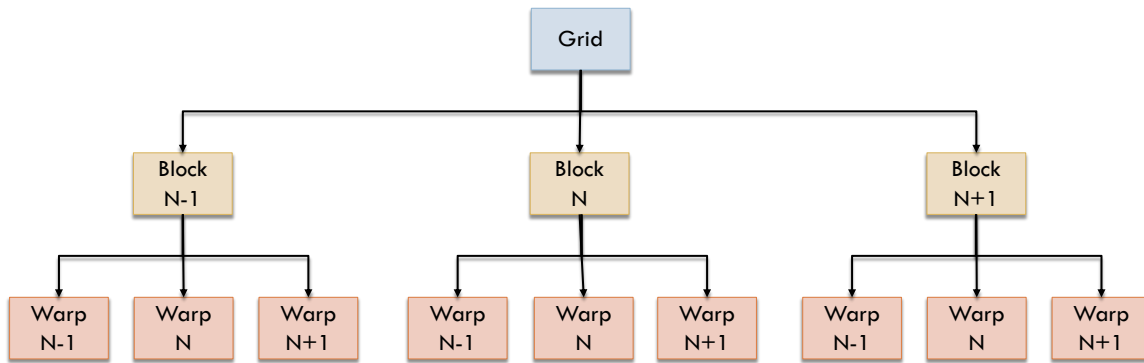


CUDA splits problems into grids of blocks, each containing multiple threads [2/2]

- 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



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.31

31



32

How computing has evolved

- Over the last decade or so, computing has moved
 - ▣ **From** one *limited by computational throughput* of the processor
 - ▣ **To** one where *moving the data* is the primary limiting factor
- When designing a processor in terms of processor real estate, compute units (or ALUs—algorithmic logic units) are cheap
 - ▣ They run at high speed, and consume little power and physical die space



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.33

33

However, ALUs are of **little use without operands**

- Considerable amounts of **power and time** are consumed in moving the operands to and from these functional units
- In modern computer designs this is addressed via the use of multilevel caches



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

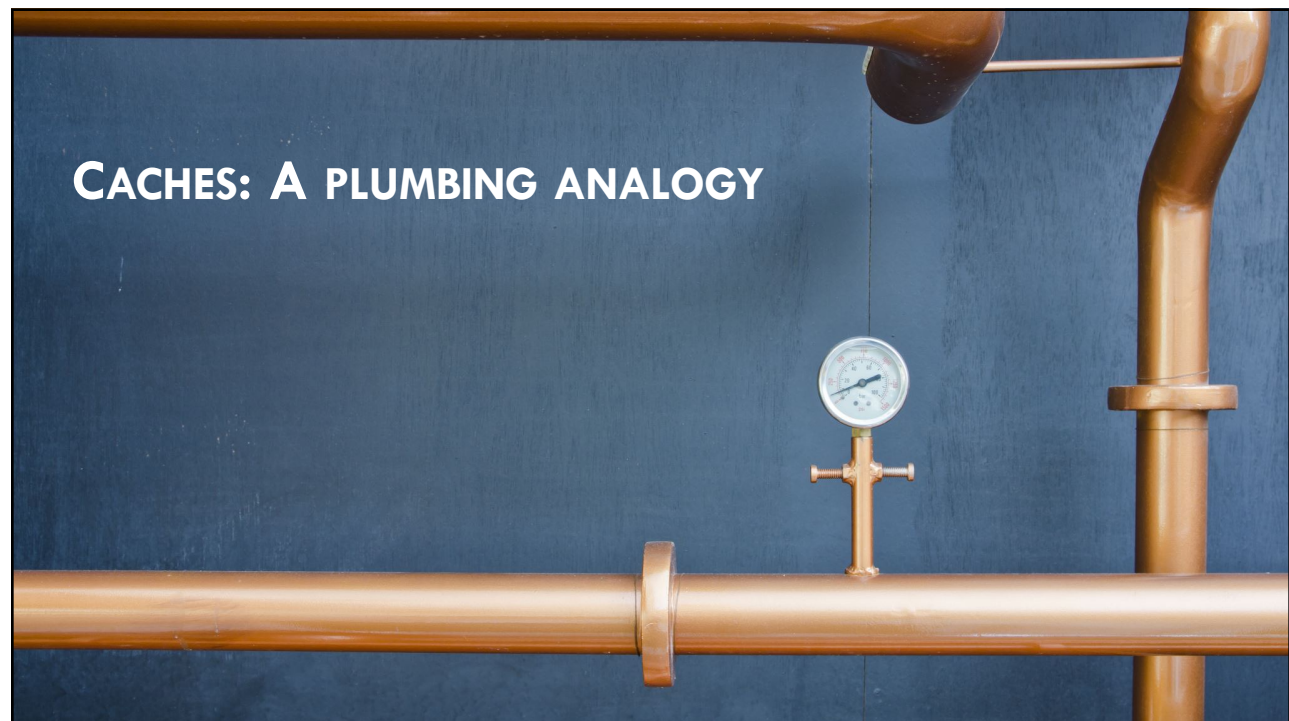
GPUs

L28.34

34

Caches and locality

- Caches work on the principle of either spatial (close in the address space) or temporal (close in time) locality
- Thus, data that has been *accessed before*, will likely be accessed again (**temporal locality**)
- Data that is *close to the last accessed data* will likely be accessed in the future (**spatial locality**)
- Caches work well where the task is repeated many times



Caches: A plumbing analogy

- Consider a plumber with a *toolbox* (a **cache**) that can hold four tools
- A number of the jobs being attended to are similar
 - ▣ So, the same four tools are *repeatedly used* (a **cache hit**)
- However, a significant number of jobs require additional tools; if the plumber does not know in advance what the job will entail?
 - ▣ Arrives and starts work
 - ▣ Partway through the job, the plumber needs an additional tool
 - ▣ Since it's not in the *toolbox* (**L1 cache**), plumber retrieves the item from the *van* (**L2 cache**)



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.37

37

Some jobs are tougher for the plumber

- Occasionally the plumber needs a special tool
 - ▣ Must leave the job, drive to the local *hardware store* (**global memory**), fetch the needed item, and return
- Plumber does not know *how long* (the latency) this operation will take
 - ▣ There may be *congestion* on the freeway and/or *queues* at the hardware store (other processes **competing for main memory access**)
 - ▣ Clearly, this is not a very efficient use of the plumber's time



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.38

38

While fetching new tools the plumber is not working on the problem at hand

- Each time a different tool or part is needed?
 - ▣ It needs to be fetched by the plumber from either the van or the hardware store
- While this might seem bad, fetching data from an HDD or an SSD is even worse, akin to **ordering an item** at the hardware store
 - ▣ In comparative form, data arrives by *snail mail*



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.39

39

Typical latencies to global memory are in the order of hundreds of clocks

- Increasingly, the answer to this problem from traditional CPU processor design has been to increase the size of the cache
 - ▣ In effect, **arrive with a bigger van** so *fewer trips* to the hardware store are necessary
- There is, however, an **increasing cost** to this approach
 - ▣ Both in terms of **capital outlay** for a larger van and
 - ▣ The time it takes to **search a bigger van** for the tool/part



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.40

40

The approach taken by most CPU designs today?

- Arrive with a *van* (**L2 cache**) and a *truck* (**L3 cache**)
- In the extreme case of the server processors?
 - ▣ A huge 18-wheeler is brought in to try to ensure that the plumber is kept busy for just that little bit longer
- All of this work is necessary because of one fundamental reason
 - ▣ The CPUs are designed to run software where the **programmer does not have to care about locality**



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

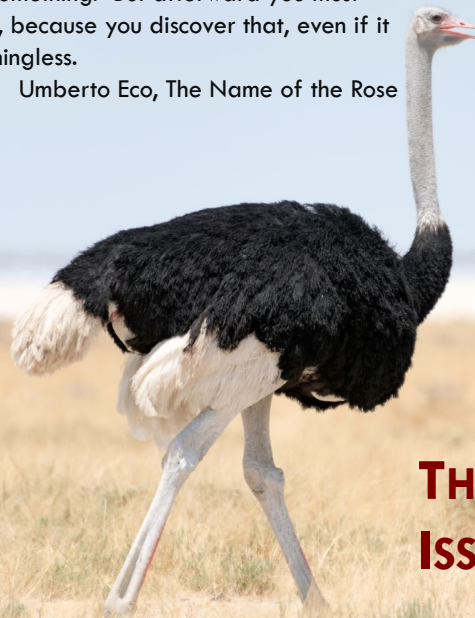
GPUs

L28.41

41

The order that our mind imagines is like a net, or like a ladder, built to attain something. But afterward you must throw the ladder away, because you discover that, even if it was useful, it was meaningless.

Umberto Eco, *The Name of the Rose*



**THERE IS NO DENYING THE
ISSUE OF LOCALITY**

42

Locality

- Locality is an issue, regardless of whether the CPU processor tries to hide it from the programmer or not
- The denial that this is an issue on CPUs is what leads to the huge amount of hardware necessary to deal with memory latency



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.43

43

GPU design takes a different approach [1/2]

- Places the GPU programmer in charge of dealing with locality
- Instead of an 18-wheeler truck, gives programmer a number of small vans and a very large number of plumbers
- The **programmer must deal with locality**
 - Programmer needs to think in advance about what tools/parts (memory locations/data structures) will be needed for a given job
 - These then need to be **collected in a single trip** to the hardware store (global memory)
 - And placed in the correct van (on chip memory) for a given job **at the outset**



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.44

44

GPU design takes a different approach [2/2]

- Given that this data has been collected, **as much work as possible** needs to be performed with the data
 - To avoid having to fetch and return it only to fetch it again later for another purpose
- Thus, the continual cycle of work-stall-fetch from global memory is broken



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.45

45

GPU design: Analogy

- **Workers at a production line**
- Workers are supplied with baskets of parts to process
 - This simple process of planning ahead allows the programmer to schedule memory loads into the on-chip memory **before** they are needed
- What if each worker individually fetches widgets one at a time from the store manager's desk?
 - Terribly inefficient!

Source: Wikipedia



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

46

But this could be done with a CPU cache as well, couldn't it? Yes, and no ...

- You can use special cache for instructions that allow **prefilling** of the cache with data you expect the program to use later
- The downside of the cache approach over the GPU shared memory approach is **eviction** and **dirty data**



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.47

47

Data in a cache is said to be **dirty** if it has been written by the program

- To free up the space in the cache for new useful data?
 - ▣ The dirty data must be written back to global memory before the cache space can be used again
 - ▣ This means instead of one trip to global memory of an unknown latency?
 - We now have two — one to write the old data and one to get the new data



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.48

48

Let's contrast this with the explicit local memory model such as the **GPU's shared memory**

- The big advantage of the programmer-controlled on-chip memory is that the programmer is in control of *when* the writes happen
- If you are performing some local transformation of the data, **there may be no need to write the intermediate transformation** back to global memory
- With a cache, the cache controller *does not know* what needs to be written and what can be discarded
 - Thus, it writes everything, potentially creating lots of useless memory traffic that may, in turn, cause unnecessary congestion on the memory interface



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.49

49

Cache coherency

- One important distinction between the caches found in GPUs and CPUs is **cache coherency**
- In a cache-coherent system, a write to a memory location needs to be **communicated to all levels of the cache in all cores**
 - Thus, all CPU processor cores see the same view of memory at any point in time
 - This is one of the key factors that *limits the number of cores in a processor*
 - Communication becomes increasingly more expensive in terms of time as the processor core count increases
 - The worst case in a cache-coherent system?
 - Each core writes adjacent memory locations as each write forces a global update to every core's cache



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.50

50

Non cache-coherent systems

- A non cache-coherent system by comparison does not automatically update the other core's caches
- **Relies on the programmer** to write the output of each processor core to separate areas/addresses
- Supports the view of a program where a single core is responsible for a single or small set of outputs
- CPUs follow the cache-coherent approach whereas the GPU does not
 - ▣ Allows GPUs to scale to a far larger number of cores (SMs) per device



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.51

51

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]



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

GPUs

L28.52

52