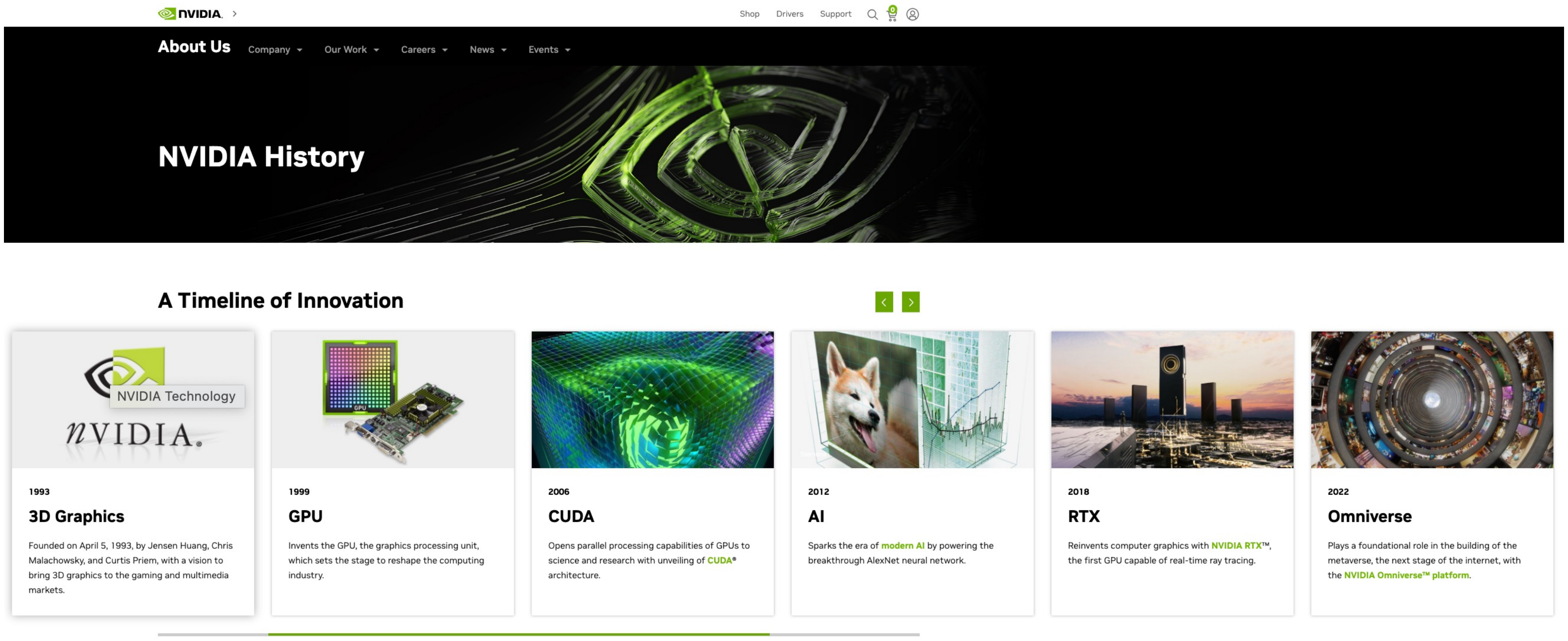
http://inst.eecs.berkeley.edu/~cs152

CS 152/252A Computer Architecture and Engineering

Sophia Shao

Lecture 18: GPUs



NVIDIA
Shop Drivers Support

About Us Company Our Work Careers News Events

NVIDIA History

A Timeline of Innovation

1993
3D Graphics
Founded on April 5, 1993, by Jensen Huang, Chris Malachowsky, and Curtis Priem, with a vision to bring 3D graphics to the gaming and multimedia markets.

1999
GPU
Invents the GPU, the graphics processing unit, which sets the stage to reshape the computing industry.

2006
CUDA
Opens parallel processing capabilities of GPUs to science and research with unveiling of CUDA® architecture.

2012
AI
Sparks the era of modern AI by powering the breakthrough AlexNet neural network.

2018
RTX
Reinvents computer graphics with NVIDIA RTX™, the first GPU capable of real-time ray tracing.

2022
Omniverse
Plays a foundational role in the building of the metaverse, the next stage of the internet, with the NVIDIA Omniverse™ platform.

https://www.nvidia.com/en-us/about-nvidia/corporate-timeline/

Last Time in Lecture

Vector supercomputers

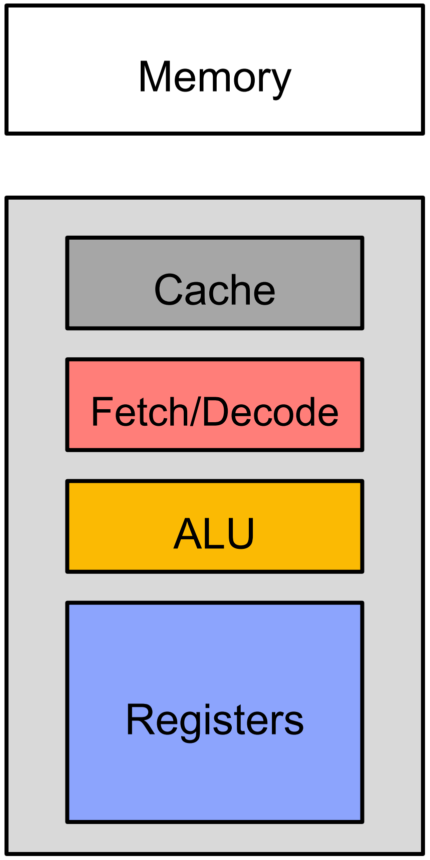
- Vector register versus vector memory
- Scaling performance with lanes
- Stripmining
- Chaining
- Masking
- Scatter/Gather

Types of Parallelism

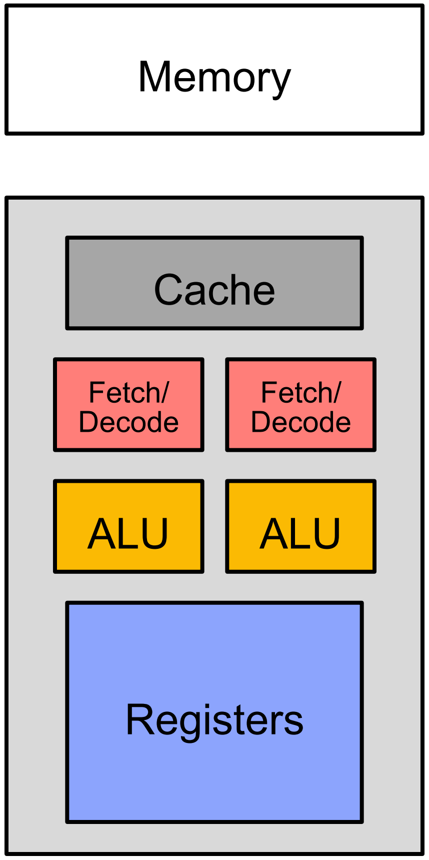
- Instruction-Level Parallelism (ILP)
  - Execute independent instructions from one instruction stream in parallel (pipelining, superscalar, VLIW)
- Thread-Level Parallelism (TLP)
  - Execute independent instruction streams in parallel (multithreading, multiple cores)
- Data-Level Parallelism (DLP)
  - Execute multiple operations of the same type in parallel (vector/SIMD execution)

- Which is easiest to program?
- Which is most flexible form of parallelism?
  - i.e., can be used in more situations
- Which is most efficient?
  - i.e., greatest tasks/second/area, lowest energy/task

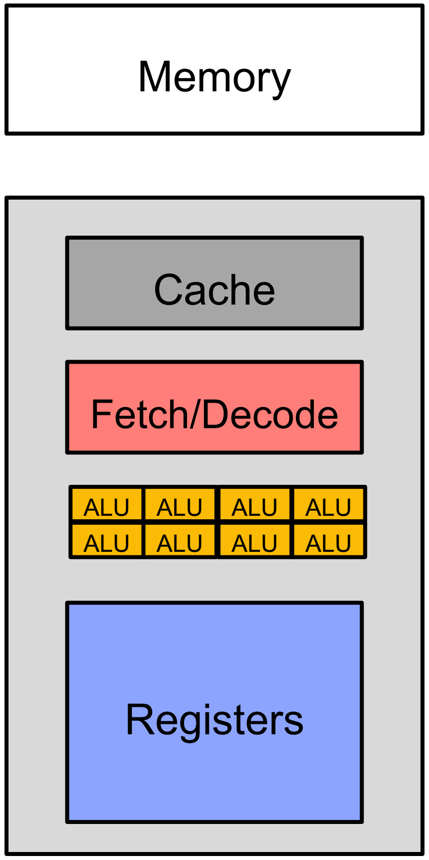
Simplified Processor Evolution



Single-core, single-thread, scalar processor

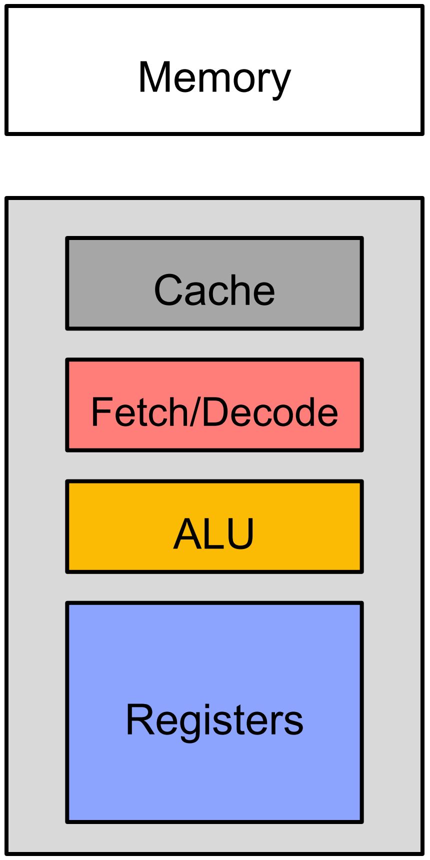


Single-core, single-thread, **super-scalar** processor

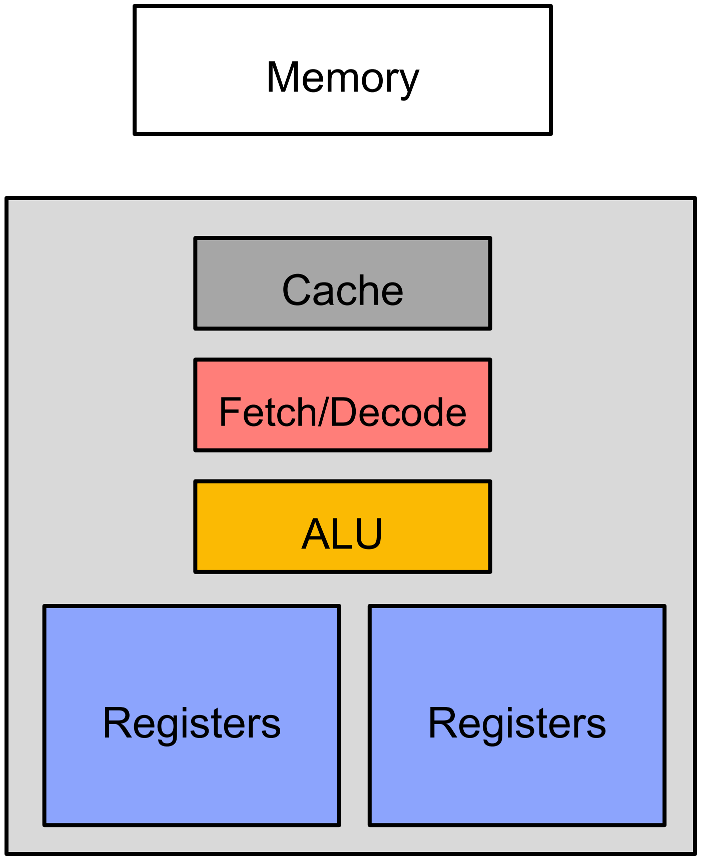


Single-core, single-thread, **SIMD/vector** processor

Simplified Processor Evolution

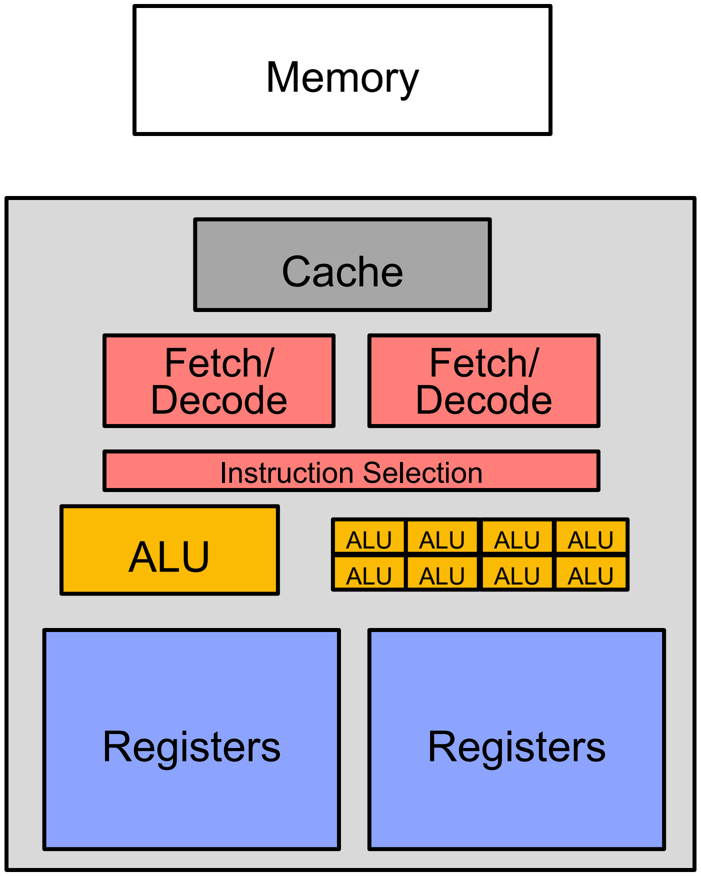


Single-core,
single-thread,
scalar processor

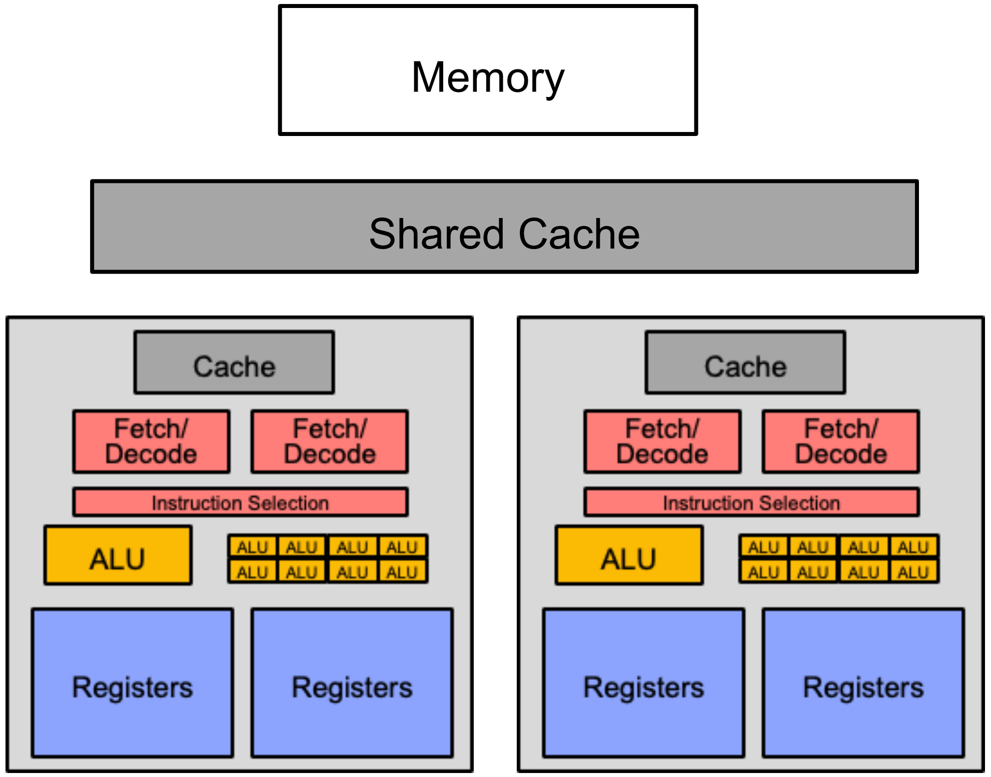


Single-core,
multi-thread,
scalar processor

Simplified Processor Evolution



Single-core, **multi-thread**, **super-scalar, OoO, SIMD** processor



multi-core, multi-thread, super-scalar, OoO, SIMD processor

Resurgence of DLP

- Convergence of application demands and technology constraints drives architecture choice

- New applications, such as graphics, machine vision, speech recognition, machine learning, etc. all require large numerical computations that are often trivially data parallel

- SIMD-based architectures (vector-SIMD, subword-SIMD, SIMT/GPUs) are most efficient way to execute these algorithms

Graphics Processing Units (GPUs)

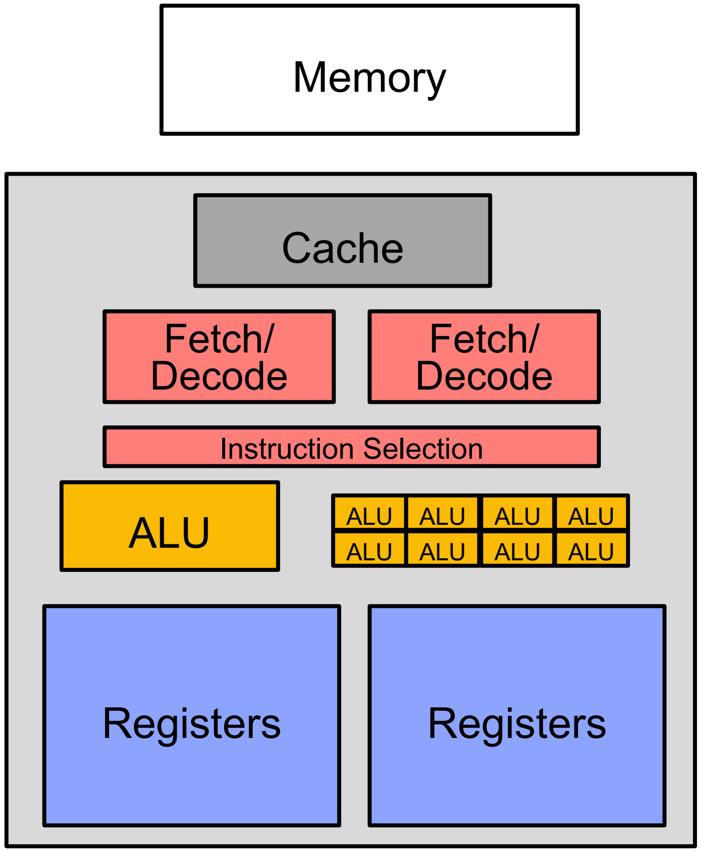
- Original GPUs were dedicated fixed-function devices for generating 3D graphics (mid-late 1990s) including high-performance floating-point units
  - Provide workstation-like graphics for PCs
  - User could configure graphics pipeline, but not really program it
- Over time, more programmability added (2001-2005)
  - E.g., New language Cg for writing small programs run on each vertex or each pixel, also Windows DirectX variants
  - Massively parallel (millions of vertices or pixels per frame) but very constrained programming model
- Some users noticed they could do general-purpose computation by mapping input and output data to images, and computation to vertex and pixel shading computations
  - Incredibly difficult programming model as had to use graphics pipeline model for general computation

General-Purpose GPUs (GP-GPUs)

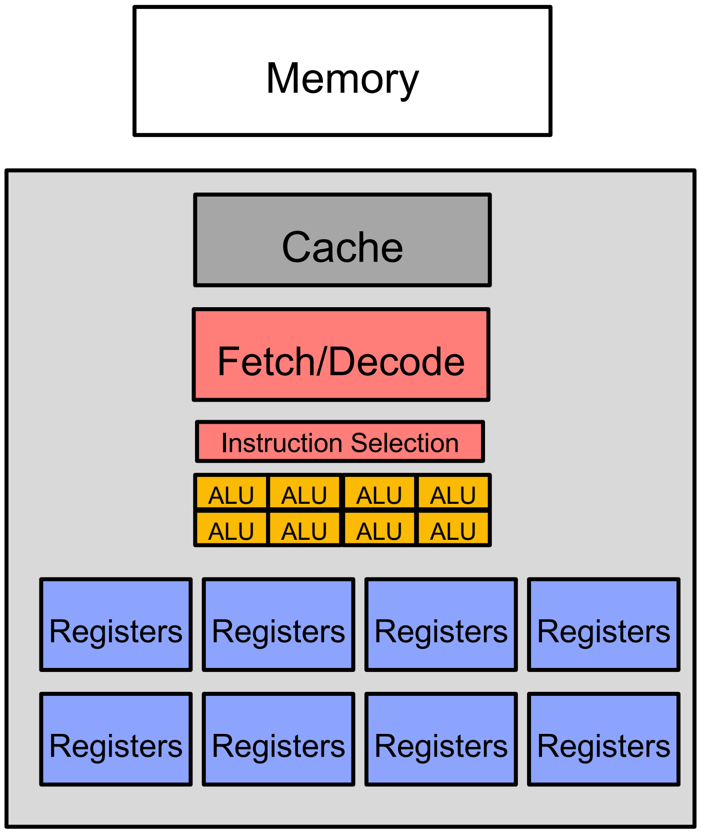
- In 2006, Nvidia introduced GeForce 8800 GPU, which supported a new programming language CUDA (in 2007)
  - "Compute Unified Device Architecture"
  - Subsequently, broader industry pushing for OpenCL, a vendor-neutral version of same ideas.
- Idea: Take advantage of GPU computational performance and memory bandwidth to accelerate some kernels for general-purpose computing
- Attached processor model: Host CPU issues data-parallel kernels to GP-GPU for execution
- This lecture has a simplified version of Nvidia CUDA-style model and only considers GPU execution for computational kernels, not graphics
  - Would need whole other course to describe graphics processing

GPU SIMT

- "Single instruction multiple thread"
  - Or multi-threaded SIMD/streaming multi-processor (SM)



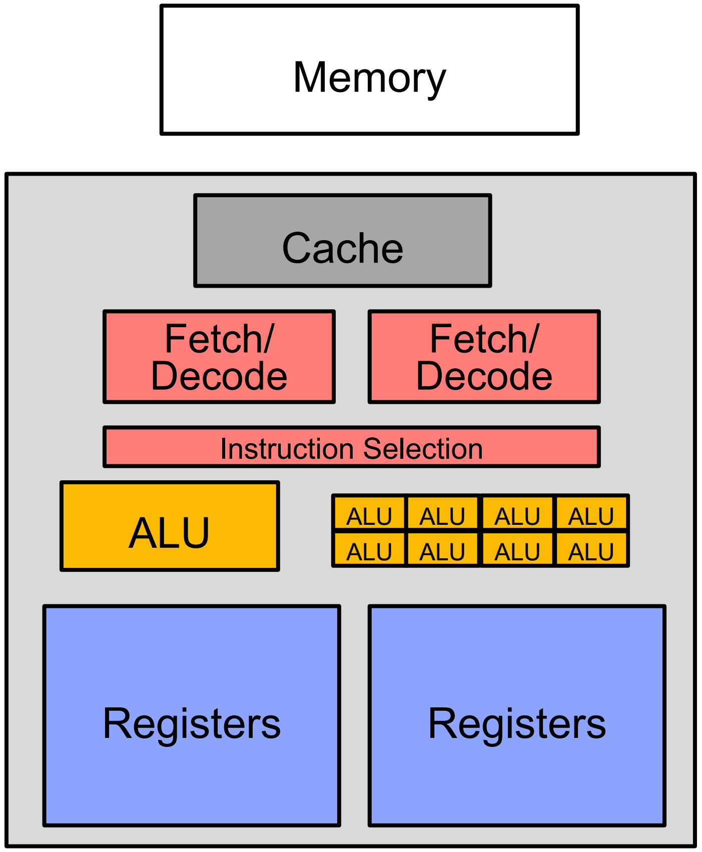
Single-core, **multi-thread**, **super-scalar, OoO, SIMD** processor


Memory
Cache
Fetch/Decode
Instruction Selection
ALU ALU ALU ALU
ALU ALU ALU ALU
Registers Registers Registers Registers
Registers Registers Registers Registers

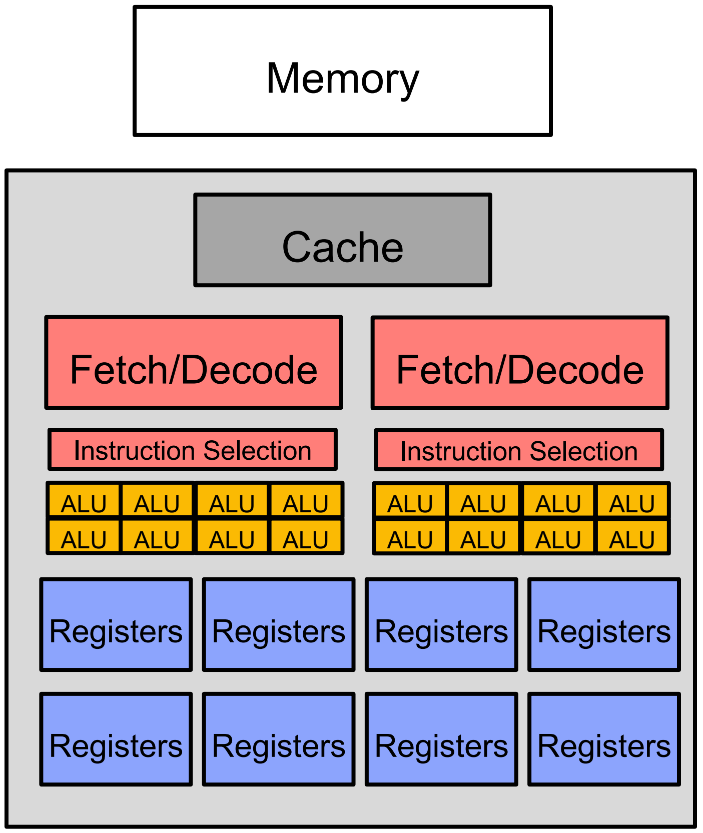
Single-core, **multi-thread**, **InO, SIMD processor**

GPU SIMT

- "Single instruction multiple thread"
  - Or multi-threaded SIMD/streaming multi-processor (SM)



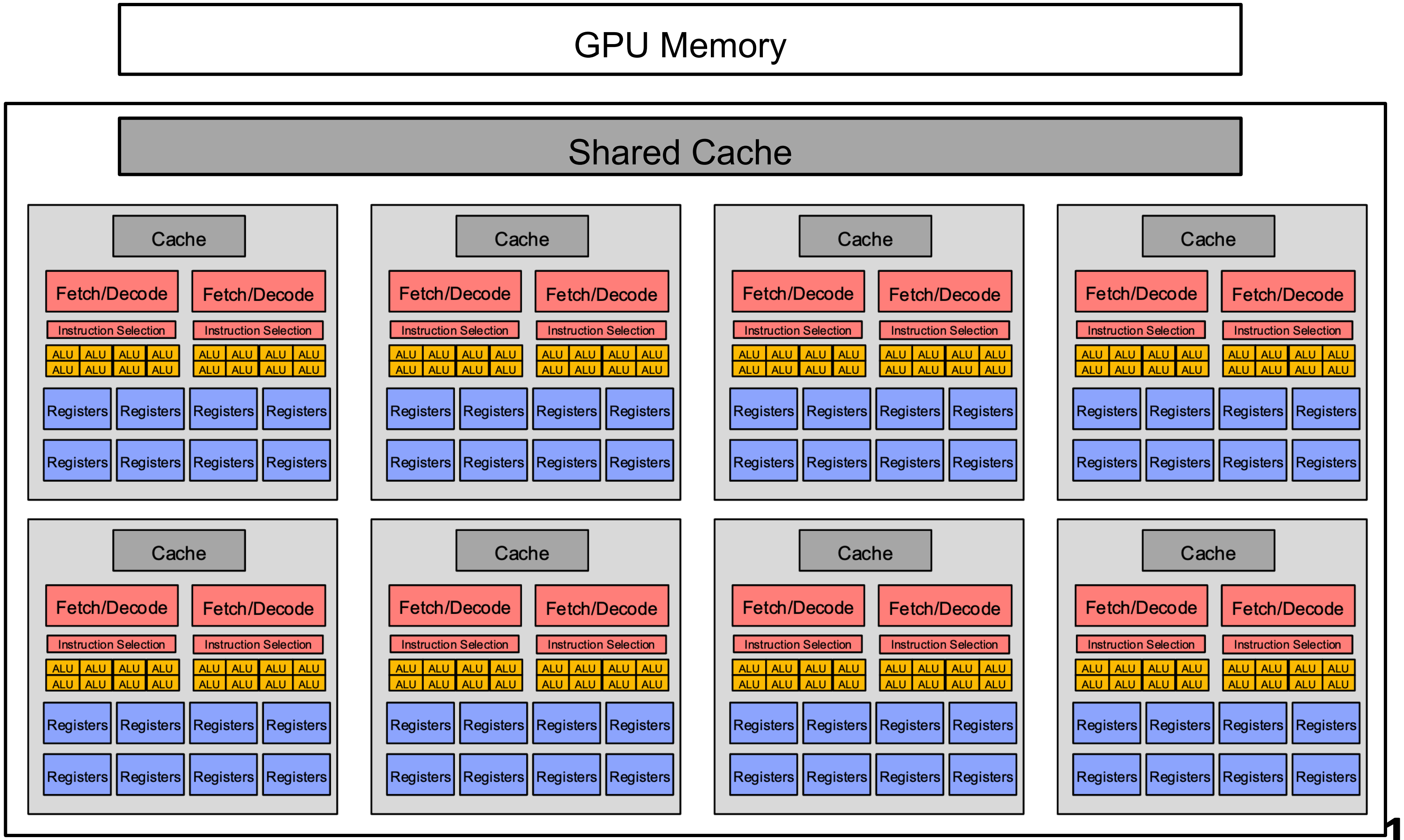
Single-core, **multi-thread**, **super-scalar, OoO, SIMD** processor



Single-core, **multi-thread, InO, SIMD processor**

GPU Architecture

- Many multi-threaded SIMD cores (streaming multi-processors)



Simplified CUDA Programming Model

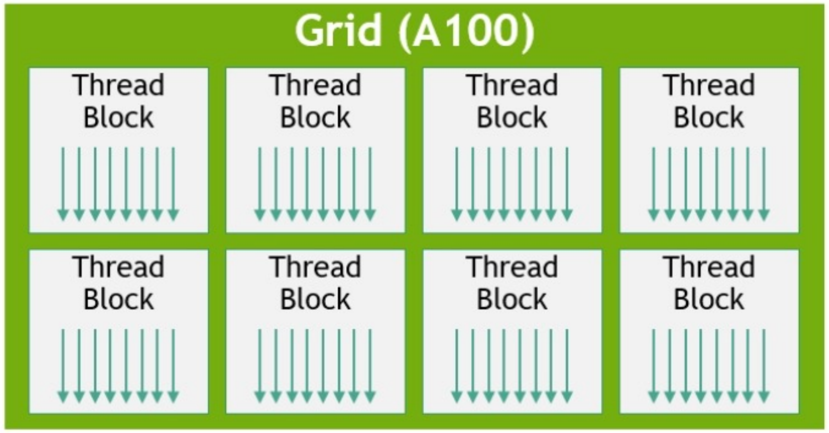
- Computation performed by a very large number of independent small scalar threads (*CUDA threads* or *microthreads*) grouped into *thread blocks.*

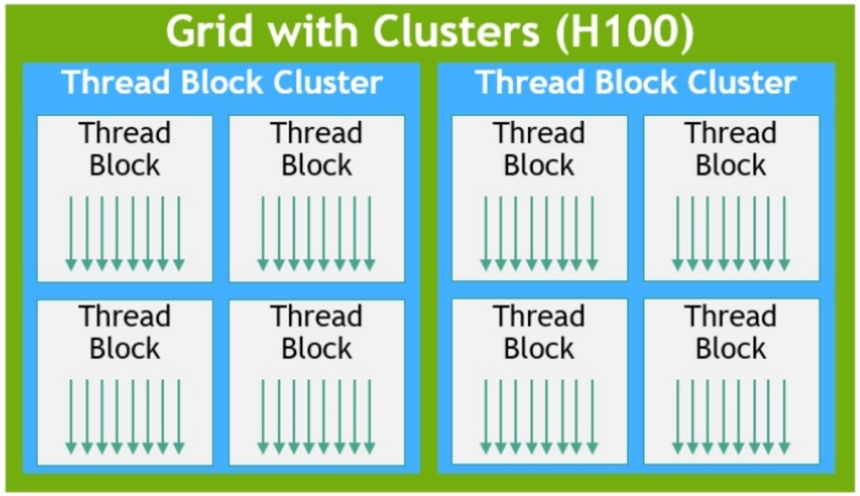
```
// C version of DAXPY loop.
void daxpy(int n, double a, double*x, double*y)
{ for (int i=0; i<n; i++)
      y[i] = a*x[i] + y[i]; }

// CUDA version.
__host__  // Piece run on host processor.
int nblocks = (n+255)/256; //256 CUDA threads/block
daxpy<<<nblocks,256>>>(n,2.0,x,y);

__device__  // Piece run on GP-GPU.
void daxpy(int n, double a, double*x, double*y)
{ int i = blockIdx.x*blockDim.x + threadId.x;
  if (i<n) y[i]=a*x[i]+y[i]; }
```

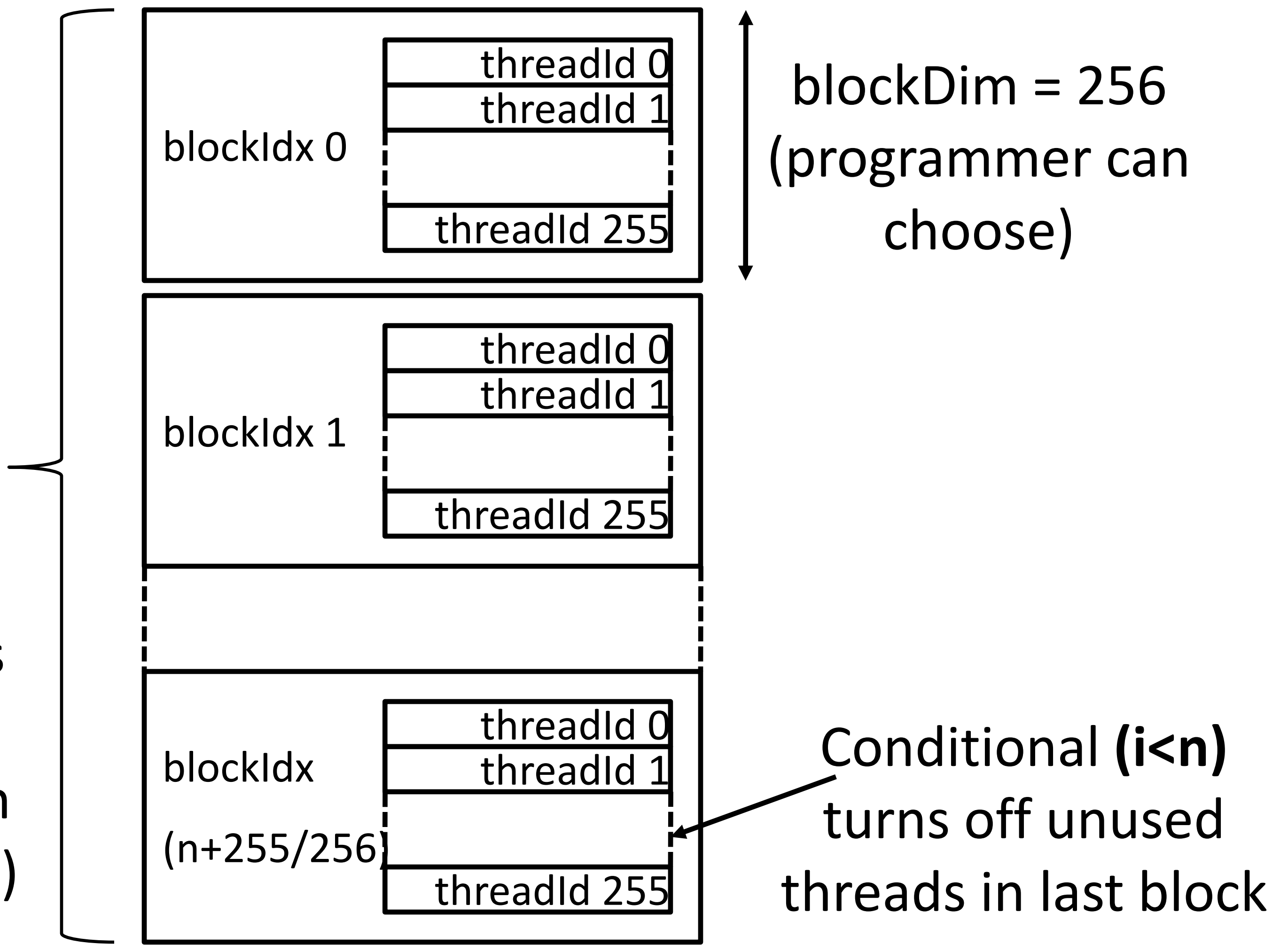
Programmer’s View of Execution





- https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/

Programmer's View of Execution

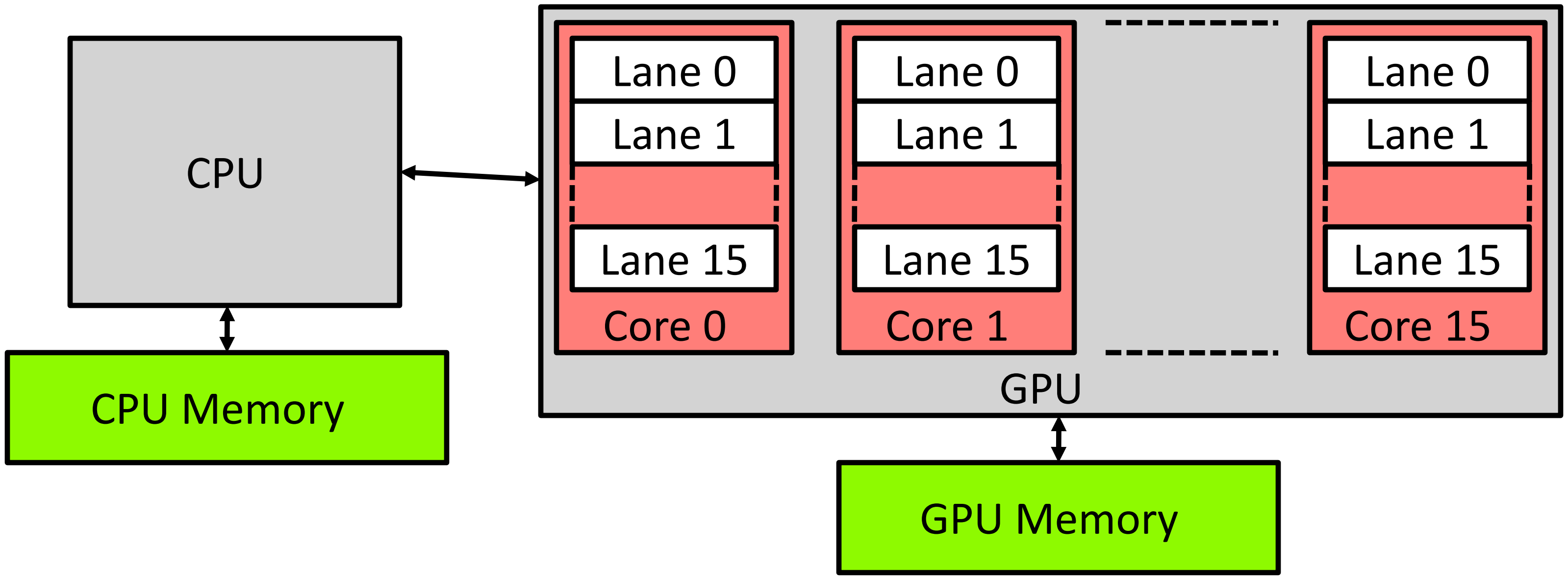


Create enough blocks to cover input vector

(NVIDIA calls this ensemble of blocks a *Grid*, can be 2-dimensional)

Conditional **(i<n)** turns off unused threads in last block

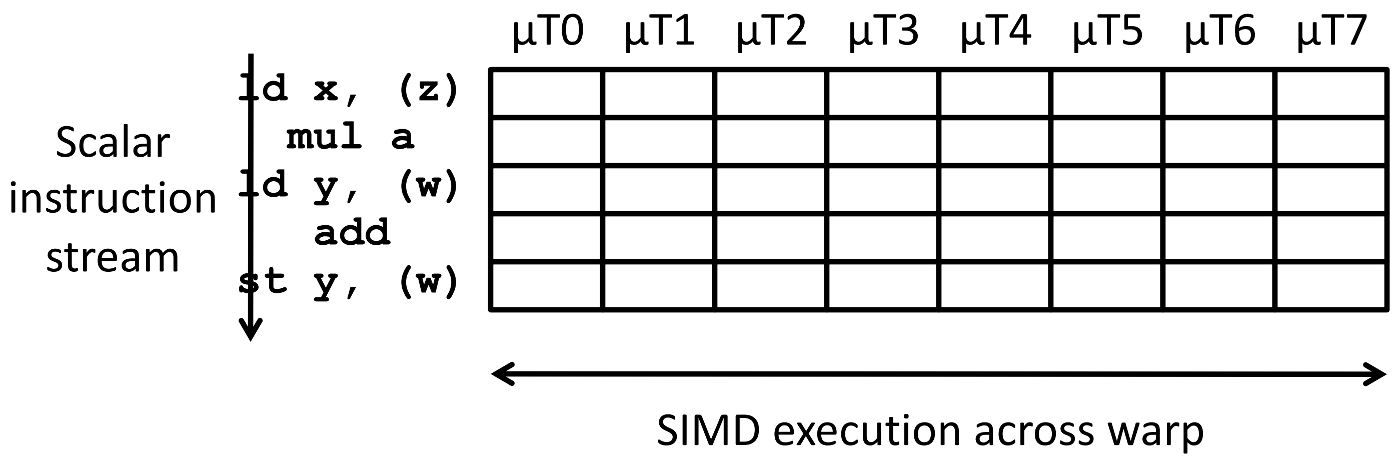
Hardware Execution Model



- GPU is built from multiple parallel cores, each core contains a multithreaded SIMD processor with multiple lanes but with no scalar processor
- CPU sends whole “grid” over to GPU, which distributes thread blocks among cores (each thread block executes on one core)
  - Programmer unaware of number of cores

"Single Instruction, Multiple Thread" (SIMT)

- GPUs use a SIMT model, where individual scalar instruction streams for each CUDA thread are grouped together for SIMD execution on hardware (NVIDIA groups 32 CUDA threads into a *warp*)



Implications of SIMT Model

- All “vector” loads and stores are scatter-gather, as individual µthreads perform scalar loads and stores
  - GPU adds hardware to dynamically coalesce individual µthread loads and stores to mimic vector loads and stores
- Every µthread has to perform stripmining calculations redundantly (“am I active?”) as there is no scalar processor equivalent

Multithreaded SIMD Processor

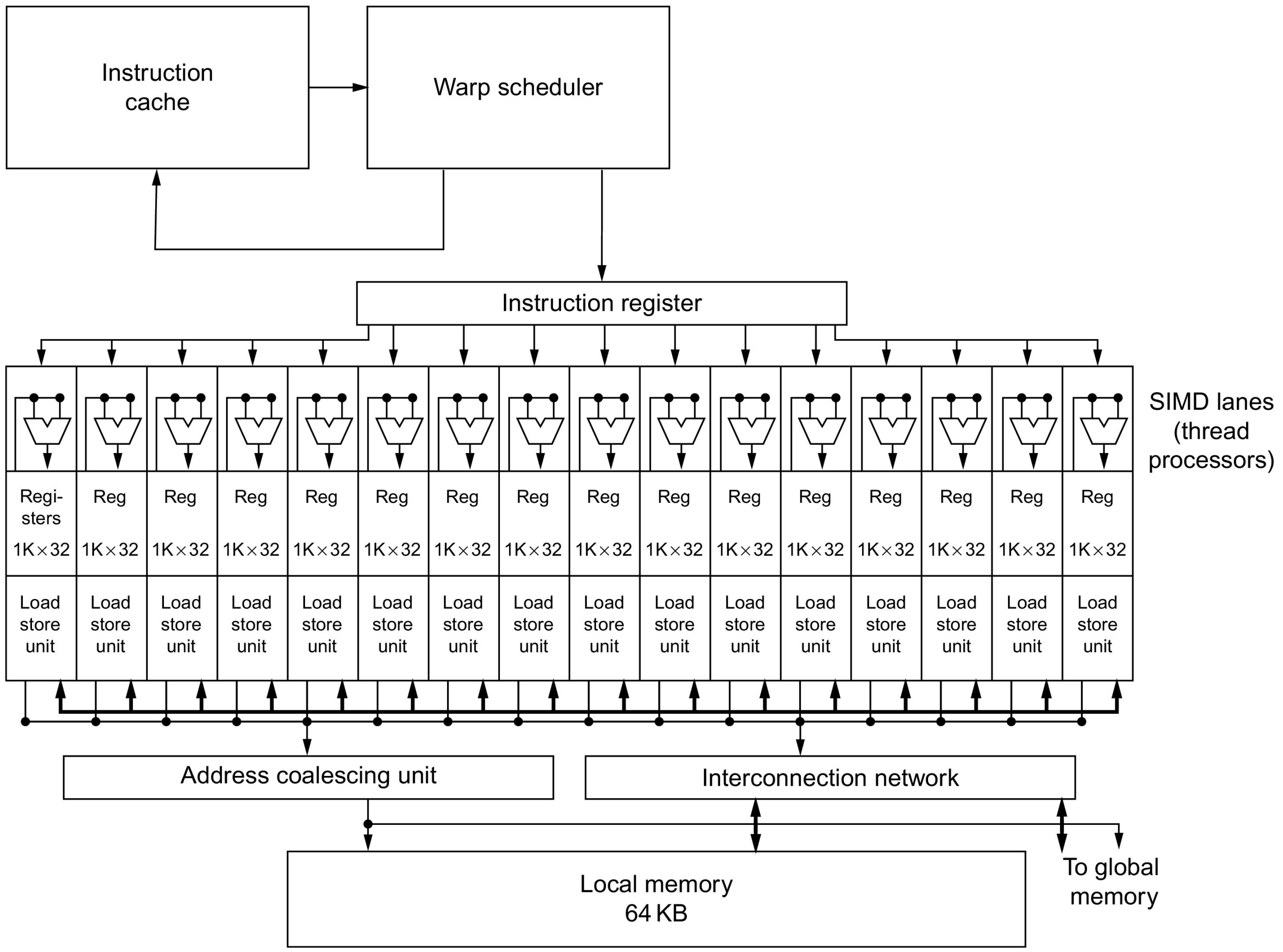
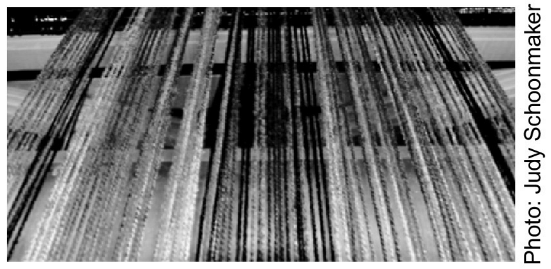
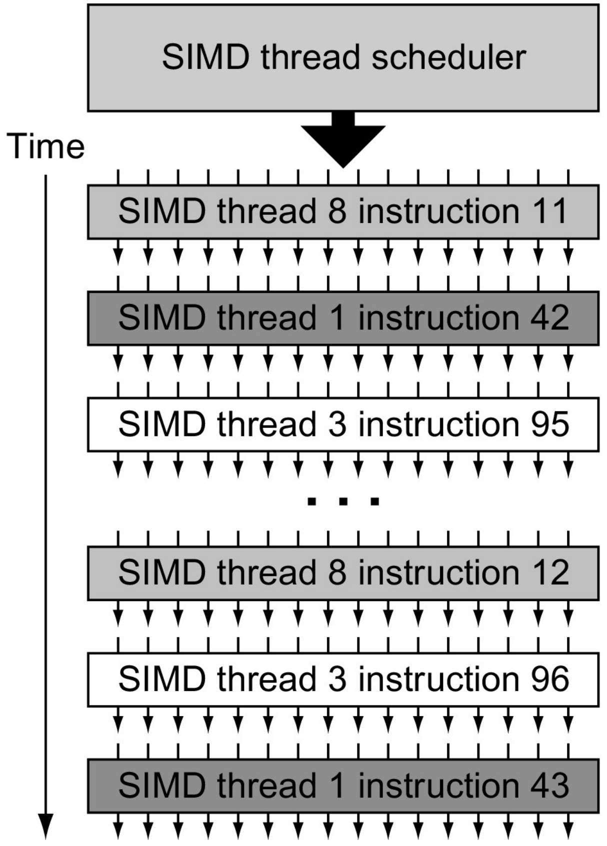


Figure 4.14 Simplified block diagram of a multithreaded SIMD Processor. It has 16 SIMD Lanes. The SIMD Thread Scheduler has, say, 64 independent threads of SIMD instructions that it schedules with a table of 64 program counters (PCs). Note that each lane has 1024 32-bit registers.

Warps are multithreaded on core




SIMD thread scheduler
Time
SIMD thread 8 instruction 11
SIMD thread 1 instruction 42
SIMD thread 3 instruction 95
. . .
SIMD thread 8 instruction 12
SIMD thread 3 instruction 96
SIMD thread 1 instruction 43

[Nvidia, 2010]

- Warp == SIMD Thread, i.e., a thread of SIMD instructions
- One warp of 32 µthreads is a single thread in the hardware
- Multiple warp threads are interleaved in execution on a single core to hide latencies (memory and functional unit)
- A single thread block can contain multiple warps (up to 512 µT max in CUDA), all mapped to single core
- Can have multiple blocks executing on one core

Vector vs Multithreaded SIMD

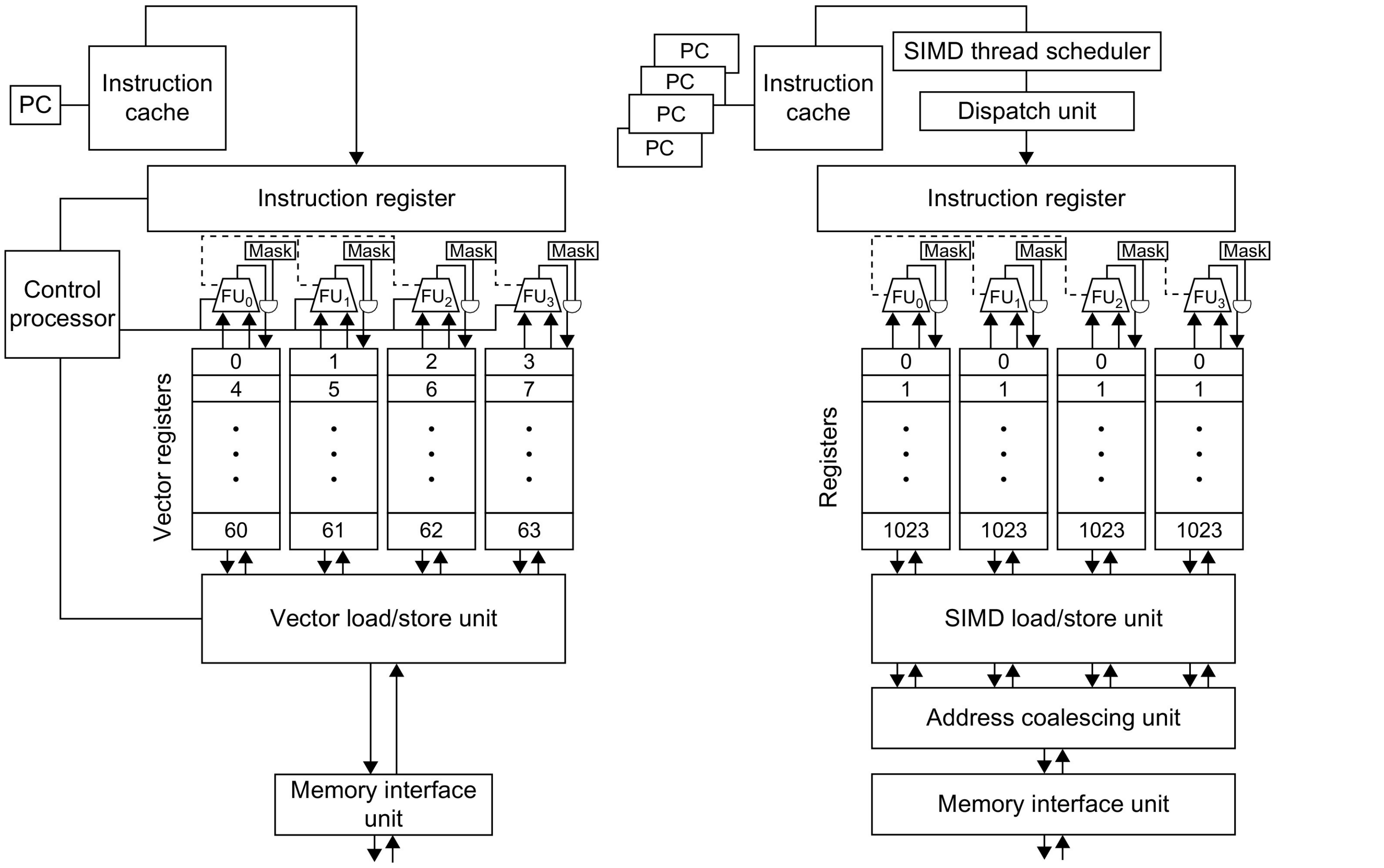


Figure 4.22 A vector processor with four lanes on the left and a multithreaded SIMD Processor of a GPU with four SIMD Lanes on the right. (GPUs typically have 16 or 32 SIMD Lanes.) The Control Processor supplies scalar operands for scalar-vector operations, increments addressing for unit and nonunit stride accesses to memory, and performs other accounting-type operations. Peak memory performance occurs only in a GPU when the Address Coalescing Unit can discover localized addressing. Similarly, peak computational performance occurs when all internal mask bits are set identically. Note that the SIMD Processor has one PC per SIMD Thread to help with multithreading.

CS152 Administrivia

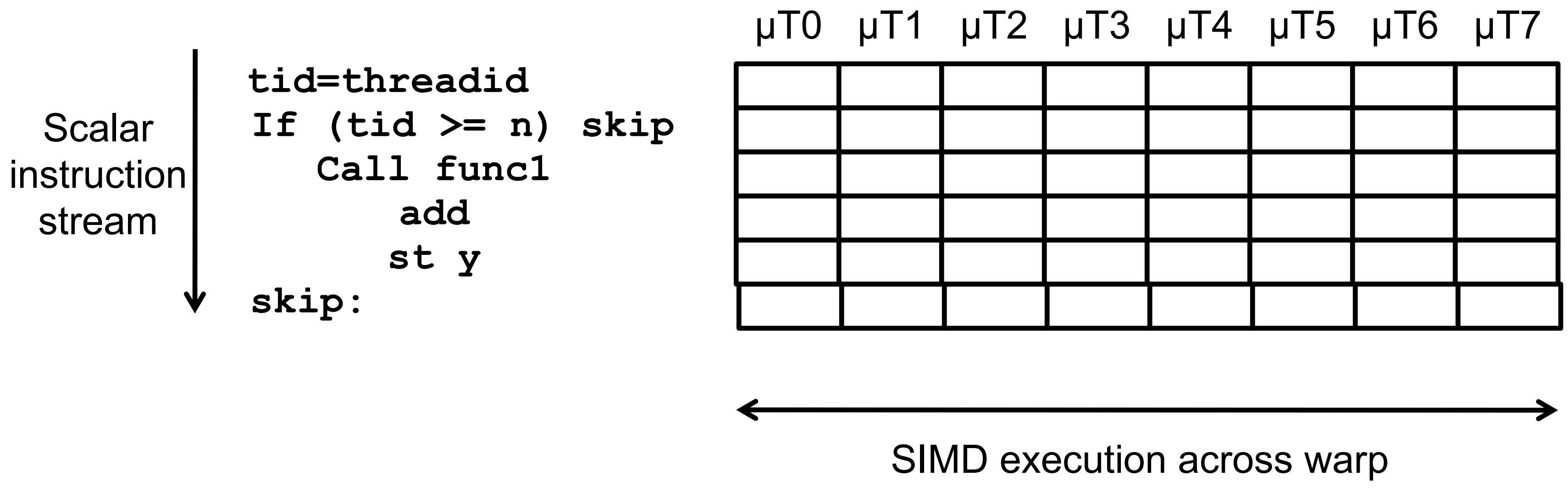
- HW 4 out this week
- Lab 3 due 3/23
- Midterm feedback

CS252 Administrivia

- Readings on OoO this Wednesday

Conditionals in SIMT model

- Simple if-then-else are compiled into predicated execution, equivalent to vector masking
- More complex control flow compiled into branches
- How to execute a vector of branches? Vector function calls?



```
tid=threadid
If (tid >= n) skip
    Call func1
          add
         st y
skip:
```

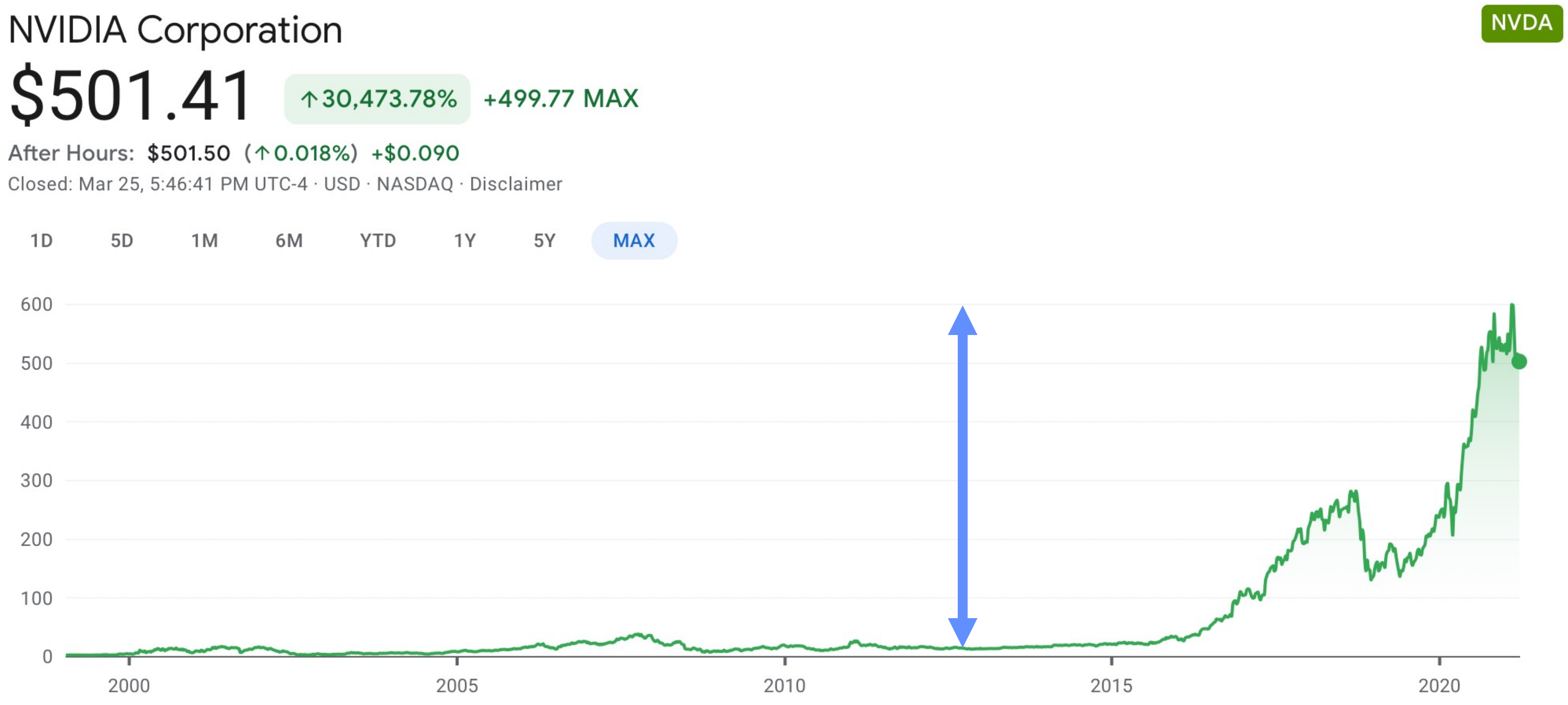
Branch Divergence

- Hardware tracks which µthreads take or don’t take branch
- If all go the same way, then keep going in SIMD fashion
- If not, create mask vector indicating taken/not-taken
- Keep executing not-taken path under mask, push taken branch PC+mask onto a hardware stack and execute later
- When can execution of µthreads in warp reconverge?

SIMT

- Illusion of many independent threads
- But for efficiency, programmer must try and keep µthreads aligned in a SIMD fashion
  - Try and do unit-stride loads and store so memory coalescing kicks in
  - Avoid branch divergence so most instruction slots execute useful work and are not masked off

Important of Machine Learning for GPUs


NVIDIA Corporation

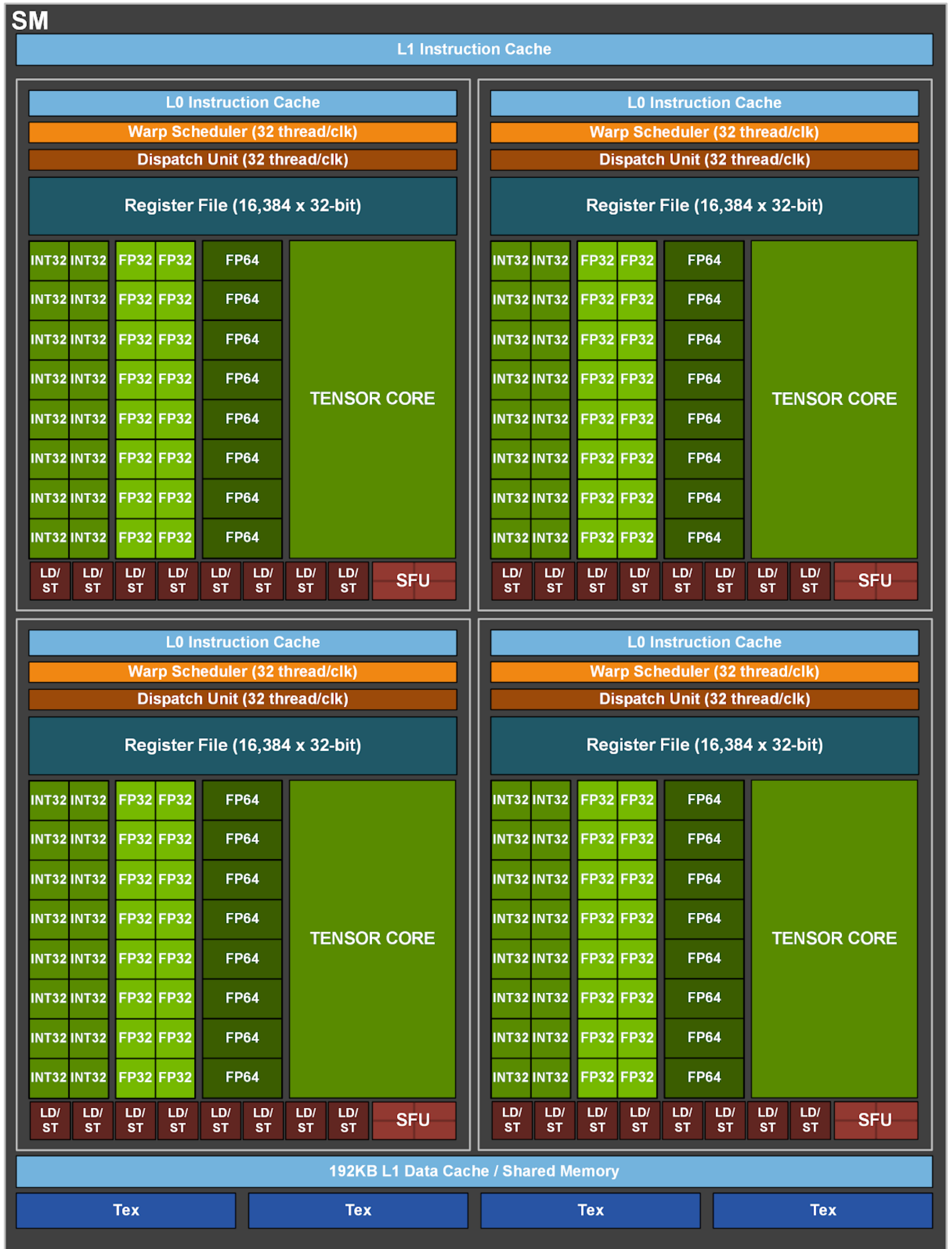
$501.41 ↑30,473.78% +499.77 MAX

After Hours: $501.50 (↑0.018%) +$0.090

Closed: Mar 25, 5:46:41 PM UTC-4 · USD · NASDAQ · Disclaimer

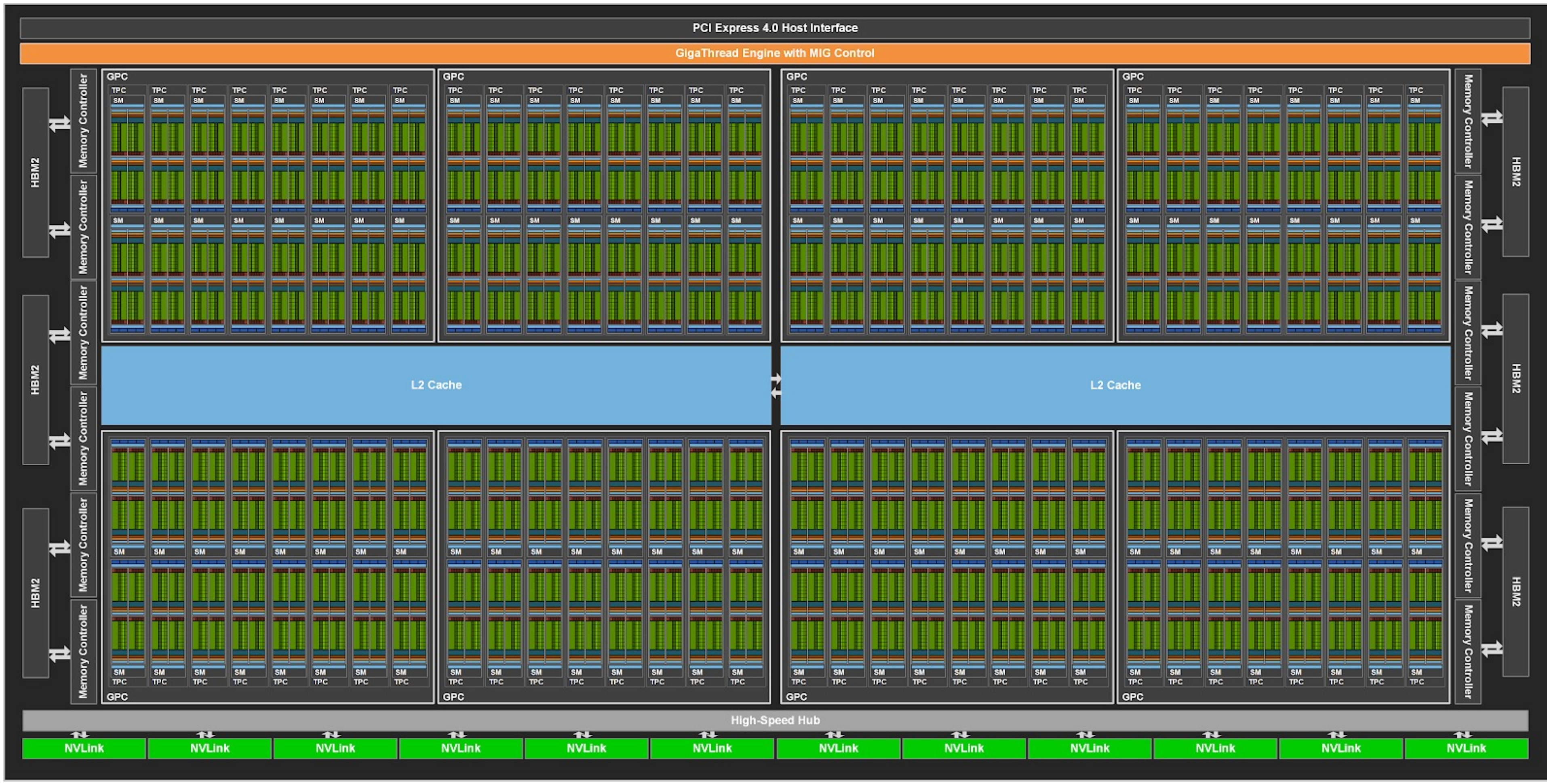
NVIDIA stock price 40x in 9 years (since deep learning became important)

Nvidia Ampere A100 GPU


SM
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
INT32
FP32
FP64
TENSOR CORE
LD/ ST
SFU
192KB L1 Data Cache / Shared Memory
Tex

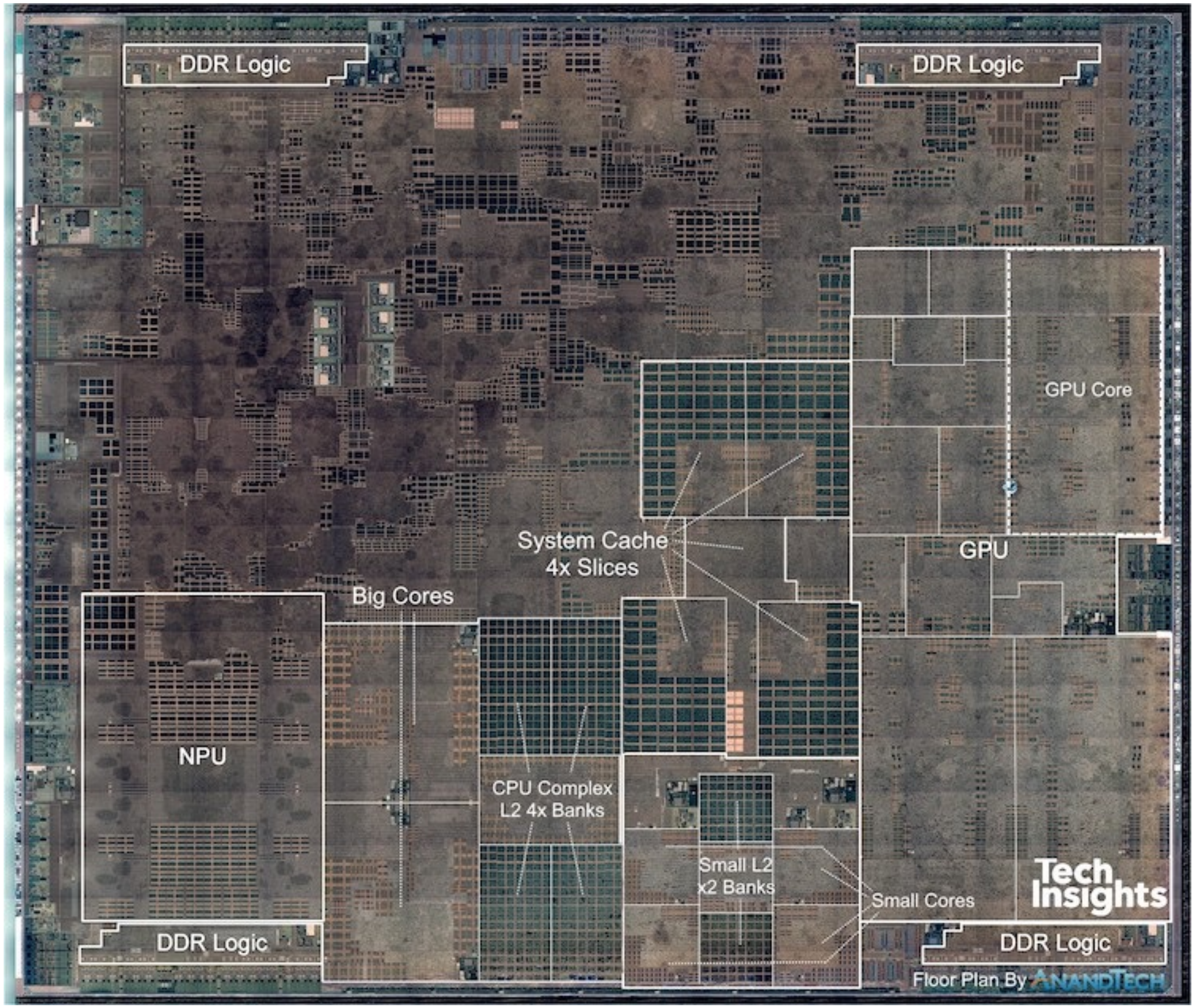
[Nvidia, 2022]

Nvidia Ampere A100 GPU


PCI Express 4.0 Host Interface
GigaThread Engine with MIG Control
GPC
TPC
SM
L2 Cache
Memory Controller
HBM2
High-Speed Hub
NVLink

[Nvidia, 2022]

Apple A12 Processor (2018)



- 83.27mm^2
- 7nm technology

[Source: Tech Insights, AnandTech]

Acknowledgements

- This course is partly inspired by previous MIT 6.823 and Berkeley CS252 computer architecture courses created by my collaborators and colleagues:
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)