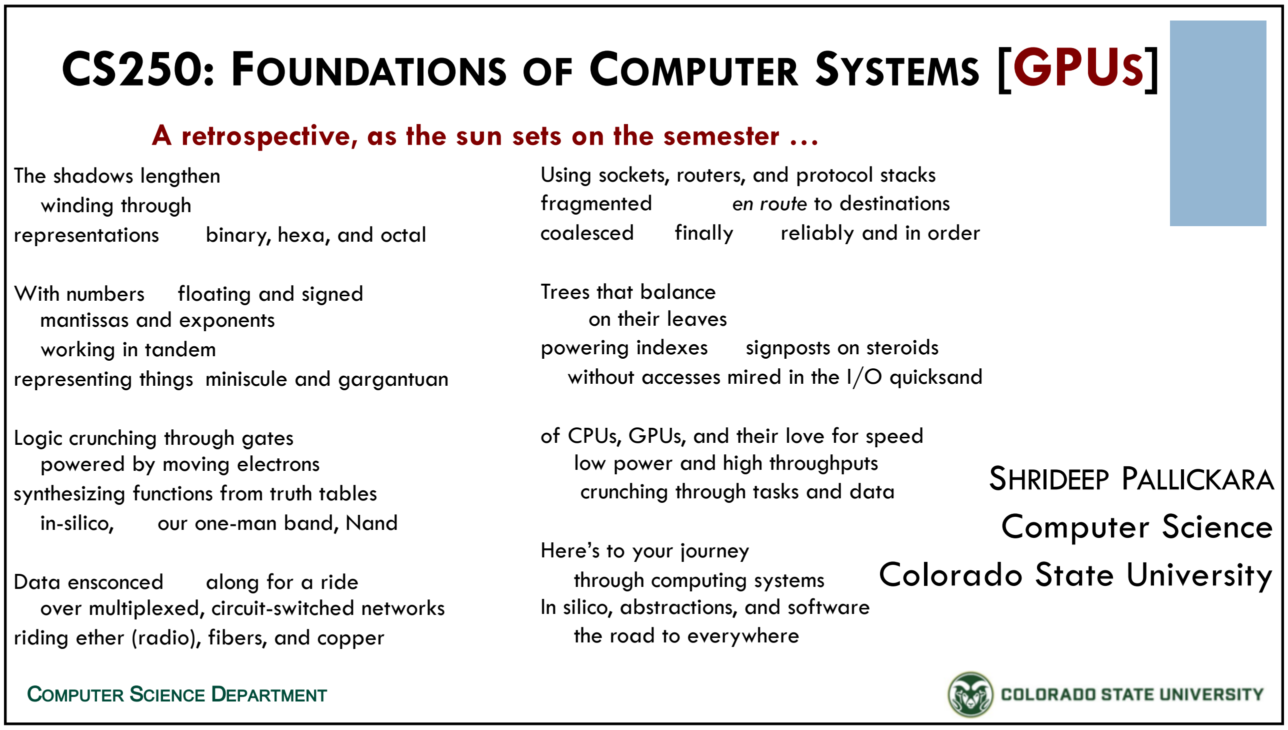


CS250: Foundations of Computer Systems [GPUs]

A retrospective, as the sun sets on the semester …

The shadows lengthen
winding through
representations binary, hexa, and octal

With numbers floating and signed
mantissas and exponents
working in tandem
representing things miniscule and gargantuan

Logic crunching through gates
powered by moving electrons
synthesizing functions from truth tables
in-silico, our one-man band, Nand

Data ensconced along for a ride
over multiplexed, circuit-switched networks
riding ether (radio), fibers, and copper

Using sockets, routers, and protocol stacks
fragmented *en route* to destinations
coalesced finally reliably and in order

Trees that balance
on their leaves
powering indexes signposts on steroids
without accesses mired in the I/O quicksand

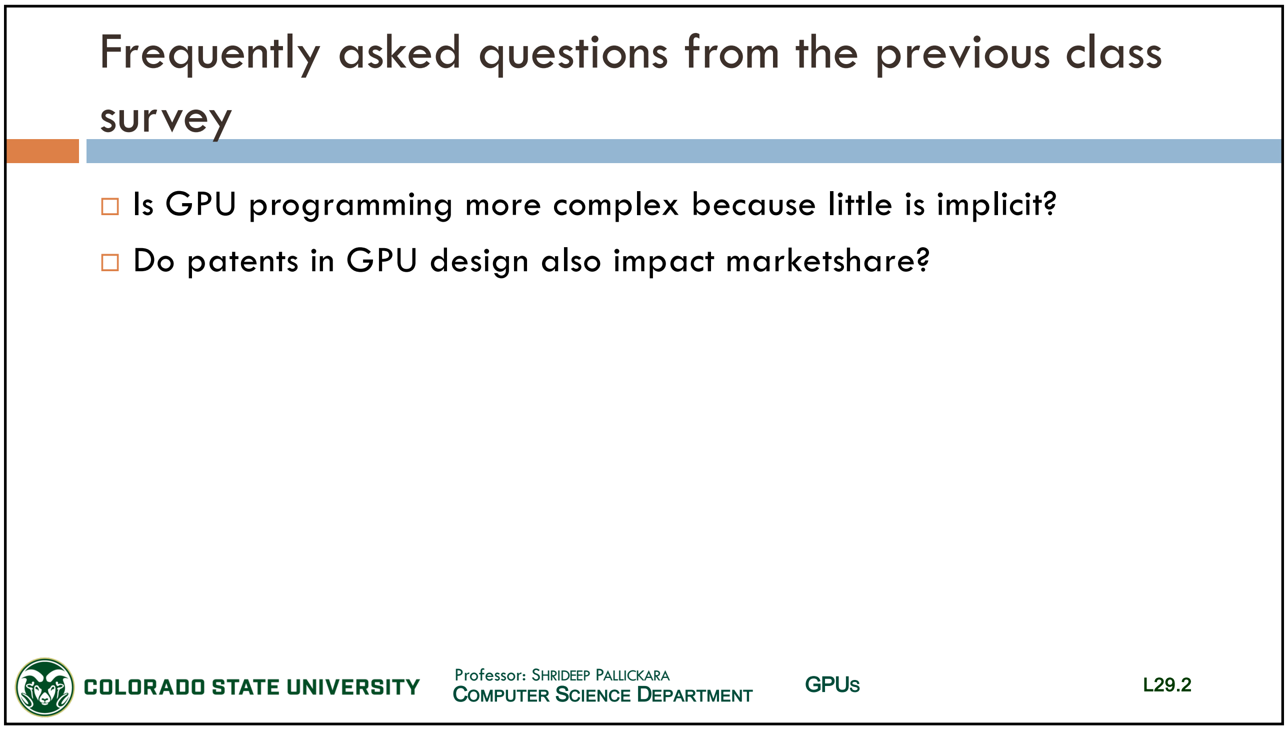
of CPUs, GPUs, and their love for speed
low power and high throughputs
crunching through tasks and data

Here's to your journey
through computing systems
In silico, abstractions, and software
the road to everywhere

Shrideep Pallickara
Computer Science
Colorado State University

Computer Science Department   Colorado State University

1



Frequently asked questions from the previous class survey

- Is GPU programming more complex because little is implicit?
- Do patents in GPU design also impact marketshare?

Colorado State University   Professor: Shrideep Pallickara Computer Science Department   GPUs   L29.2

2

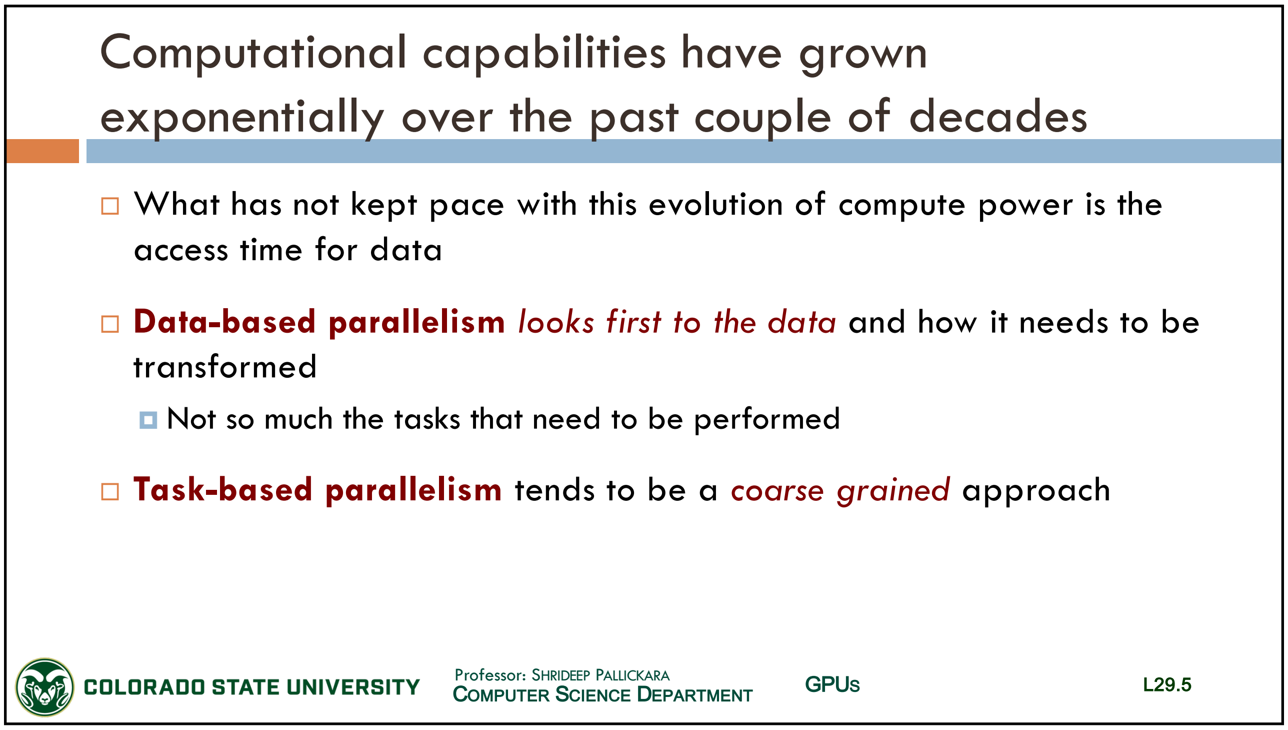
Topics covered in this lecture

- GPUs
  - Data and task based parallelism
  - Flynn's taxonomy
  - CPU-GPU differences
- CSx55
- Final Exam

3

DATA BASED PARALLELISM

4

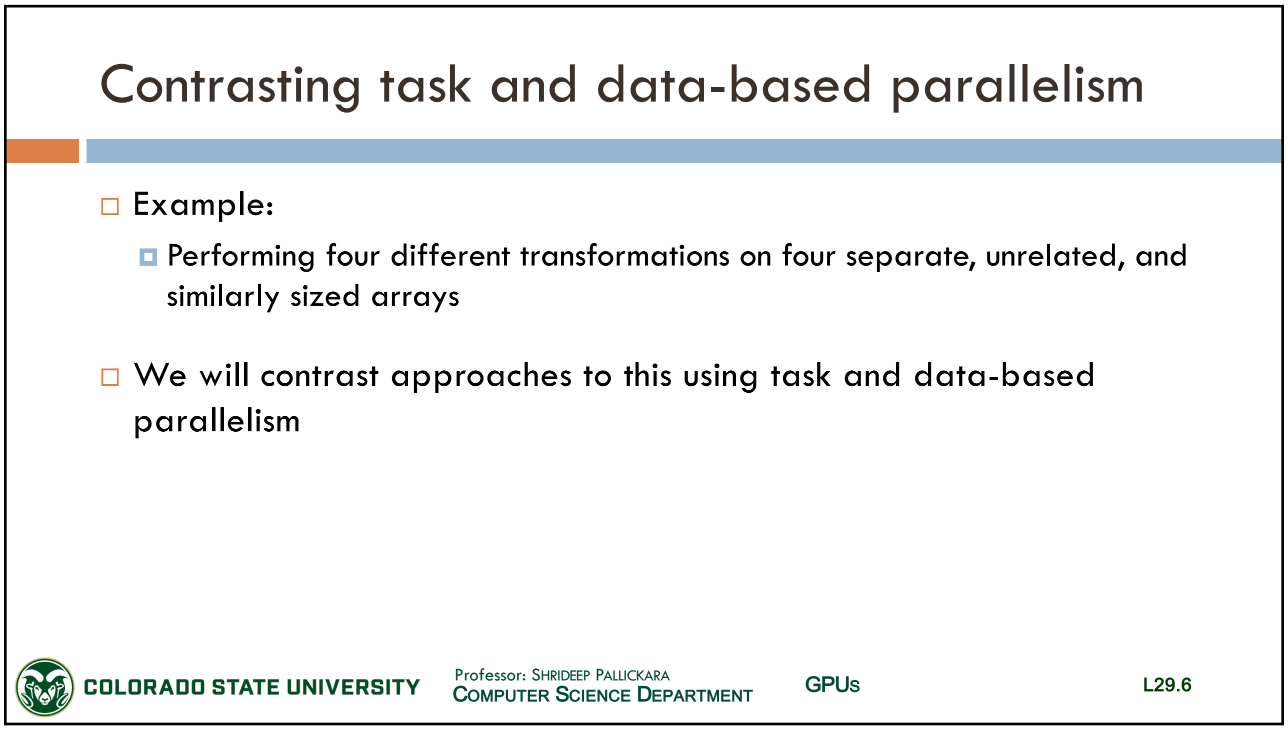


Computational capabilities have grown exponentially over the past couple of decades

- What has not kept pace with this evolution of compute power is the access time for data
- **Data-based parallelism** *looks first to the data* and how it needs to be transformed
  - Not so much the tasks that need to be performed
- **Task-based parallelism** tends to be a *coarse grained* approach

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT GPUs L29.5

5



Contrasting task and data-based parallelism

- Example:
  - Performing four different transformations on four separate, unrelated, and similarly sized arrays
- We will contrast approaches to this using task and data-based parallelism

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT GPUs L29.6

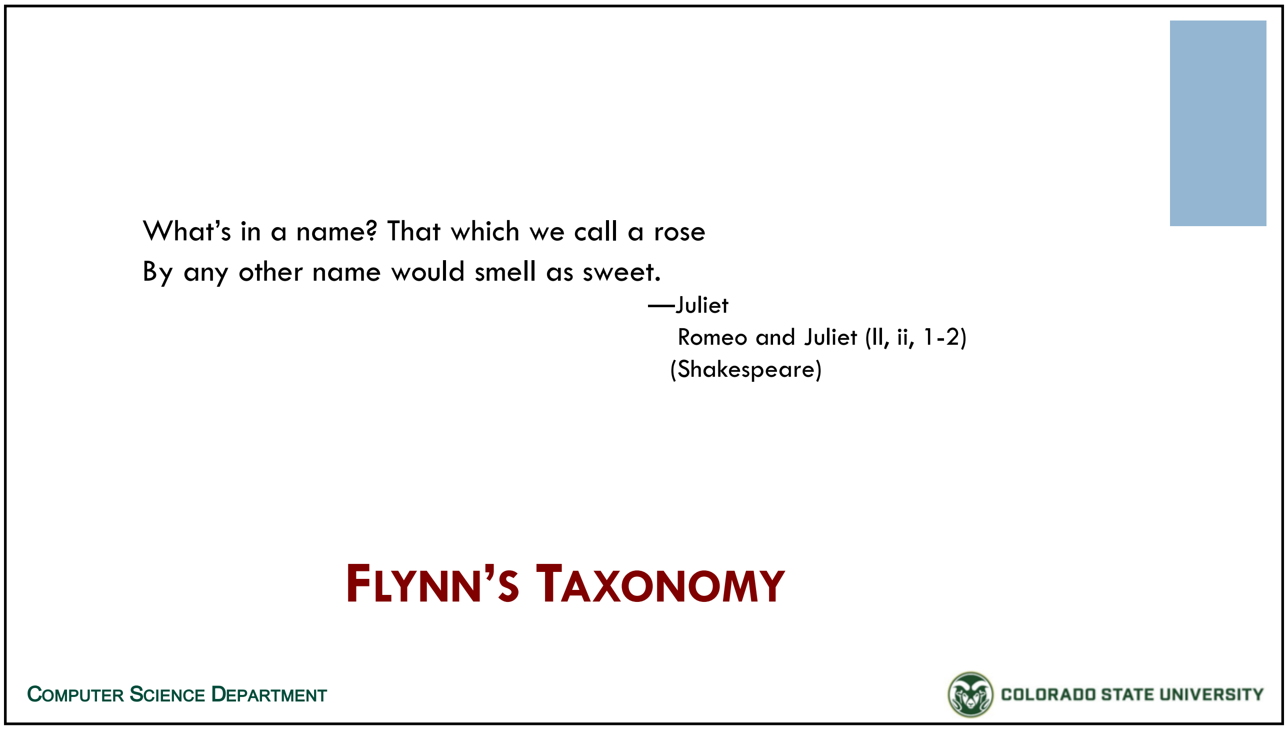
6

Task-based parallelism

- Assign one array to each of the CPU cores (or SMs in the GPU)
  - On the CPU side we could create four threads or processes to achieve this
  - On GPUs, we would create four separate kernels, one to process each array and run it concurrently
- The parallel decomposition of the problem is driven by thinking about the tasks or transformations, not the data

Data-based parallelism

- A data-based decomposition would instead **split the first array into four blocks**
  - Assign one CPU core or one GPU SM to each section of the array
- Once completed, the remaining three arrays would be processed in a similar way
- The parallel decomposition here is driven by thinking about the data first and the transformations second



What's in a name? That which we call a rose
By any other name would smell as sweet.

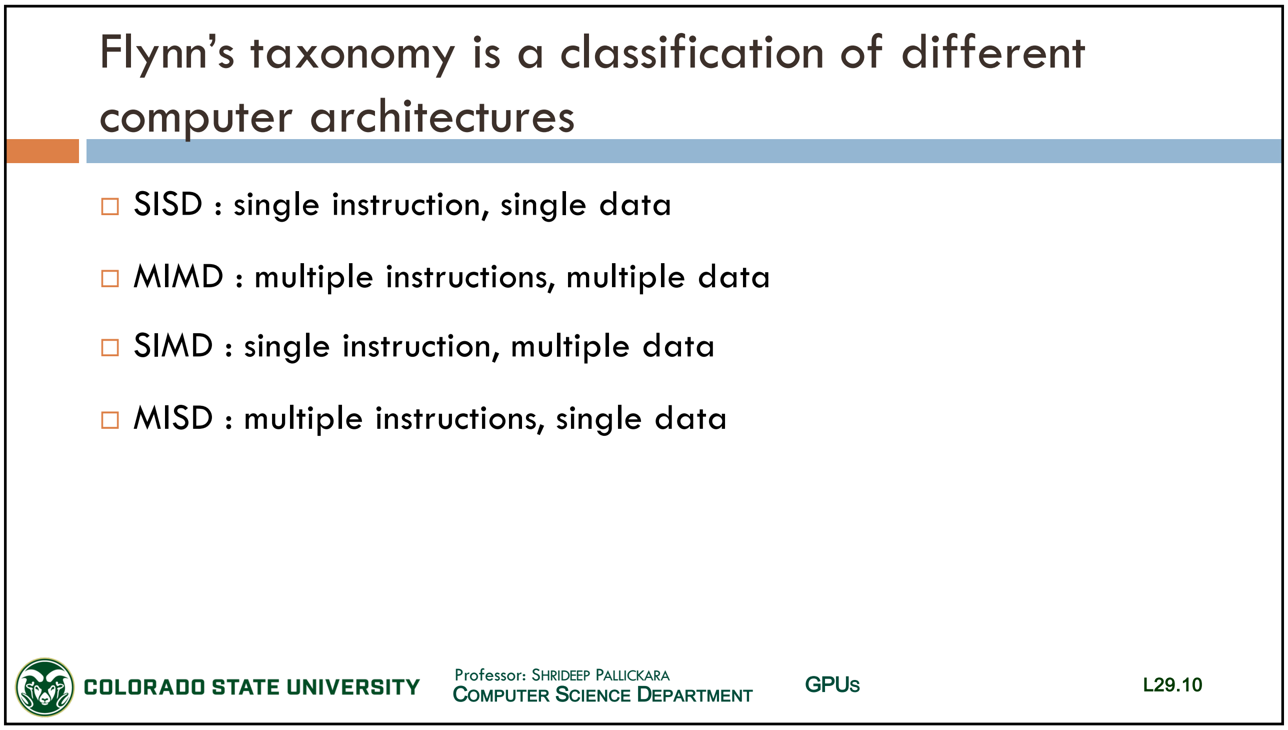
—Juliet
Romeo and Juliet (II, ii, 1-2)
(Shakespeare)

FLYNN'S TAXONOMY

COMPUTER SCIENCE DEPARTMENT

COLORADO STATE UNIVERSITY

9



Flynn's taxonomy is a classification of different computer architectures

- SISD : single instruction, single data
- MIMD : multiple instructions, multiple data
- SIMD : single instruction, multiple data
- MISD : multiple instructions, single data

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT GPUs L29.10

10

The standard serial programming we are familiar with follows the **SISD** model

- There is a single instruction stream working on a single data item at any one point in time
- This equates to a single-core CPU able to perform one task at a time
- Of course, it's quite possible to provide the illusion of being able to perform more than a single task
  - By simply switching between tasks very quickly, so-called time-slicing

MIMD systems are what we see today in dual- or quad-core desktop machines

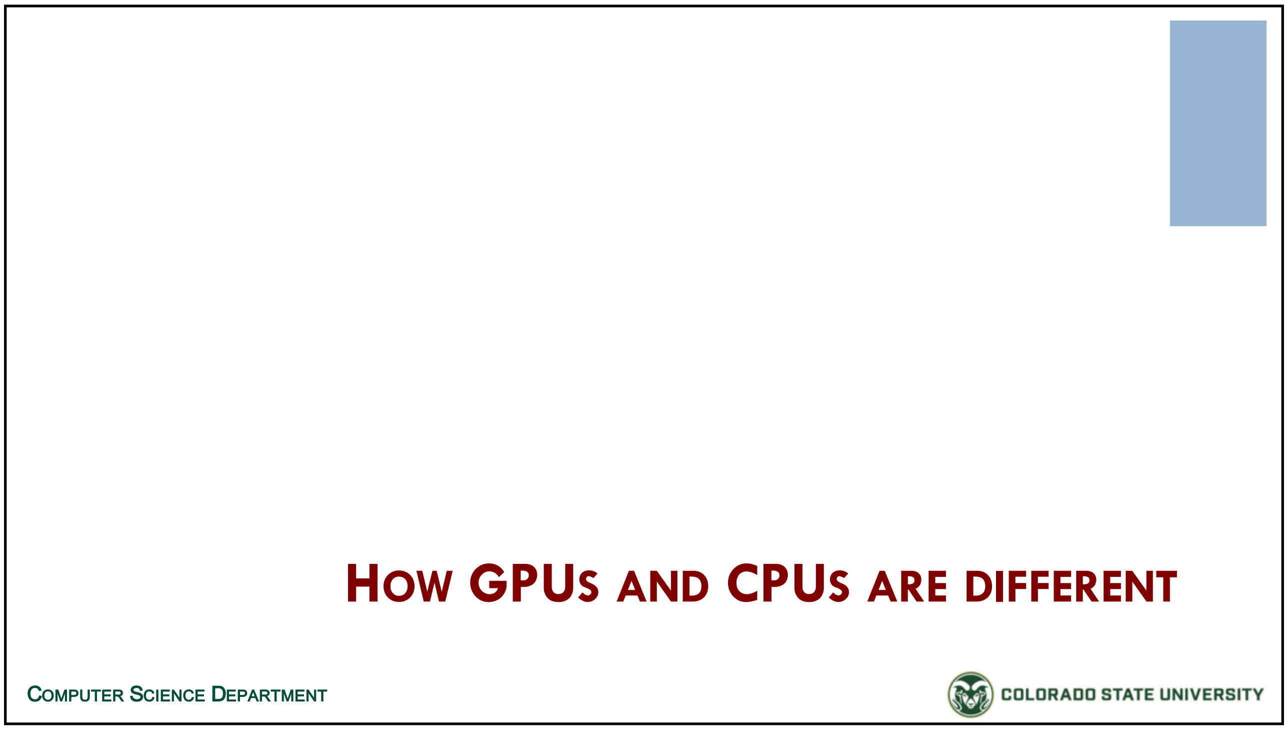
- Typical multi-core desktops have a worker pool of threads/processes that the OS will allocate to one of N CPU cores
- Each thread/process has an independent stream of instructions
  - The hardware contains all the control logic for **decoding many separate instruction streams** concurrently

SIMD systems try to simplify the approach taken in MIMD systems

- They accomplish this with the **data parallelism** model
- SIMD systems follow a single instruction stream at any one point in time.
  - Require a single set of logic inside the device to decode and execute the instruction stream, rather than multiple-instruction decode paths
- By removing this silicon real estate from the device?
  - Can be smaller, cheaper, consume less power, and run at higher clock rates than their MIMD cousins

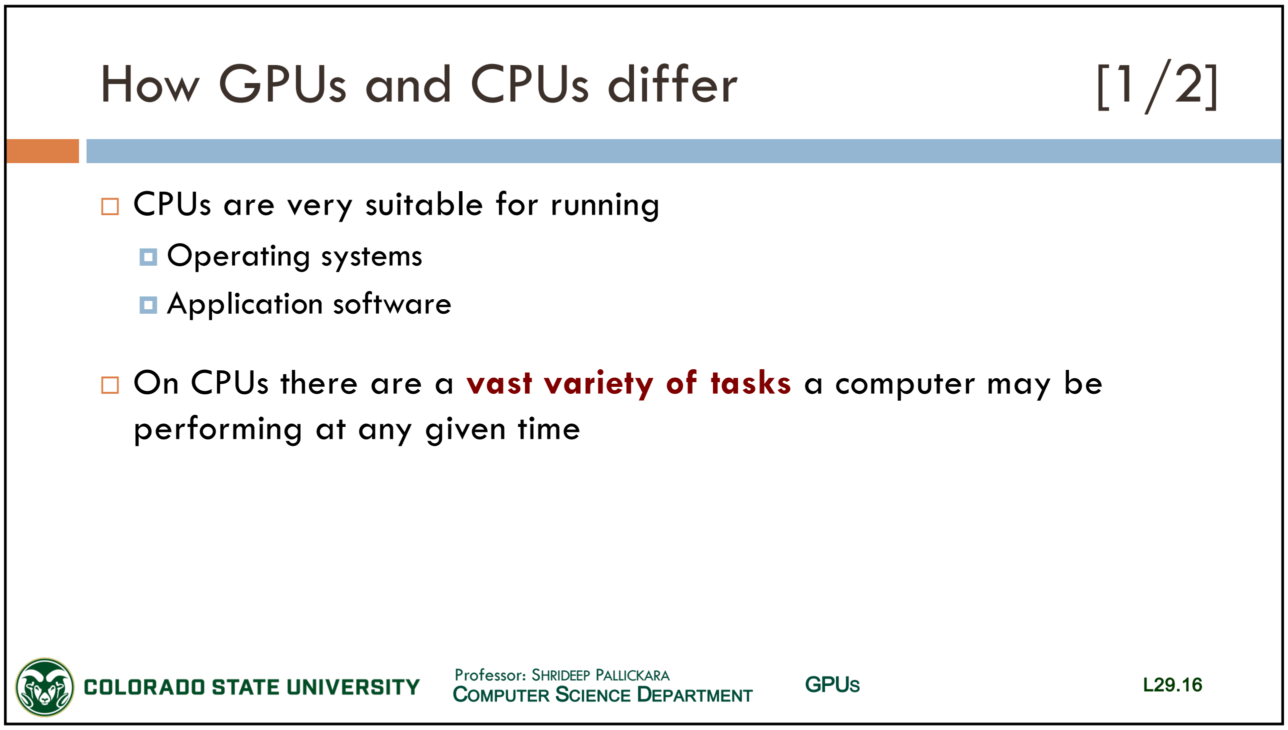
Multiple instruction, single data (**MISD**)

- Uncommon architecture that is generally used for fault tolerance
- A type of parallel computing architecture where many functional units perform **different operations on the same data**
  - E.g.: Heterogeneous systems operating on the same data stream and needing to agree on the result
    - Space shuttle flight control computer
  - Task replication may be considered as MISD as well
    - Executing the same instructions redundantly in order to detect and mask errors
- Applications for MISD are much less common than MIMD and SIMD
  - MIMD and SIMD are often more appropriate for common data parallel techniques



HOW GPUs AND CPUs ARE DIFFERENT

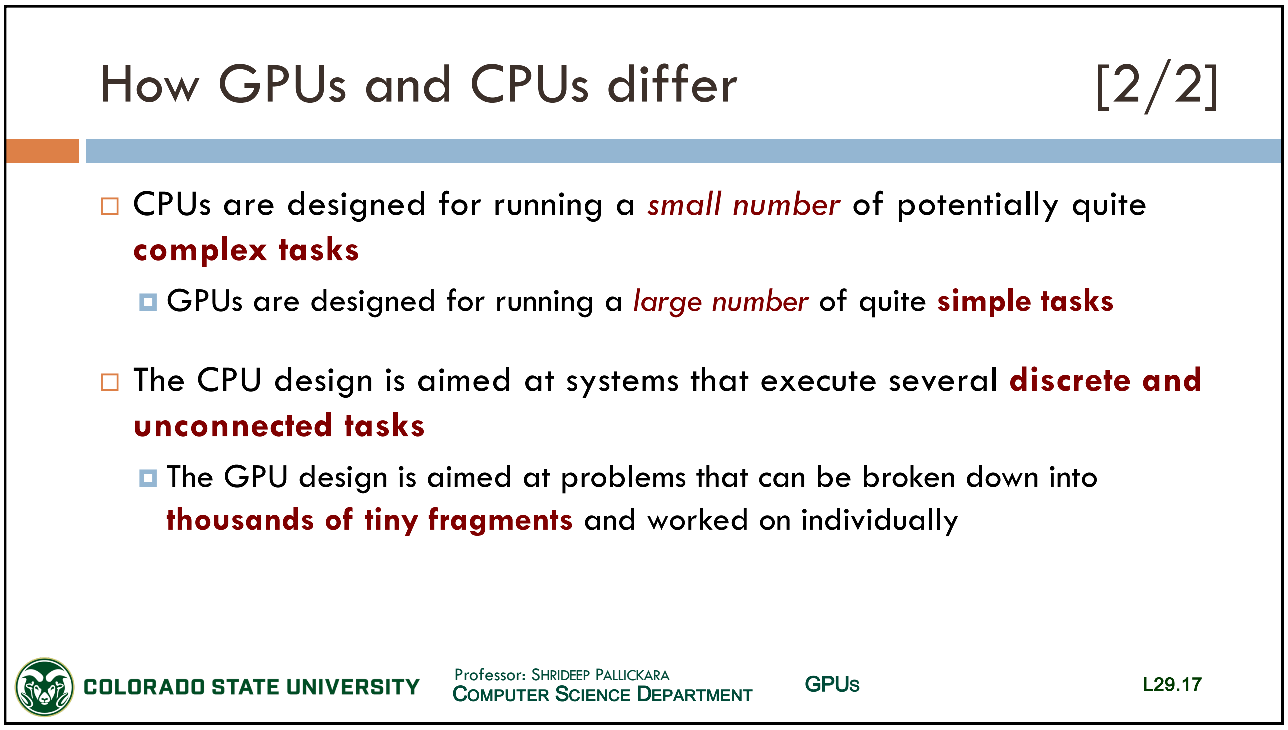
15



How GPUs and CPUs differ [1/2]

- CPUs are very suitable for running
  - Operating systems
  - Application software
- On CPUs there are a **vast variety of tasks** a computer may be performing at any given time

16

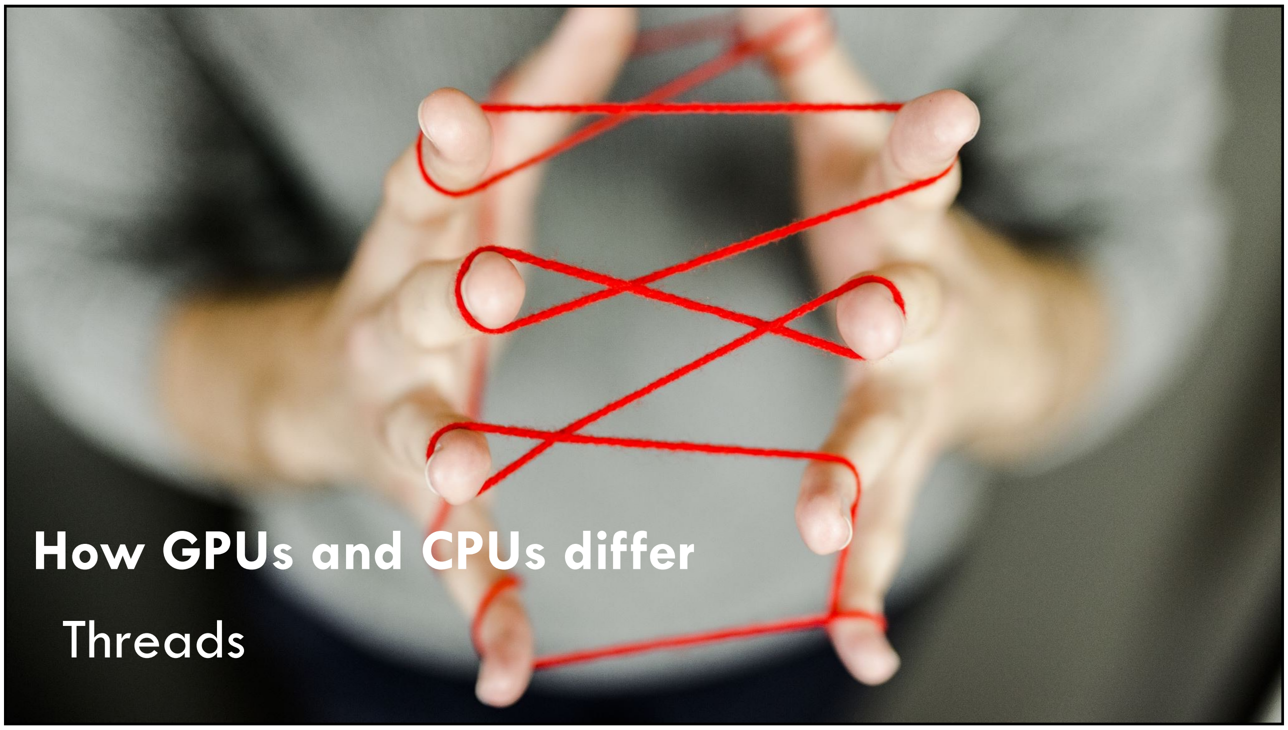


How GPUs and CPUs differ [2/2]

- CPUs are designed for running a *small number* of potentially quite **complex tasks**
  - GPUs are designed for running a *large number* of quite **simple tasks**
- The CPU design is aimed at systems that execute several **discrete and unconnected tasks**
  - The GPU design is aimed at problems that can be broken down into **thousands of tiny fragments** and worked on individually

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT GPUs L29.17

17



How GPUs and CPUs differ

Threads

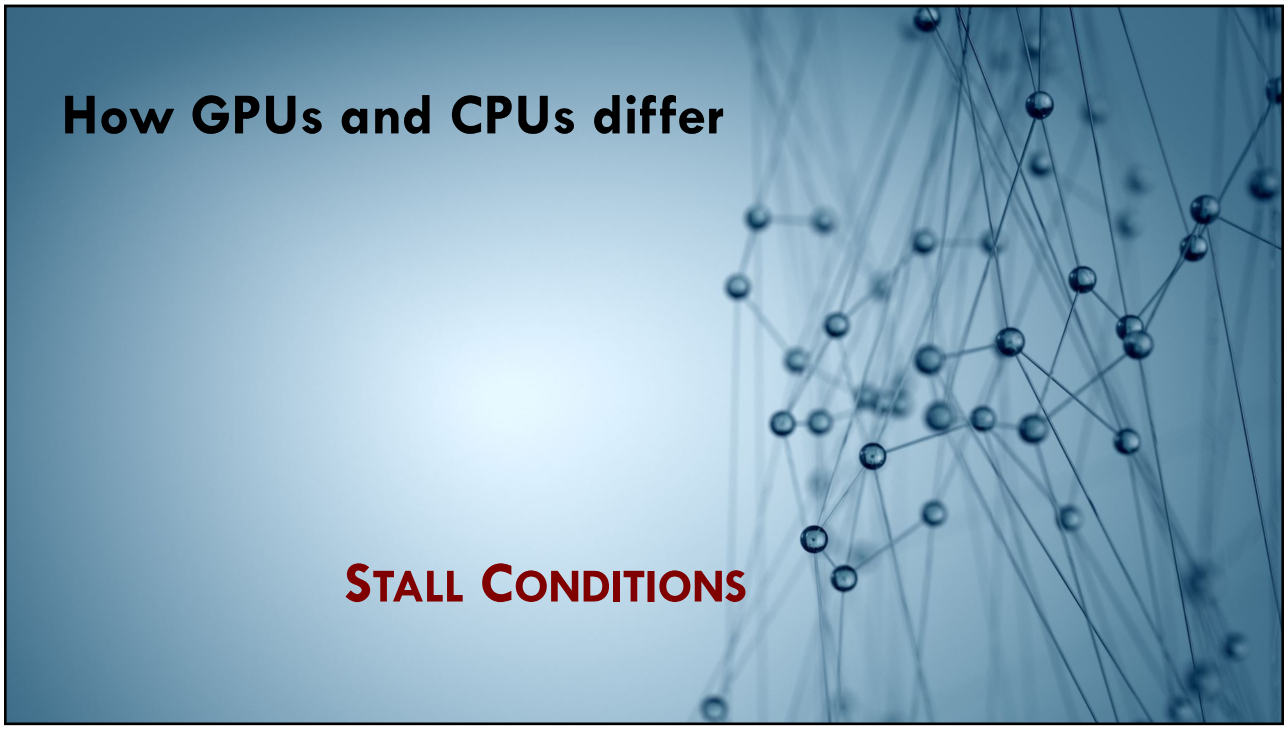
18

CPUs and GPUs support threads in very different ways [1/2]

- The **CPU has a small number of registers per core** that must be used to execute any given task
  - To achieve this, they rapidly context switch between tasks
  - On CPUs, *context switching is expensive* in terms of time
    - The entire register set must be saved to RAM and the next one restored from RAM

CPUs and GPUs support threads in very different ways [2/2]

- GPUs also use the same concept of context switching
- But instead of having a single set of registers, they have **multiple banks of registers**
  - A context switch simply involves *setting a bank selector* to switch in and out the current set of registers
    - Which is **several orders of magnitude faster** than having to save to RAM



How GPUs and CPUs differ

STALL CONDITIONS

21

Both CPUs and GPUs must deal with stall conditions [1/2]

- Stalls are generally caused by I/O operations and memory fetches
- The CPU does this by **context switching**
  - Provided that there are enough tasks, and the runtime of a thread is not too small, this works reasonably well
  - If there are not enough processes to keep the CPU busy, it will idle
  - If there are too many small tasks, each blocking after a short period?
    - The CPU will spend most of its time context switching; very little time doing work
    - CPU scheduling policies are often based on time slicing
    - As the number of threads increases, the percentage of time spent context switching becomes increasingly large and the efficiency starts to rapidly drop off

22

Both CPUs and GPUs must deal with stall conditions [2/2]

- GPUs are designed to handle stall conditions and **expect this to happen with high frequency**
- The GPU model is a data-parallel one and needs thousands of threads to work efficiently
- GPUs uses this pool of available work to ensure it always has something useful to work on
  - Thus, when it hits a memory fetch or must wait on a computation result?
    - The SPs simply switch to another instruction stream and return to the stalled instruction stream sometime later

GPUs also provide something quite unique

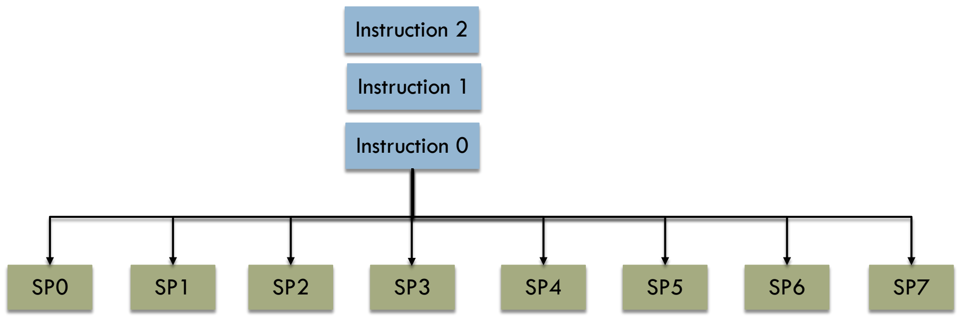
- High-speed memory next to the SM, so-called **shared memory**
- Programmer can leave data in this shared memory
  - Knowing that hardware will not evict it behind programmer's back
- This shared memory is also the primary mechanism communication between threads

The GPU's task execution model differs in two key ways

- Groups of N SPs execute in a **lock-step basis** running the same program but on different data
- The second is that, because of the **huge register file**
  - Switching threads has effectively zero overhead
  - As a result, GPUs can support a very large number of threads

Lock-step instruction dispatch in GPUs

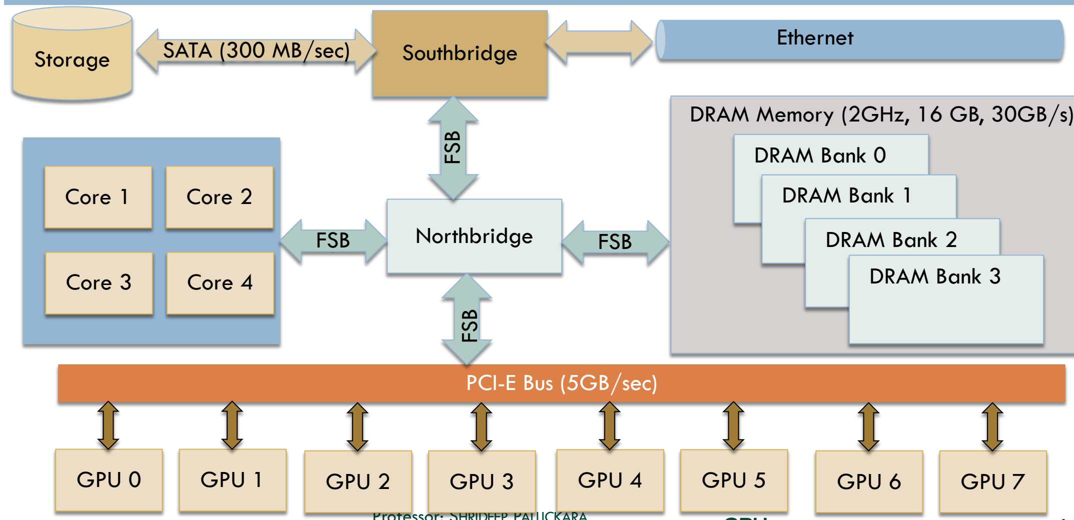
- Each instruction in the instruction queue is **dispatched to every SP within an SM**



A TYPICAL PC ARCHITECTURE

27

Typical PC architecture



28

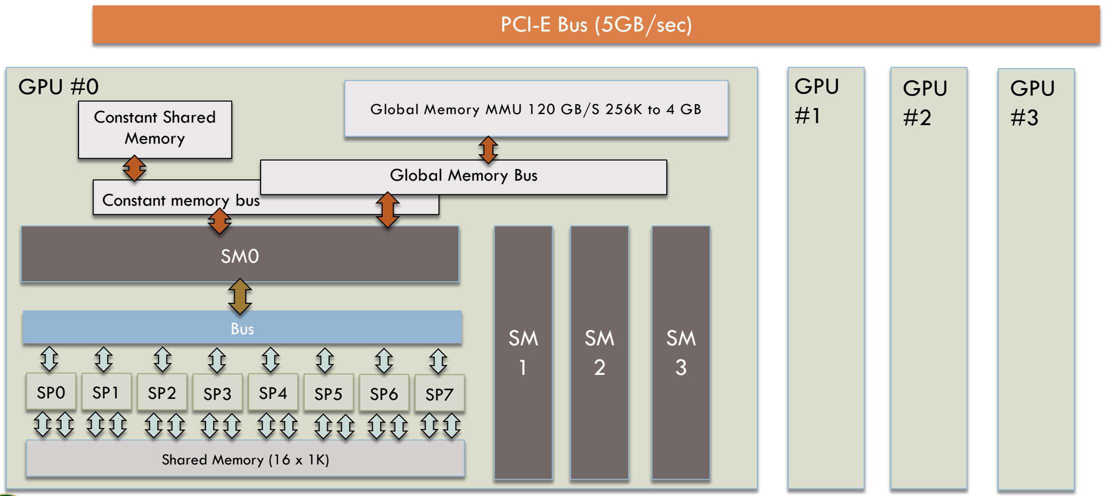
The GPU hardware

- The GPU hardware consists of a number of key blocks:
  - Memory (global, constant, shared)
  - Streaming multiprocessors (SMs)
  - Streaming processors (SPs)

GPUs and SMs

- A GPU device consists of one or more SMs
- Add more SMs to the device and you make the GPU able to
  - Process **more tasks** at the same time, or
  - Process the **same task quicker**, if you have enough parallelism in the task

Overview of a GPUs

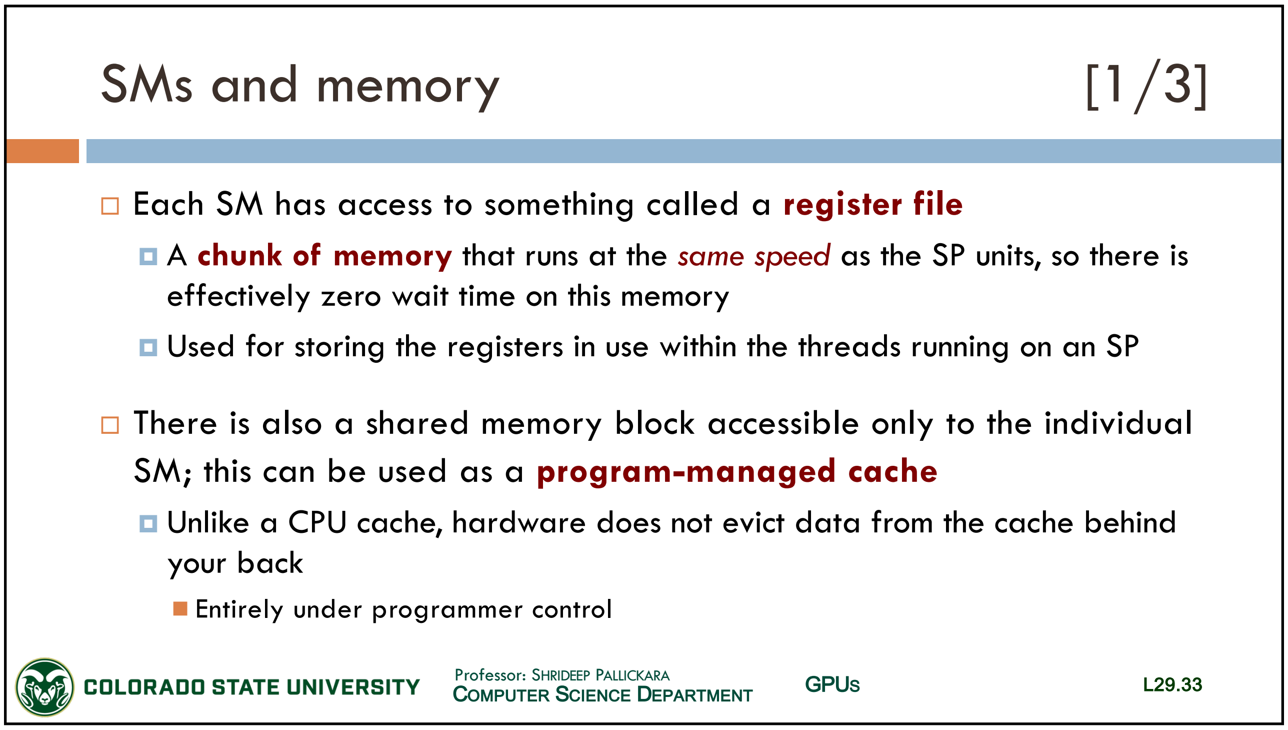


31

Taking a closer look at the SMs

- There are multiple SPs in each SM
  - Our diagram showed 8
  - In Fermi this grows to 32–48 SPs and in Kepler to 192
- Each hardware revision increases both the number of SMs and the number of SPs (in each SM)

32

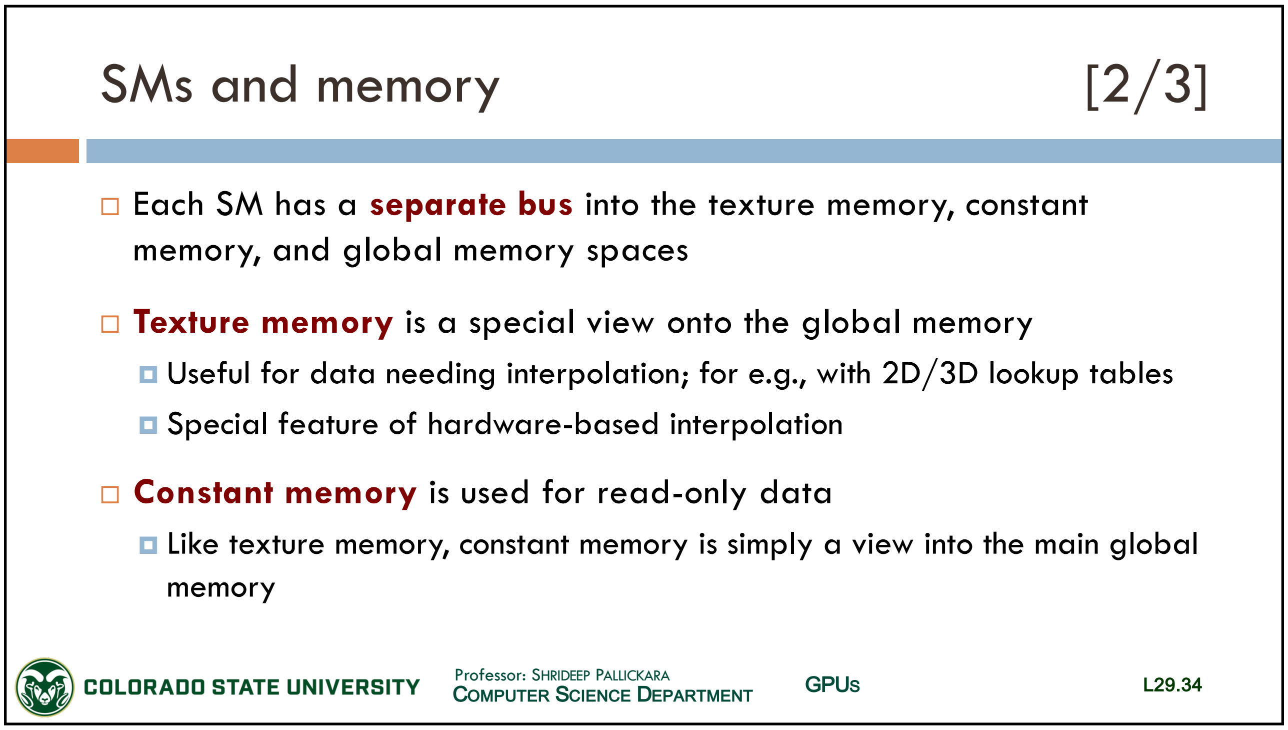


SMs and memory [1/3]

- Each SM has access to something called a **register file**
  - A **chunk of memory** that runs at the *same speed* as the SP units, so there is effectively zero wait time on this memory
  - Used for storing the registers in use within the threads running on an SP
- There is also a shared memory block accessible only to the individual SM; this can be used as a **program-managed cache**
  - Unlike a CPU cache, hardware does not evict data from the cache behind your back
    - Entirely under programmer control

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT GPUs L29.33

33



SMs and memory [2/3]

- Each SM has a **separate bus** into the texture memory, constant memory, and global memory spaces
- **Texture memory** is a special view onto the global memory
  - Useful for data needing interpolation; for e.g., with 2D/3D lookup tables
  - Special feature of hardware-based interpolation
- **Constant memory** is used for read-only data
  - Like texture memory, constant memory is simply a view into the main global memory

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT GPUs L29.34

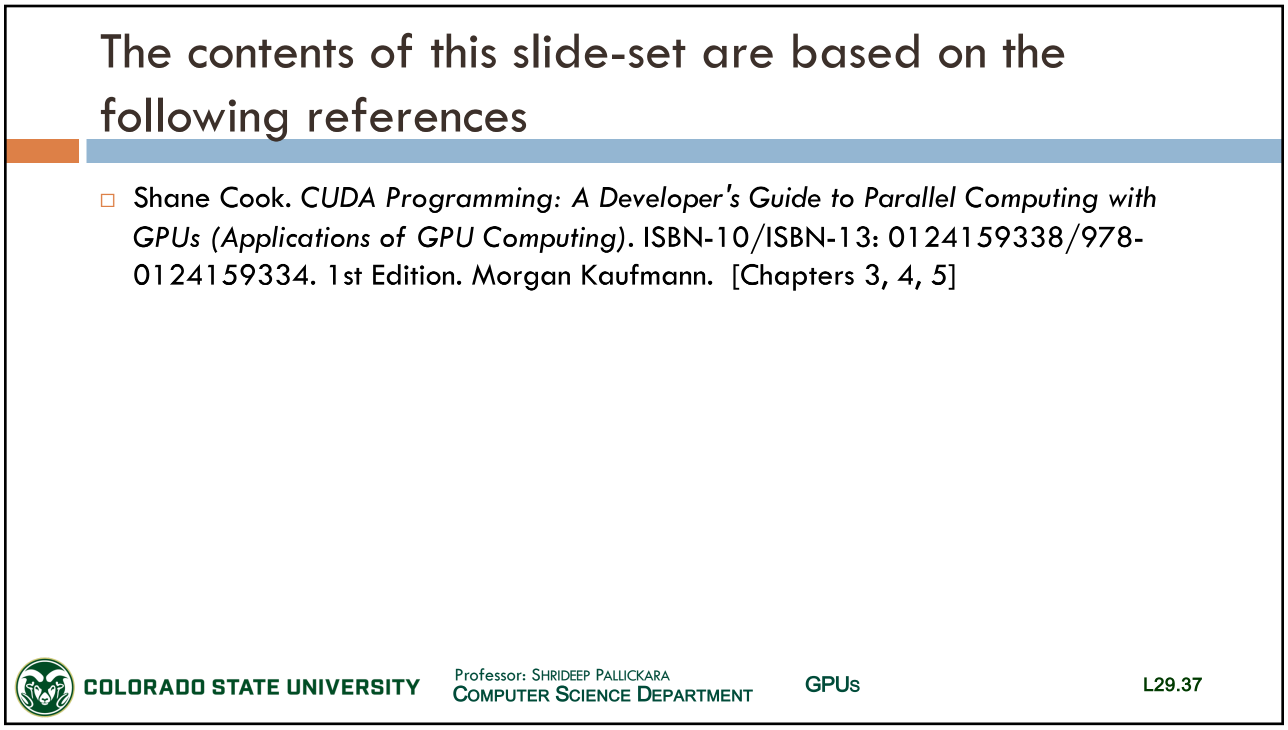
34

SMs and memory [3/3]

- Global memory is supplied via GDDR (Graphic Double Data Rate) on the graphics card
  - A high-performance version of DDR (Double Data Rate) memory
- Memory bus width can be up to **512 bits wide**, giving a bandwidth of 5 to 10 times more than found on CPUs
  - Up to 190 GB/s with the Fermi hardware

Single core CPUs and parallelism

- Most programs written in the past few decades, with the exception of perhaps the past 15 years or so, were single-thread programs
  - The primary hardware on which they would execute was a single-core CPU
- Sure, you had clusters and supercomputers that sought to exploit a high level of parallelism
  - **Duplicating the hardware** and having thousands of commodity servers instead of a handful of massively powerful machines



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 3, 4, 5]

COLORADO STATE UNIVERSITY Professor: Shrideep Pallickara
Computer Science Department GPUs L29.37

37