

CSC 631: High-Performance Computer Architecture

Spring 2022

Lecture 10: GPUs

Types of Parallelism

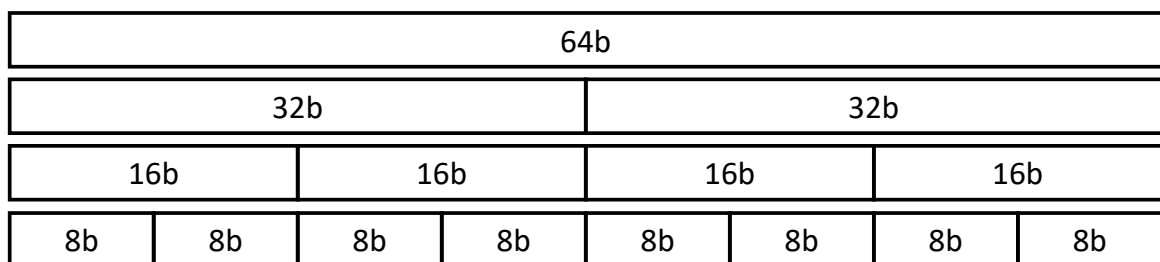
- **Instruction-Level Parallelism (ILP)**
 - Execute independent instructions from one instruction stream in parallel (pipelining, superscalar, VLIW)
- **Thread-Level Parallelism (TLP)**
 - Execute independent instruction streams in parallel (multithreading, multiple cores)
- **Data-Level Parallelism (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

Resurgence of DLP

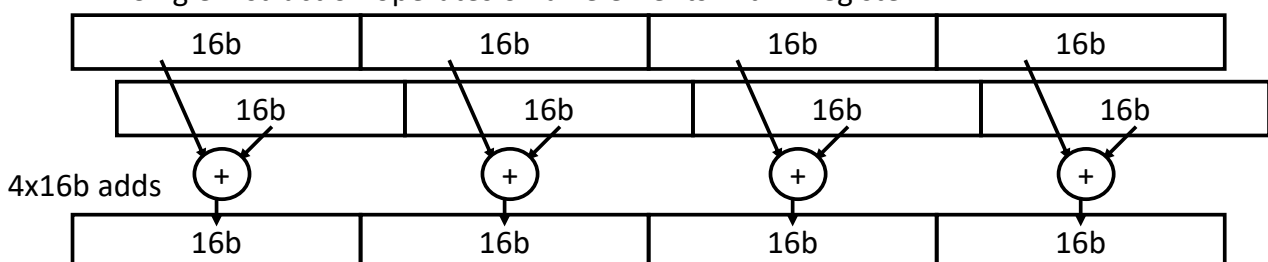
- Convergence of application demands and technology constraints drives architecture choice
- New applications, such as graphics, machine vision, speech recognition, machine learning, etc. all require large numerical computations that are often trivially data parallel
- SIMD-based architectures (vector-SIMD, subword-SIMD, SIMT/GPUs) are most efficient way to execute these algorithms

3

Packed SIMD Extensions



- Short vectors added to existing microprocessors ISAs, for multimedia
- Use existing 64-bit registers split into 2x32b or 4x16b or 8x8b
 - Lincoln Labs TX-2 from 1957 had 36b datapath split into 2x18b or 4x9b
 - Newer designs have wider registers
 - 128b for PowerPC AltiVec, Intel SSE2/3/4
 - 256b for Intel AVX
- Single instruction operates on all elements within register



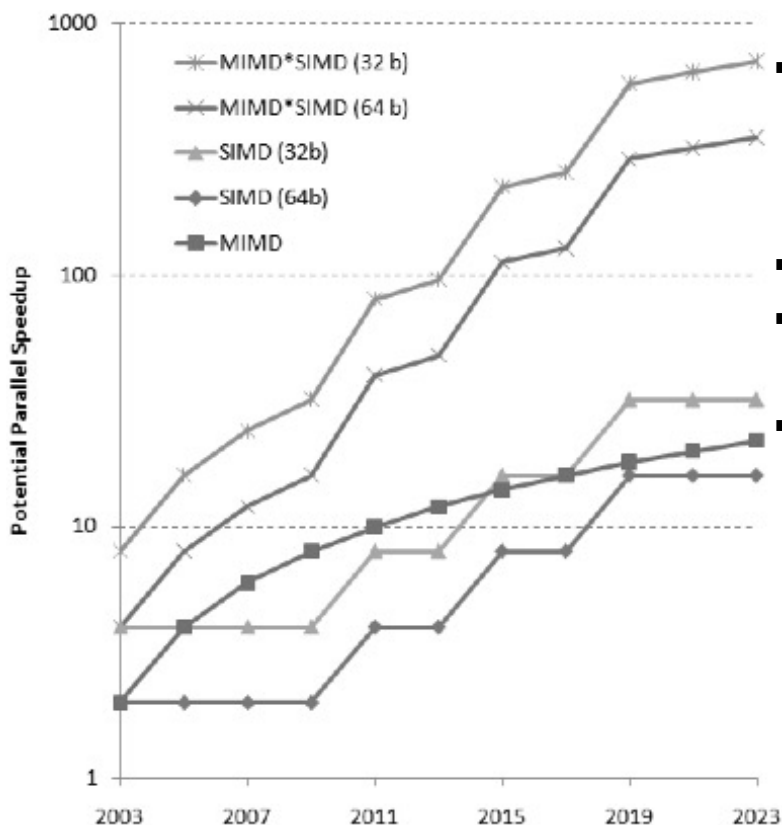
4

Multimedia Extensions versus Vectors

- Limited instruction set
 - no vector length control
 - no strided load/store or scatter/gather
 - unit-stride loads must be naturally aligned to whole register width (e.g., 64 or 128-bit)
- Limited vector register length
 - requires superscalar issue to keep multiply/add/load units busy
 - loop unrolling to hide latencies increases register pressure
- Trend towards fuller vector support in microprocessors
 - Better support for misaligned memory accesses
 - Support of double-precision (64-bit floating-point)
 - New Intel AVX spec (announced April 2008), 256b vector registers (expandable up to 1024b) , adding scatter/gather
 - New ARM SVE/MVE vector ISA closer to traditional vector designs

5

DLP important for conventional CPUs



- Prediction for x86 processors, from Hennessy & Patterson, 5th edition
 - *Note: Educated guess, not Intel product plans!*
- TLP: 2+ cores / 2 years
- DLP: 2x width / 4 years
- DLP will account for more mainstream parallelism growth than TLP in next decade.
 - SIMD –single-instruction multiple-data (DLP)
 - MIMD- multiple-instruction multiple-data (TLP)

6

Graphical Processing Units

- Basic idea:
 - Heterogeneous execution model
 - CPU is the host, GPU is the device
 - Develop a C-like programming language for GPU
 - Unify all forms of GPU parallelism as CUDA thread
 - Programming model is “Single Instruction Multiple Thread”

7

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

8

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 only considers GPU execution for computational kernels, not graphics
 - Would need whole other course to describe graphics processing

9

Threads and Blocks

- A thread is associated with each data element
- Threads are organized into blocks
- Blocks are organized into a grid
- GPU hardware handles thread management, not applications or OS

10

NVIDIA GPU Architecture

- Similarities to vector machines:
 - Works well with data-level parallel problems
 - Scatter-gather transfers
 - Mask registers
 - Large register files
- Differences:
 - No scalar processor
 - Uses multithreading to hide memory latency
 - Has many functional units, as opposed to a few deeply pipelined units like a vector processor

11

Example

- Code that works over all elements is the grid
- Thread blocks break this down into manageable sizes
 - 512 threads per block
- SIMD instruction executes 32 elements at a time
- Thus grid size = 16 blocks
- Block is analogous to a strip-mined vector loop with vector length of 32
- Block is assigned to a multithreaded SIMD processor by the thread block scheduler
- Current-generation GPUs have 7-15 multithreaded SIMD processors

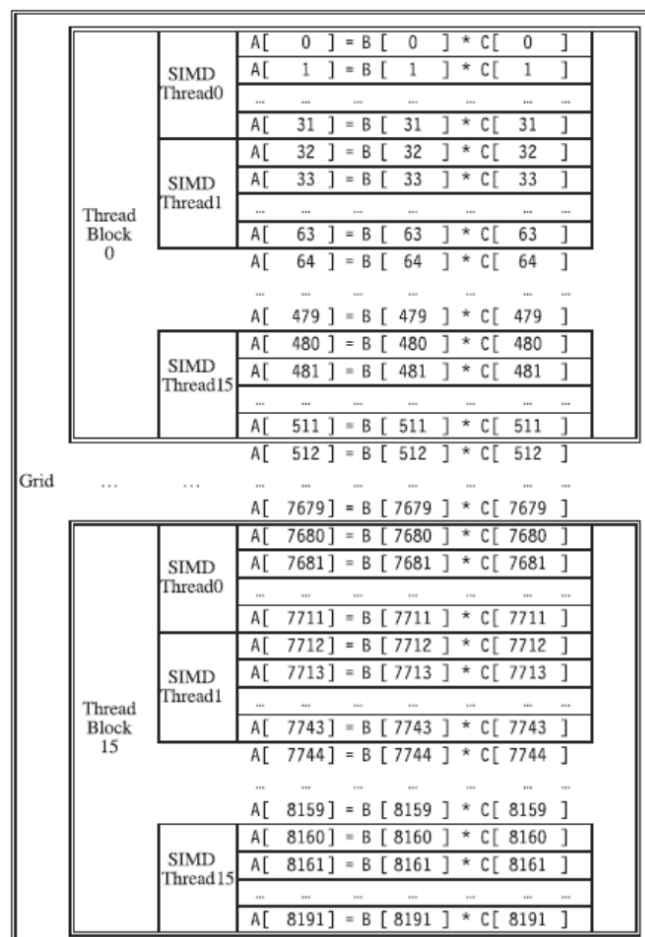
12

Terminology

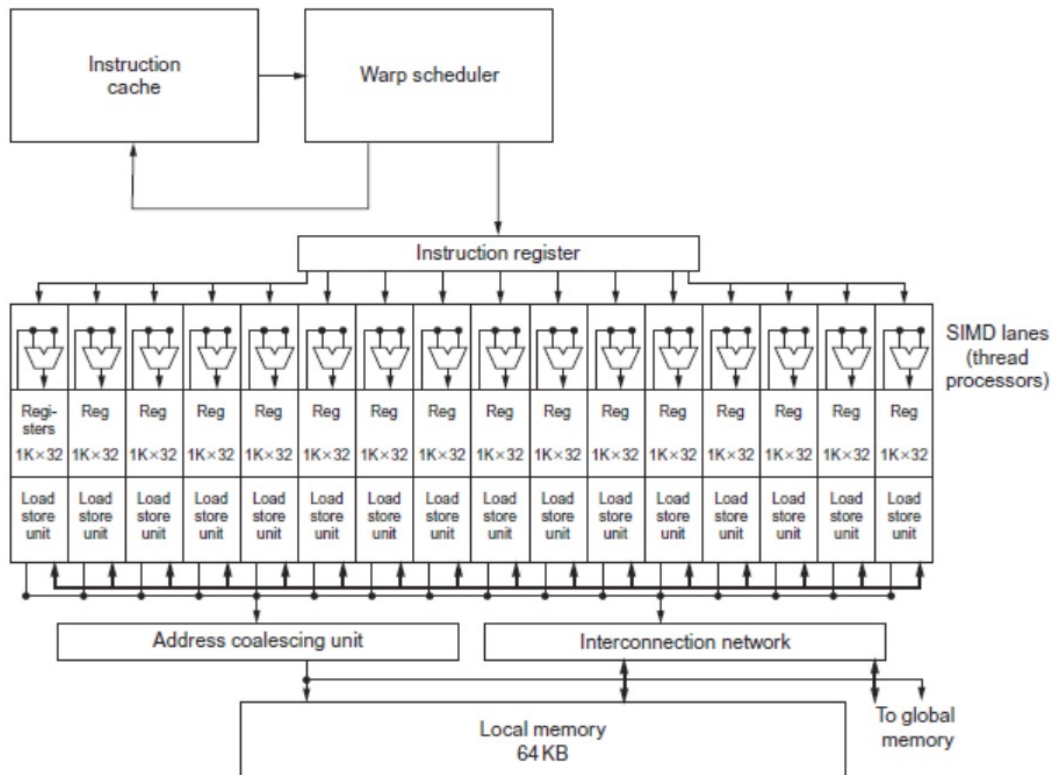
- Each thread is limited to 64 registers
- Groups of 32 threads combined into a SIMD thread or “warp”
 - Mapped to 16 physical lanes
- Up to 32 warps are scheduled on a single SIMD processor
 - Each warp has its own PC
 - Thread scheduler uses scoreboard to dispatch warps
 - By definition, no data dependencies between warps
 - Dispatch warps into pipeline, hide memory latency
- Thread block scheduler schedules blocks to SIMD processors
- Within each SIMD processor:
 - 32 SIMD lanes
 - Wide and shallow compared to vector processors

13

Example



GPU Organization



Copyright © 2019, Elsevier Inc. All rights Reserved

NVIDIA GPU Memory Structures

- Each SIMD Lane has private section of off-chip DRAM
 - “Private memory”
 - Contains stack frame, spilling registers, and private variables
- Each multithreaded SIMD processor also has local memory
 - Shared by SIMD lanes / threads within a block
- Memory shared by SIMD processors is GPU Memory
 - Host can read and write GPU memory

Copyright © 2019, Elsevier Inc. All rights Reserved

Simplified CUDA Programming Model

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

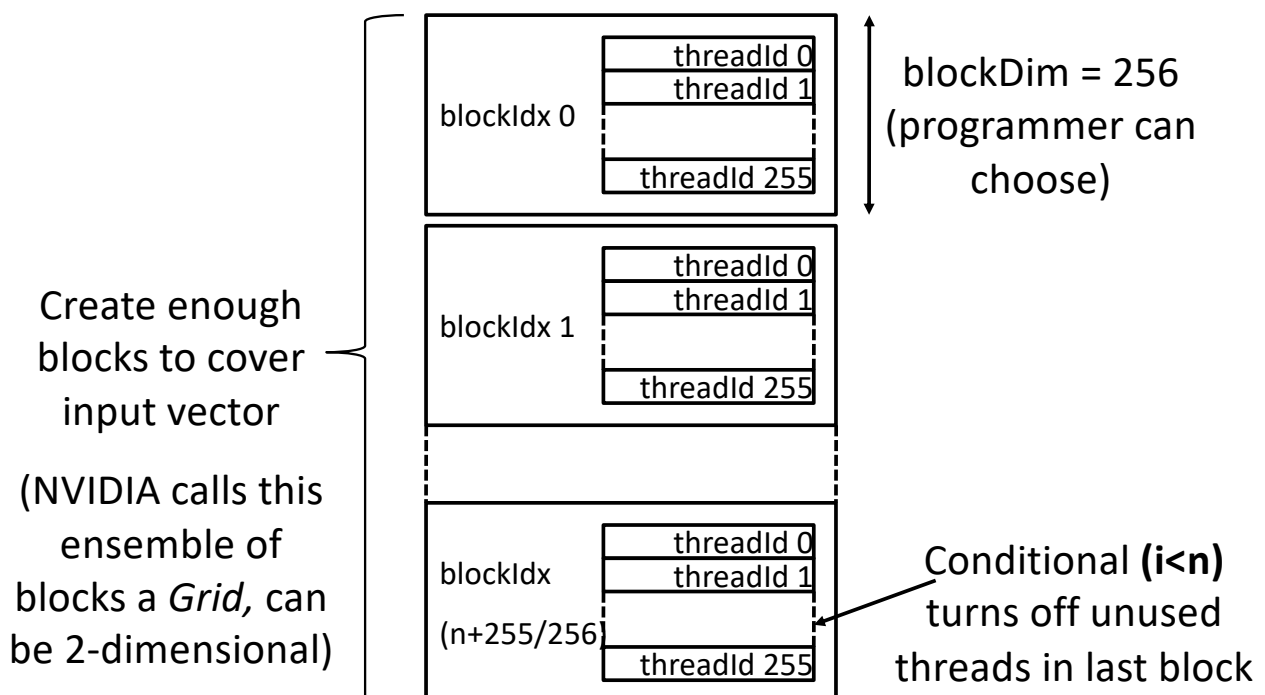
```
// C version of DAXPY loop.
void daxpy(int n, double a, double*x, double*y)
{
    for (int i=0; i<n; i++)
        y[i] = a*x[i] + y[i];
}

// 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 + threadIdx.x;
    if (i<n)
        y[i]=a*x[i]+y[i];
}
```

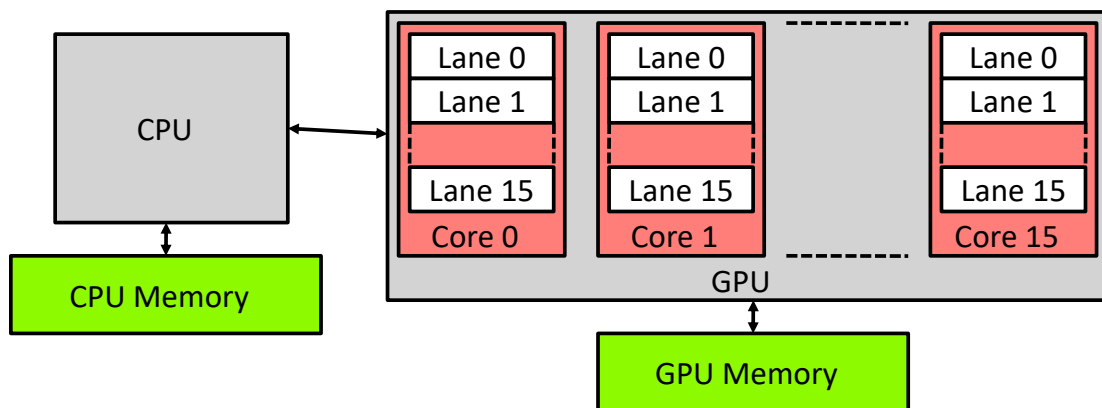
17

Programmer's View of Execution



18

Hardware Execution Model

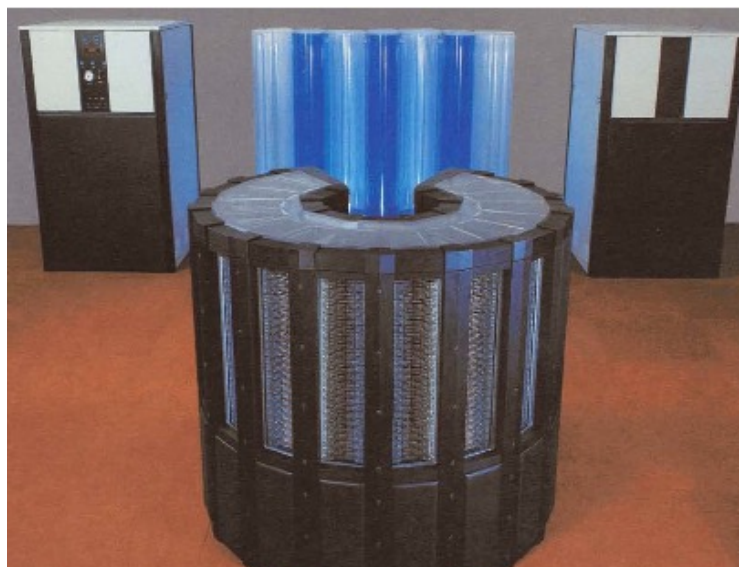
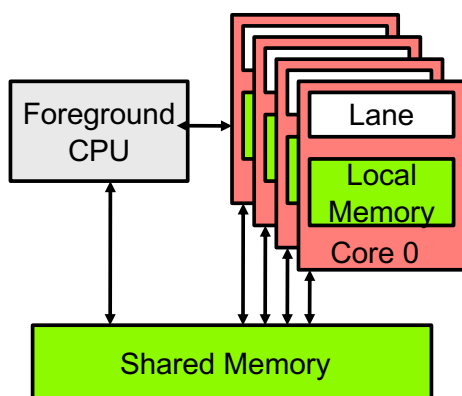


- GPU is built from multiple parallel cores, each core contains a multithreaded SIMD processor with multiple lanes but with no scalar processor
 - some adding “scalar coprocessors” now
- CPU sends whole “grid” over to GPU, which distributes thread blocks among cores (each thread block executes on one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

19

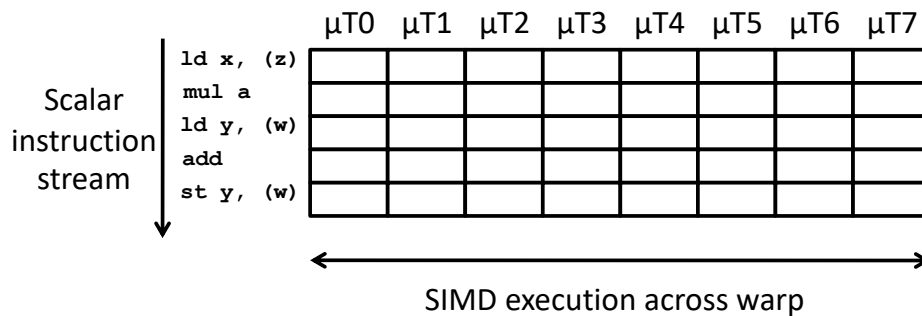
Historical Retrospective, Cray-2 (1985)

- 243MHz ECL logic
- 2GB DRAM main memory (128 banks of 16MB each)
 - Bank busy time 57 clocks!
- Local memory of 128KB/core
- 1 foreground + 4 background vector processors



“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*)



21

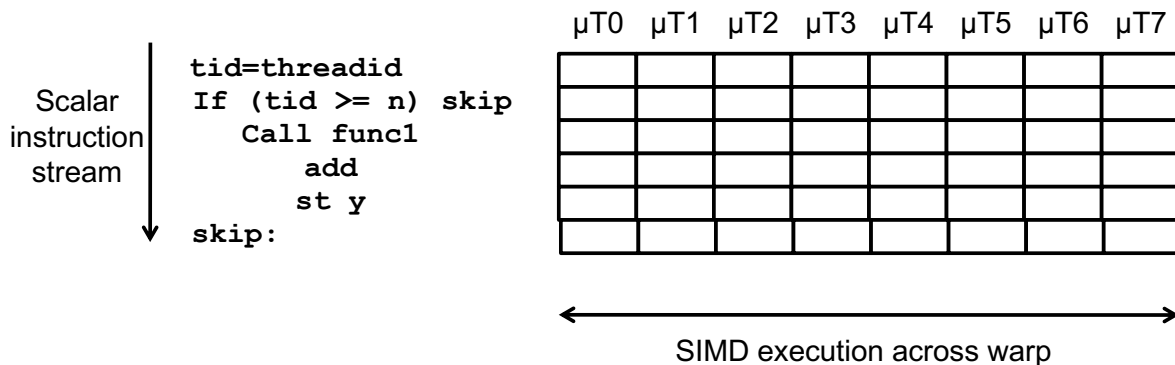
Implications of SIMT Model

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

22

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?



23

Branch Divergence

- Hardware tracks which μ threads 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 μ threads in warp reconverge?

24

NVIDIA Instruction Set Arch.

- ISA is an abstraction of the hardware instruction set

- “Parallel Thread Execution (PTX)”
 - opcode.type d,a,b,c;
- Uses virtual registers
- Translation to machine code is performed in software
- Example:

```
shl.s32  R8, blockIdx, 9      ; Thread Block ID * Block size (512 or 29)
add.s32  R8, R8, threadIdx    ; R8 = i = my CUDA thread ID
ld.global.f64 RD0, [X+R8]     ; RD0 = X[i]
ld.global.f64 RD2, [Y+R8]     ; RD2 = Y[i]
mul.f64  RD0, RD0, RD4        ; Product in RD0 = RD0 * RD4 (scalar a)
add.f64  RD0, RD0, RD2        ; Sum in RD0 = RD0 + RD2 (Y[i])
st.global.f64 [Y+R8], RD0     ; Y[i] = sum (X[i]*a + Y[i])
```

25

Conditional Branching

- Like vector architectures, GPU branch hardware uses internal masks
- Also uses
 - Branch synchronization stack
 - Entries consist of masks for each SIMD lane
 - I.e. which threads commit their results (all threads execute)
 - Instruction markers to manage when a branch diverges into multiple execution paths
 - Push on divergent branch
 - ...and when paths converge
 - Act as barriers
 - Pops stack
- Per-thread-lane 1-bit predicate register, specified by programmer

26

Example

```

if (X[i] != 0)
    X[i] = X[i] - Y[i];
else X[i] = Z[i];

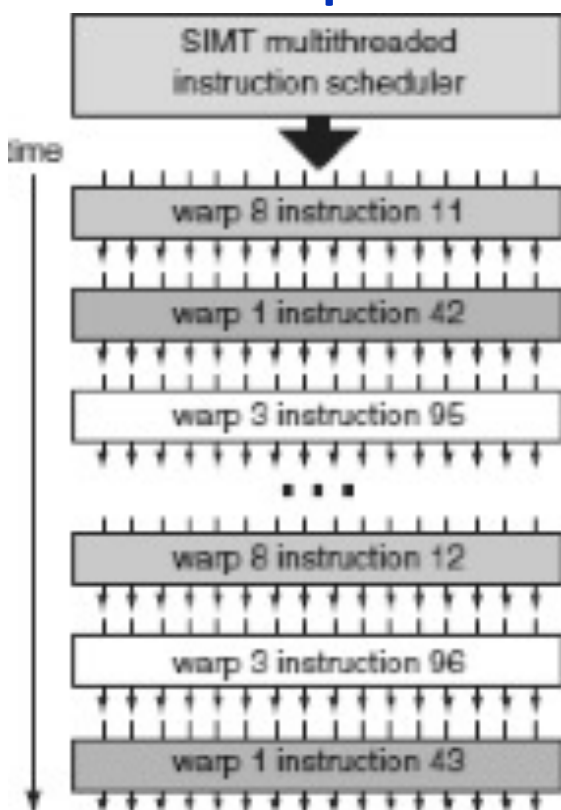
ld.global.f64    RD0, [X+R8]          ; RD0 = X[i]
setp.neq.s32     P1, RD0, #0          ; P1 is predicate register 1
@!P1, bra        ELSE1, *Push         ; Push old mask, set new
mask bits

                                ; if P1 false, go to ELSE1
ld.global.f64    RD2, [Y+R8]          ; RD2 = Y[i]
sub.f64          RD0, RD0, RD2         ; Difference in RD0
st.global.f64    [X+R8], RD0          ; X[i] = RD0
@P1, bra         ENDIF1, *Comp        ; complement mask bits
                                ; if P1 true, go to ENDIF1
ELSE1:           ld.global.f64 RD0, [Z+R8] ; RD0 = Z[i]
                  st.global.f64 [X+R8], RD0 ; X[i] = RD0
ENDIF1: <next instruction>, *Pop      ; pop to restore old mask

```

27

Warps are multithreaded on core

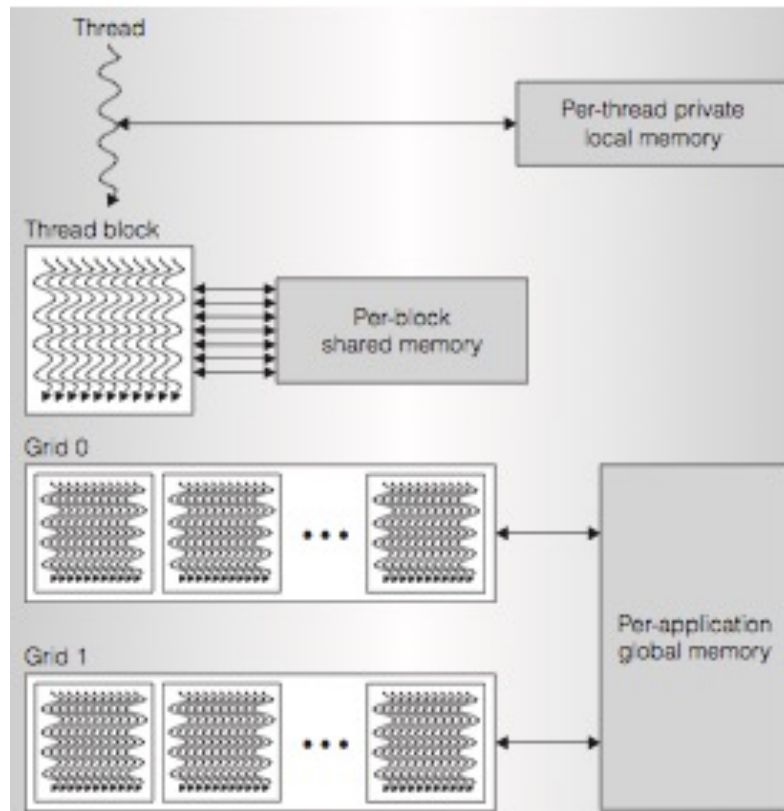


- One warp of 32 μ threads is a single thread in the hardware
- Multiple warp threads are interleaved in execution on a single core to hide latencies (memory and functional unit)
- A single thread block can contain multiple warps (up to 512 μ T max in CUDA), all mapped to single core
- Can have multiple blocks executing on one core

[Nvidia, 2010]

28

GPU Memory Hierarchy



[Nvidia, 2010]

29

SIMT

- Illusion of many independent threads
- But for efficiency, programmer must try and keep μ threads 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

30

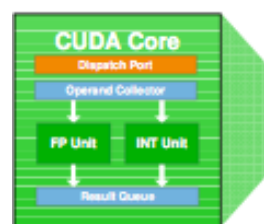
Nvidia Fermi GF100 GPU

[Nvidia,
2010]



31

Fermi “Streaming Multiprocessor” Core



32

Pascal Architecture Innovations

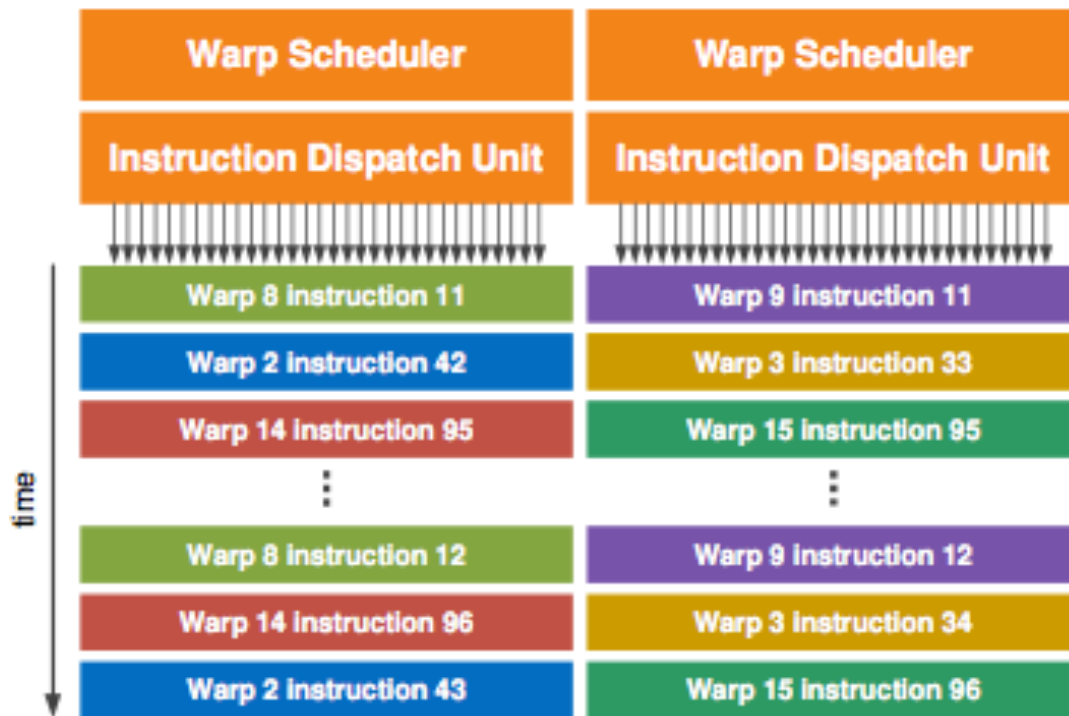
- Each SIMD processor has
 - Two or four SIMD thread schedulers, two instruction dispatch units
 - 16 SIMD lanes (SIMD width=32, chime=2 cycles), 16 load-store units, 4 special function units
 - Two threads of SIMD instructions are scheduled every two clock cycles
- Fast single-, double-, and half-precision
- High Bandwidth Memory 2 (HBM2) at 732 GB/s
- NVLink between multiple GPUs (20 GB/s in each direction)
- Unified virtual memory and paging support

Copyright © 2019, Elsevier Inc. All rights Reserved

NVIDIA Pascal Multithreaded GPU Core



Fermi Dual-Issue Warp Scheduler



35

Important of Machine Learning for GPUs

NVIDIA Corporation

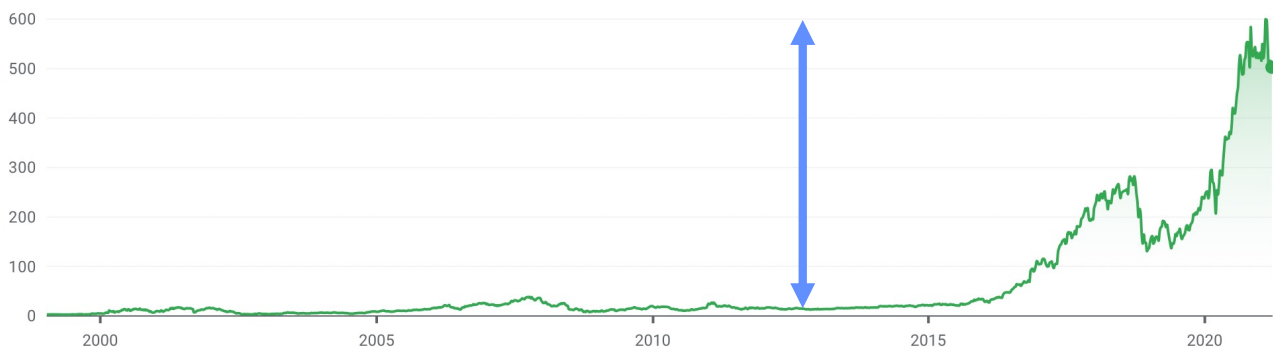
NVDA

\$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

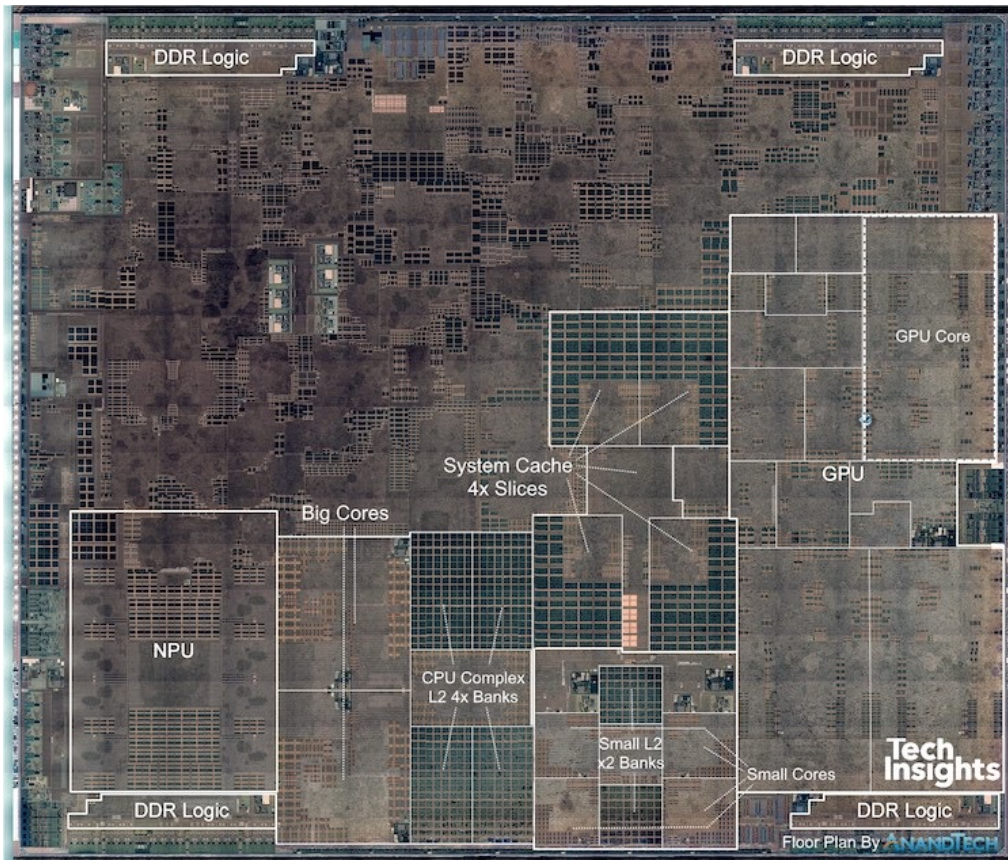
1D 5D 1M 6M YTD 1Y 5Y **MAX**



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

36

Apple A12 Processor (2018)



- 83.27mm²
- 7nm technology

[Source: Tech Insights, AnandTech]