

CS 380 - GPU and GPGPU Programming

Lecture 8: GPU Architecture, Pt. 6

Markus Hadwiger, KAUST

Reading Assignment #4 (until Sep 26)



Read (required):

- Get an overview of NVIDIA Ampere (GA102) white paper:

<https://www.nvidia.com/content/PDF/nvidia-ampere-ga-102-gpu-architecture-whitepaper-v2.pdf>

- Get an overview of NVIDIA Ampere (A100) Tensor Core GPU white paper:

<https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf>

- Get an overview of NVIDIA Hopper (H100) Tensor Core GPU white paper:

<https://resources.nvidia.com/en-us-tensor-core>

Read (optional):

- Look at the “Tuning Guides“ for different architectures in the CUDA SDK
- PTX Instruction Set Architecture (8.5): <https://docs.nvidia.com/cuda/parallel-thread-execution/>
Read Chapters 1 – 3; get an overview of Chapter 9;
browse through the other chapters to get a feeling for what PTX looks like
- CUDA SASS ISA (12.6), Chap. 6: https://docs.nvidia.com/cuda/pdf/CUDA_Binary_Utils.pdf



Next Lectures

Lecture 9: Thu, Sep 19

no lecture on Sep 23 !

Lecture 10: Thu, Sep 26 (Quiz #1)

GPU Architecture: Real Architectures

NVIDIA Architectures (since first CUDA GPU)



Tesla [CC 1.x]: 2007-2009

- G80, G9x: 2007 (Geforce 8800, ...)
GT200: 2008/2009 (GTX 280, ...)

Fermi [CC 2.x]: 2010 (2011, 2012, 2013, ...)

- GF100, ... (GTX 480, ...)
GF104, ... (GTX 460, ...)
GF110, ... (GTX 580, ...)

Kepler [CC 3.x]: 2012 (2013, 2014, 2016, ...)

- GK104, ... (GTX 680, ...)
GK110, ... (GTX 780, GTX Titan, ...)

Maxwell [CC 5.x]: 2015

- GM107, ... (GTX 750Ti, ...)
GM204, ... (GTX 980, Titan X, ...)

Pascal [CC 6.x]: 2016 (2017, 2018, 2021, 2022, ...)

- GP100 (Tesla P100, ...)
- GP10x: x=2,4,6,7,8, ...
(GTX 1060, 1070, 1080, Titan X *Pascal*, Titan Xp, ...)

Volta [CC 7.0, 7.2]: 2017/2018

- GV100, ...
(Tesla V100, Titan V, Quadro GV100, ...)

Turing [CC 7.5]: 2018/2019

- TU102, TU104, TU106, TU116, TU117, ...
(Titan RTX, RTX 2070, 2080 (Ti), GTX 1650, 1660, ...)

Ampere [CC 8.0, 8.6, 8.7]: 2020

- GA100, GA102, GA104, GA106, ...
(A100, RTX 3070, 3080, 3090 (Ti), RTX A6000, ...)

Hopper [CC 9.0], Ada Lovelace [CC 8.9]: 2022/23

- GH100, AD102, AD103, AD104, ...
(H100, L40, RTX 4080 (12/16 GB), 4090, RTX 6000, ...)

Blackwell [CC 10.0]: *coming in 2024/25*

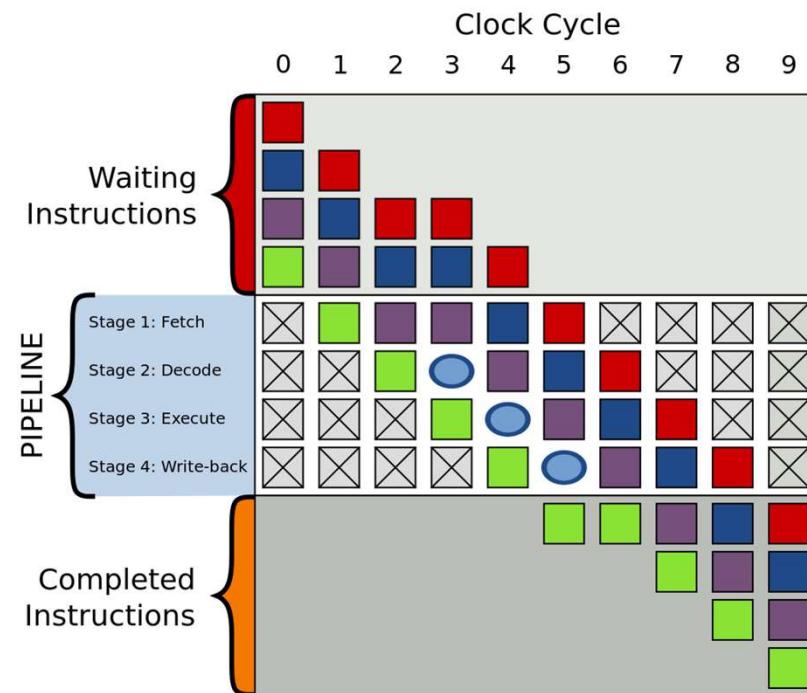
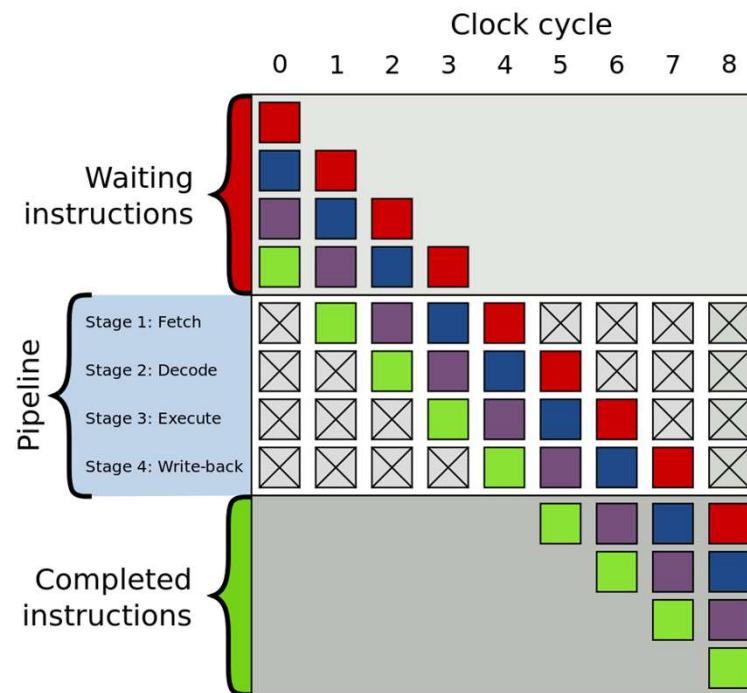
- GB200/GB202, GB20x, ...?
(RTX 5080/5090, GB200 NVL72, HGX B100/200, ...?)

Instruction Pipelining



Most basic way to exploit instruction-level parallelism (ILP)

Problem: hazards (different solutions: bubbles, ...)



https://en.wikipedia.org/wiki/Instruction_pipelining
https://en.wikipedia.org/wiki/Classic_RISC_pipeline

wikipedia

Example: “Scalar” GF100

Main concept here:

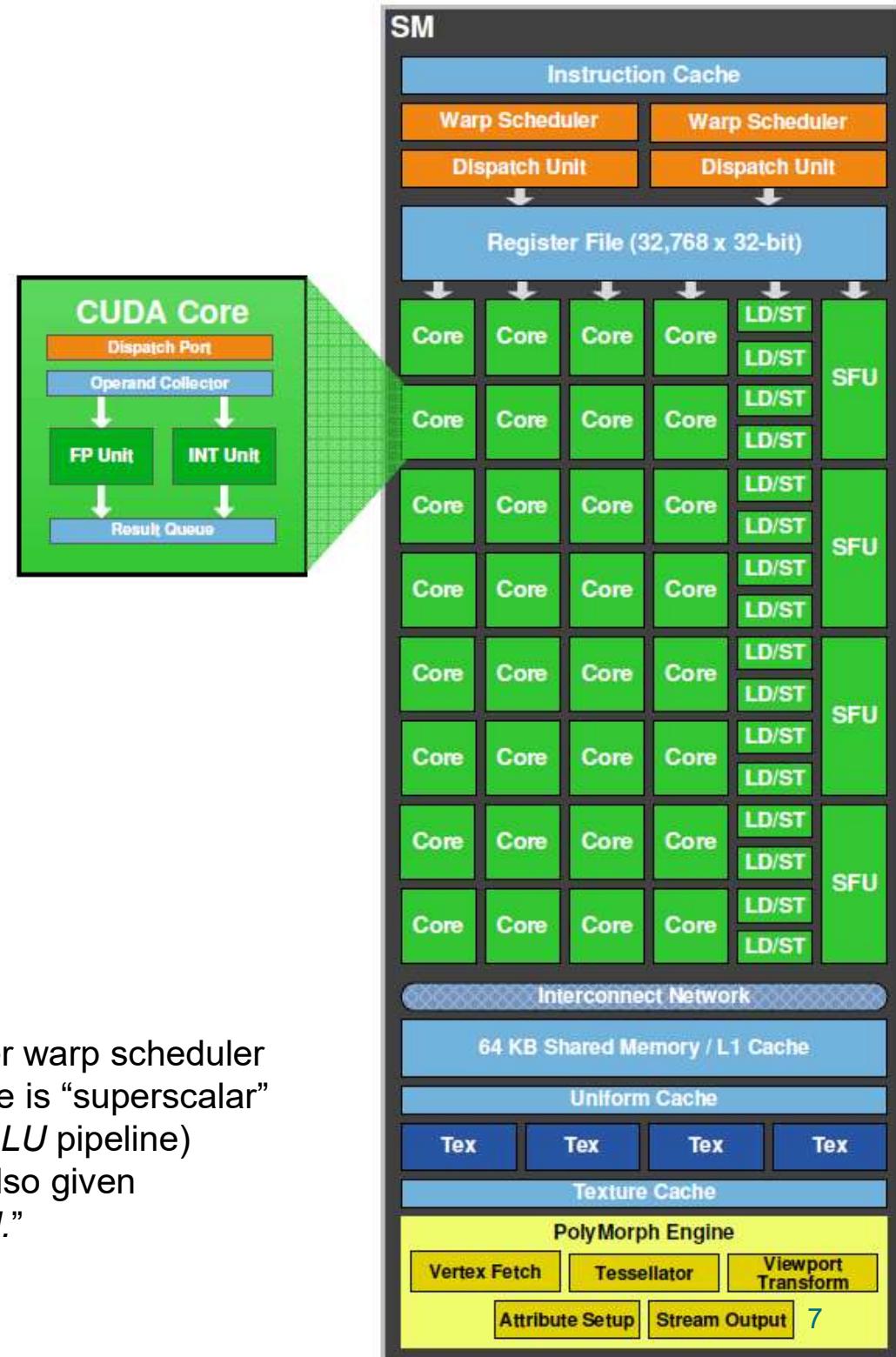
There is one instruction dispatcher
(dispatch unit / fetch/decode unit)
per warp scheduler
(warp selector)

Details later...

Ignore less important subtleties...

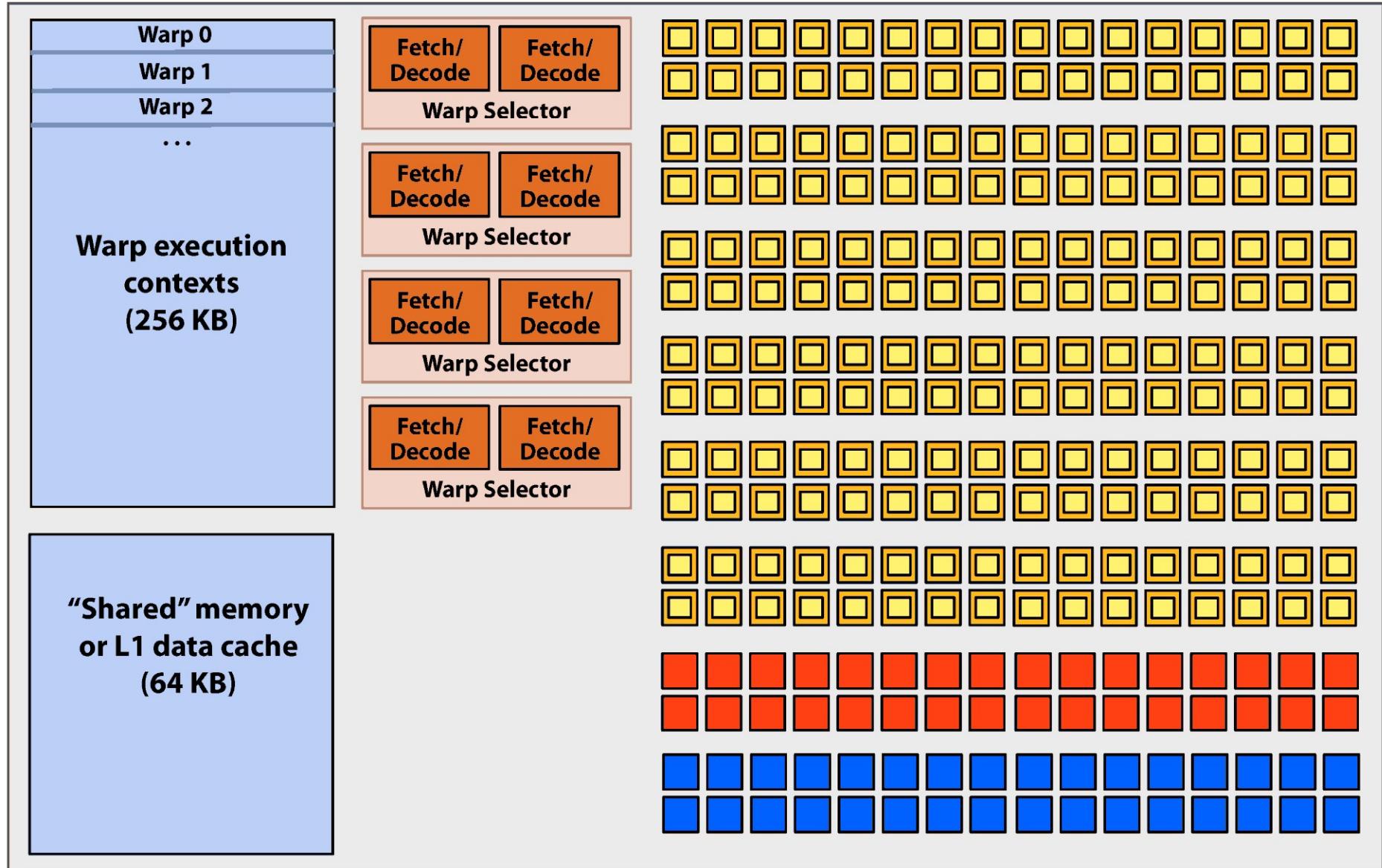
GF100 has two warp schedulers, not one,
and each 32-thread instruction is executed
over two clock cycles, not one, etc.

Caveat on NVIDIA diagrams: if two dispatchers per warp scheduler are shown, it still doesn't mean that the ALU pipeline is “superscalar” (often, the second dispatcher dispatches to a *non-ALU* pipeline)
... need to look at CUDA programming guide info, also given in our tables in row “# ALU dispatch / warp sched.”



Example: “Superscalar” ALUs in SM Architecture

NVIDIA Kepler GK104 architecture SMX unit (one “core”)



= SIMD function unit,
control shared across 32 units
(1 MUL-ADD per clock)

= “special” SIMD function unit,
control shared across 32 units
(operations like sin/cos)

= SIMD load/store unit
(handles warp loads/stores, gathers/scatters)



Instruction Throughput

Instruction throughput numbers in CUDA C Programming Guide (Chapter 8.4)

| | Compute Capability | | | | | | | | | | |
|---|---|-------------|-----|-----|-----|-----|---|-----|-----|-----|--|
| | 3.5, 3.7 | 5.0, 5.2 | 5.3 | 6.0 | 6.1 | 6.2 | 7.x | 8.0 | 8.6 | 8.9 | 9.0 |
| 16-bit floating-point add, multiply, multiply-add | N/A | | 256 | 128 | 2 | 256 | 128 | 256 | 128 | 256 | 128 for __nv_bfloat16 128 for __nv_bfloat16 |
| 32-bit floating-point add, multiply, multiply-add | 192 | 128 | | 64 | 128 | | 64 | | 128 | | |
| 64-bit floating-point add, multiply, multiply-add | 64 8 for GeForce GPUs, except for Titan GPUs | 4 | | 32 | 4 | | 32 2 for compute capability 7.5 GPUs | 32 | 2 | 2 | 64 |



Instruction Throughput

Instruction throughput numbers in CUDA C Programming Guide (Chapter 8.4)

| | Compute Capability | | | | | | | | | | |
|--|--------------------|-------------|--------------------|-----|-----|-----|-----|-----|---------------------------|-----|-----|
| | 3.5, 3.7 | 5.0, 5.2 | 5.3 | 6.0 | 6.1 | 6.2 | 7.x | 8.0 | 8.6 | 8.9 | 9.0 |
| 32-bit floating-point reciprocal, reciprocal square root, base-2 logarithm (<code>__log2f</code>), base 2 exponential (<code>exp2f</code>), sine (<code>__sinf</code>), cosine (<code>__cosf</code>) | | 32 | | 16 | 32 | | | 16 | | | |
| 32-bit integer add, extended-precision add, subtract, extended-precision subtract | 160 | | 128 | 64 | 128 | | | 64 | | | |
| 32-bit integer multiply, multiply-add, extended-precision multiply-add | 32 | | Multiple instruct. | | | | 64 | | 32 for extended-precision | | |

list continues...

ALU Instruction Latencies and Instructs. / SM



| CC | 2.0 (Fermi) | 2.1 (Fermi) | 3.x (Kepler) | 5.x (Maxwell) | 6.0 (Pascal) | 6.1/6.2 (Pascal) | 7.x (Volta, Turing) | 8.0/8.6 (Ampere) | 8.9/9.0 (Ada/Hopper) |
|------------------------------------|----------------------|----------------------|---------------------|------------------|-----------------|---------------------|------------------------|---------------------|-------------------------|
| # warp sched. / SM | 2 | 2 | 4 | 4 | 2 | 4 | 4 | 4 | 4 |
| # ALU dispatch / warp sched. | 1 (over 2 clocks) | 2 (over 2 clocks) | 2 | 1 | 1 | 1 | 1 | 1 | 1 |
| SM busy with # warps + inst | L | 2L | 8L | 4L | 2L | 4L | 4L | 4L | 4L |
| inst. pipe latency (L) | 22 | 22 | 11 | 9 | 6 | 6 | 4 | 4 | 4 |
| SM busy with # warps | 22 | 22 + ILP | 44 + ILP | 36 | 12 | 24 | 16 | 16 | 16 |

see NVIDIA CUDA C Programming Guides (different versions)
performance guidelines/multiprocessor level; compute capabilities

ALU Instruction Latencies and Instructs. / SM



| CC | 2.0 (Fermi) | 2.1 (Fermi) | 3.x (Kepler) | 5.x (Maxwell) | 6.0 (Pascal) | 6.1/6.2 (Pascal) | 7.x (Volta, Turing) | 8.0/8.6 (Ampere) | 8.9/9.0 (Ada/Hopper) |
|------------------------------------|----------------------|----------------------|---------------------|------------------|-----------------|---------------------|------------------------|---------------------|-------------------------|
| # warp sched. / SM | 2 | 2 | 4 | 4 | 2 | 4 | 4 | 4 | 4 |
| # ALU dispatch / warp sched. | 1 (over 2 clocks) | 2 (over 2 clocks) | 2 | 1 | 1 | 1 | 1 | 1 | 1 |
| SM busy with # warps + inst | L | 2L | 8L | 4L | 2L | 4L | 4L | 4L | 4L |
| inst. pipe latency (L) | 22 | 22 | 11 | 9 | 6 | 6 | 4 | 4 | 4 |
| SM busy with # warps | 22 | 22 + ILP | 44 + ILP | 36 | 12 | 24 | 16 | 16 | 16 |

*IF no other stalls occur!
(i.e., except inst. pipe hazards)*

see NVIDIA CUDA C Programming Guides (different versions)
performance guidelines/multiprocessor level; compute capabilities

ALU Instruction Latencies and Instructs. / SM



| CC | 2.0 (Fermi) | 2.1 (Fermi) | 3.x (Kepler) | 5.x (Maxwell) | 6.0 (Pascal) | 6.1/6.2 (Pascal) | 7.x (Volta, Turing) | 8.0/8.6 (Ampere) | 8.9/9.0 (Ada/Hopper) |
|------------------------------------|----------------------|----------------------|-----------------|------------------|-----------------|---------------------|------------------------|---------------------|-------------------------|
| # warp sched. / SM | 2 | 2 | 4 | 4 | 2 | 4 | 4 | 4 | 4 |
| # ALU dispatch / warp sched. | 1 (over 2 clocks) | 2 (over 2 clocks) | 2 | 1 | 1 | 1 | 1 | 1 | 1 |
| SM busy with # warps + inst | L | 2L | 8L | 4L | 2L | 4L | 4L | 4L | 4L |
| inst. pipe latency (L) | 22 | 22 | 11 | 9 | 6 | 6 | 4 | 4 | 4 |
| SM busy with # warps | 22 | 22 + ILP | 44 + ILP | 36 | 12 | 24 | 16 | 16 | 16 |

IF no other stalls occur! “superscalar”
(i.e., except inst. pipe hazards)

see NVIDIA CUDA C Programming Guides (different versions)
performance guidelines/multiprocessor level; compute capabilities

ALU Instruction Latencies and Instructs. / SM



| CC | 2.0 (Fermi) | 2.1 (Fermi) | 3.x (Kepler) | 5.x (Maxwell) | 6.0 (Pascal) | 6.1/6.2 (Pascal) | 7.x (Volta, Turing) | 8.0/8.6 (Ampere) | 8.9/9.0 (Ada/Hopper) |
|------------------------------------|----------------------|----------------------|-----------------|------------------|-----------------|---------------------|------------------------|---------------------|-------------------------|
| # warp sched. / SM | 2 | 2 | 4 | 4 | 2 | 4 | 4 | 4 | 4 |
| # ALU dispatch / warp sched. | 1 (over 2 clocks) | 2 (over 2 clocks) | 2 | 1 | 1 | 1 | 1 | 1 | 1 |
| SM busy with # warps + inst | L | 2L | 8L | 4L | 2L | 4L | 4L | 4L | 4L |
| inst. pipe latency (L) | 22 | 22 | 11 | 9 | 6 | 6 | 4 | 4 | 4 |
| SM busy with # warps | 22 | 22 + ILP | 44 + ILP | 36 | 12 | 24 | 16 | 16 | 16 |

IF no other stalls occur! “superscalar”
(i.e., except inst. pipe hazards)

see NVIDIA CUDA C Programming Guides (different versions)
performance guidelines/multiprocessor level; compute capabilities



NVIDIA Tesla Architecture

2007-2009

(compute capability 1.x)

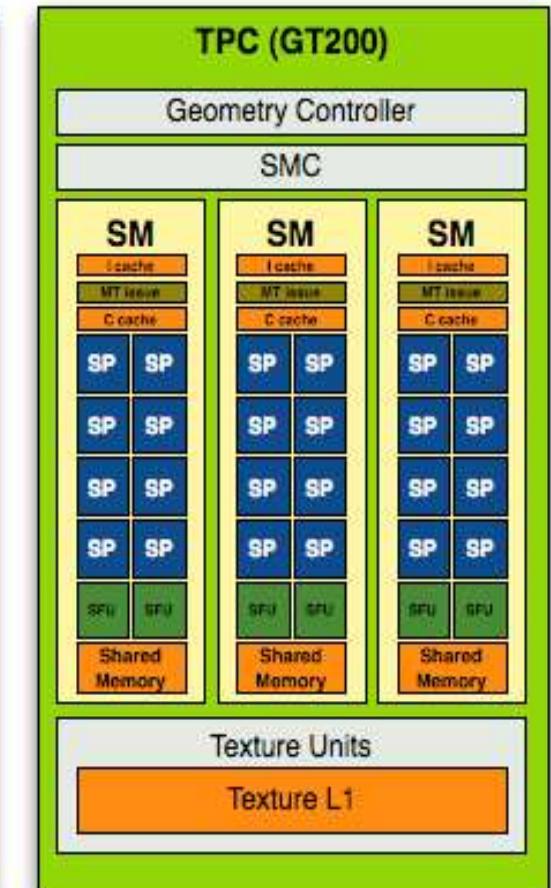
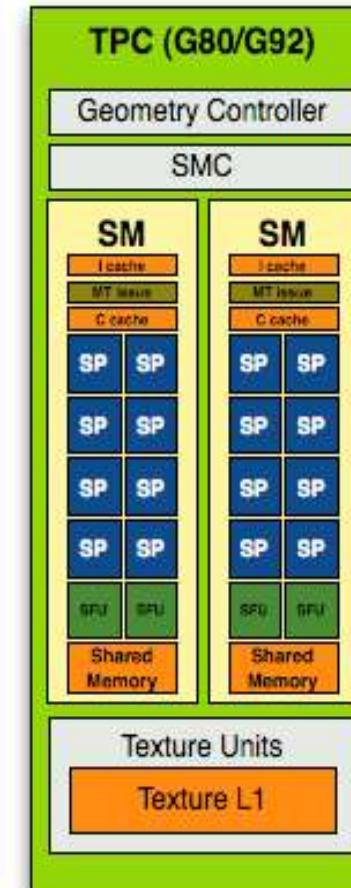
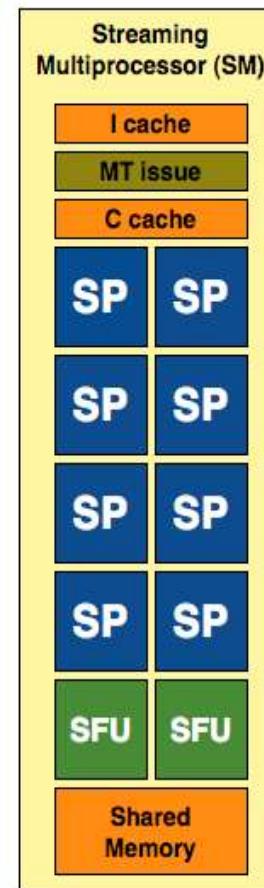
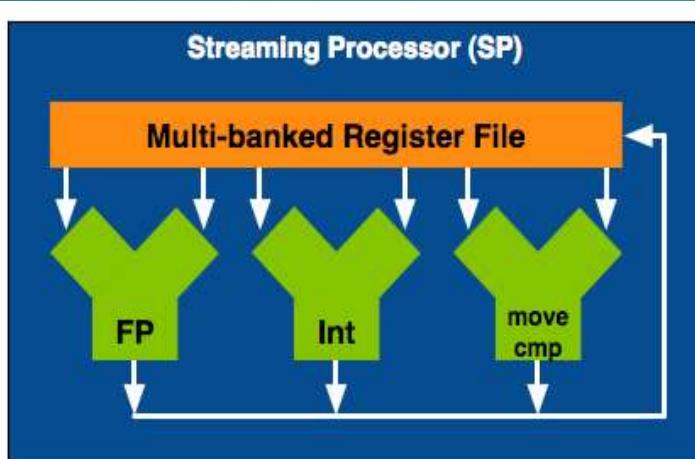
G80 (cc 1.0): 2007 (Geforce 8800, ...)

G9x (cc 1.1): 2008 (Geforce 9800, ...)

GT200 (cc 1.3): 2008/2009 (GTX 280, GTX 285, ...)

(this is not the Tesla product line!)

NVIDIA Tesla Architecture (not the Tesla product line!), G80: 2007, GT200: 2008/2009



G80: first CUDA GPU!

Multiprocessor: SM (CC 1.x)

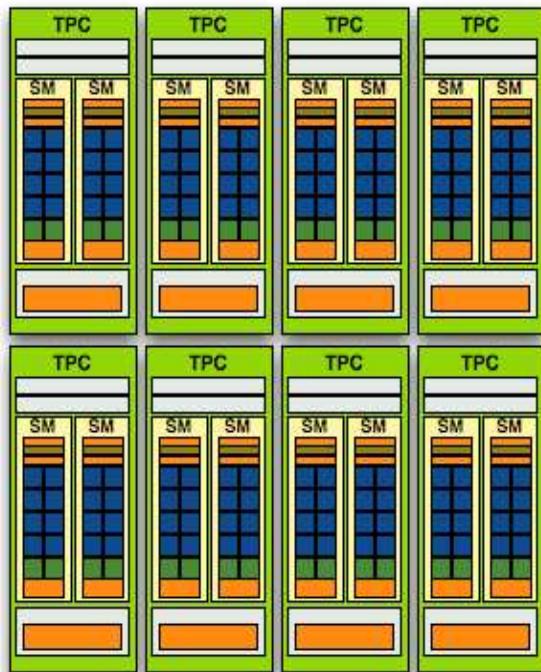
Courtesy AnandTech

- Streaming Processor (SP) [or: CUDA core; or: FP32 / FP64 / INT32 core, ...]
- Streaming Multiprocessor (SM)
- Texture/Processing Cluster (TPC)

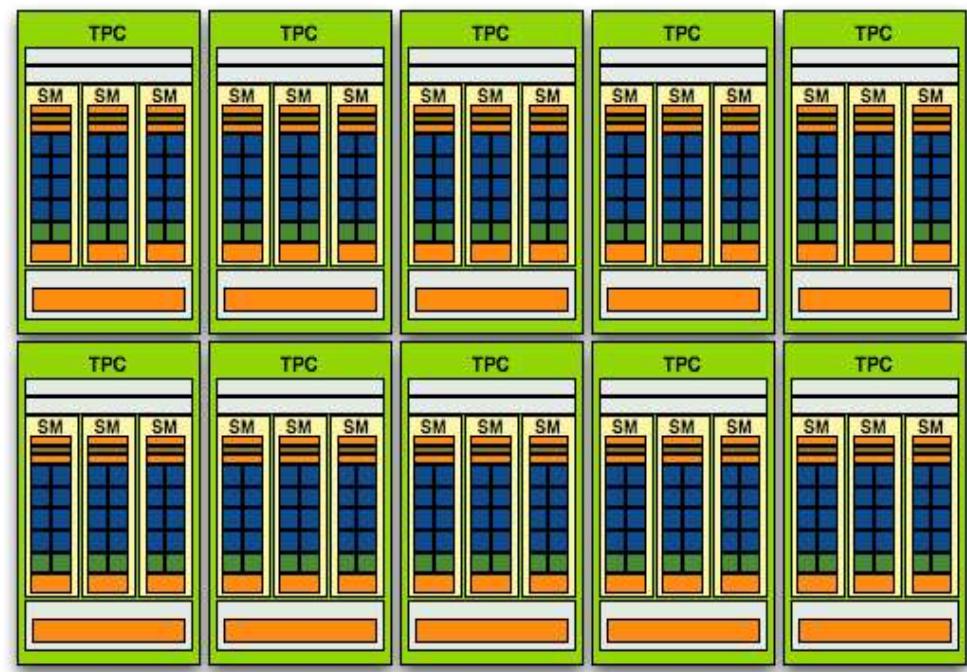
NVIDIA Tesla Architecture (not the Tesla product line!), G80: 2007, GT200: 2008/2009



- G80/G92: $8 \text{ TPCs} * (2 * 8 \text{ SPs}) = 128 \text{ SPs} [= \text{CUDA cores}]$
- GT200: $10 \text{ TPCs} * (3 * 8 \text{ SPs}) = 240 \text{ SPs} [= \text{CUDA cores}]$
- **Arithmetic intensity** has increased (num. of ALUs vs. texture units)



G80 / G92



GT200

Courtesy AnandTech



NVIDIA Fermi Architecture

2010

(compute capability 2.x)

GF100 (cc 2.0), ... (GTX 480, ...)

GF104 (cc 2.1), ... (GTX 460, ...)

GF110 (cc 2.0), ... (GTX 580, ...)

NVIDIA Fermi (GF100) Architecture (2010)



Full size

- 4 GPCs
- 4 SMs each
- 6 64-bit memory controllers (= 384 bit)

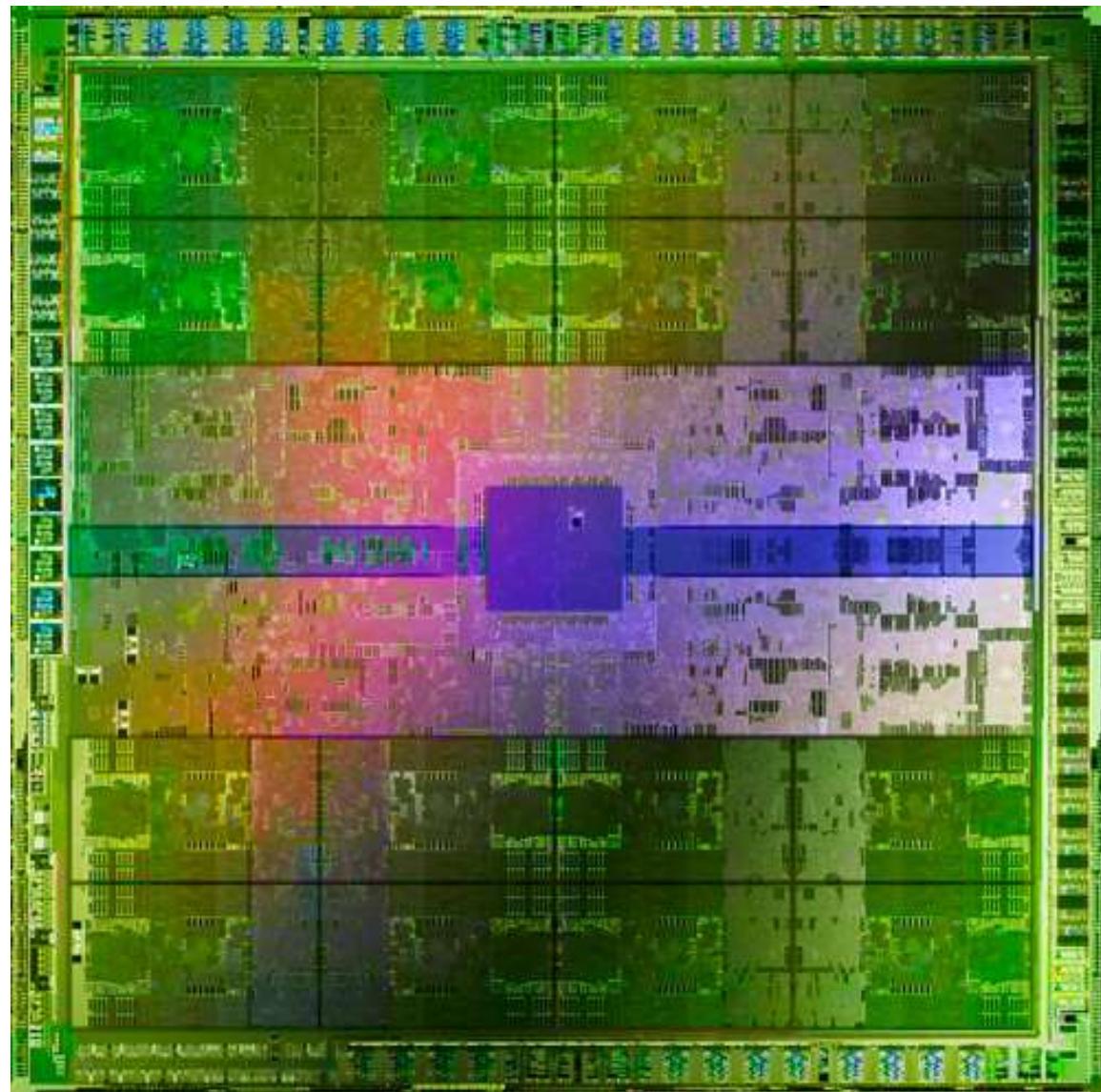




NVIDIA Fermi (GF100) Die Photo

Full size

- 4 GPCs
- 4 SMs each



ALU Instruction Latencies and Instructs. / SM



| CC | 2.0 (Fermi) | 2.1 (Fermi) | 3.x (Kepler) | 5.x (Maxwell) | 6.0 (Pascal) | 6.1/6.2 (Pascal) | 7.x (Volta, Turing) | 8.0/8.6 (Ampere) | 8.9/9.0 (Ada/Hopper) |
|------------------------------------|----------------------|----------------------|-----------------|------------------|-----------------|---------------------|------------------------|---------------------|-------------------------|
| # warp sched. / SM | 2 | 2 | 4 | 4 | 2 | 4 | 4 | 4 | 4 |
| # ALU dispatch / warp sched. | 1 (over 2 clocks) | 2 (over 2 clocks) | 2 | 1 | 1 | 1 | 1 | 1 | 1 |
| SM busy with # warps + inst | L | 2L | 8L | 4L | 2L | 4L | 4L | 4L | 4L |
| inst. pipe latency (L) | 22 | 22 | 11 | 9 | 6 | 6 | 4 | 4 | 4 |
| SM busy with # warps | 22 | 22 + ILP | 44 + ILP | 36 | 12 | 24 | 16 | 16 | 16 |

see NVIDIA CUDA C Programming Guides (different versions)
 performance guidelines/multiprocessor level; compute capabilities

NVIDIA GF100 SM (2010)

Multiprocessor: SM (CC 2.0)

Streaming processors now called
CUDA cores

32 CUDA cores per Fermi GF100/GF110
streaming multiprocessor (SM)

Example GPU with 15 SMs = 480 CUDA cores (GTX 480)

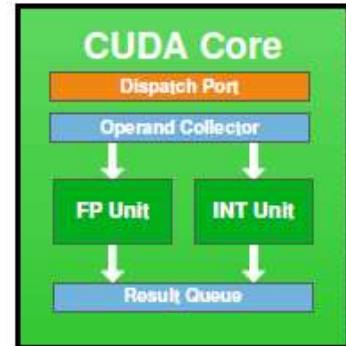
Example GPU with 16 SMs = 512 CUDA cores (GTX 580)

CPU-like cache hierarchy

- L1 cache / shared memory
- L2 cache

Texture units and caches now in SM

(instead of with TPC=multiple SMs in G80/GT200)





Graphics Processor Clusters (GPC)

(instead of TPC on GT200)

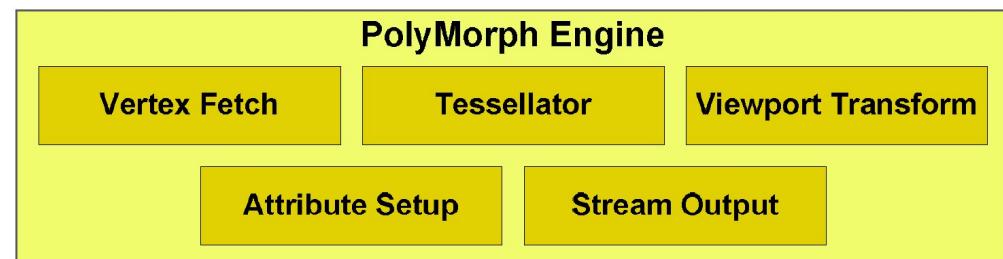
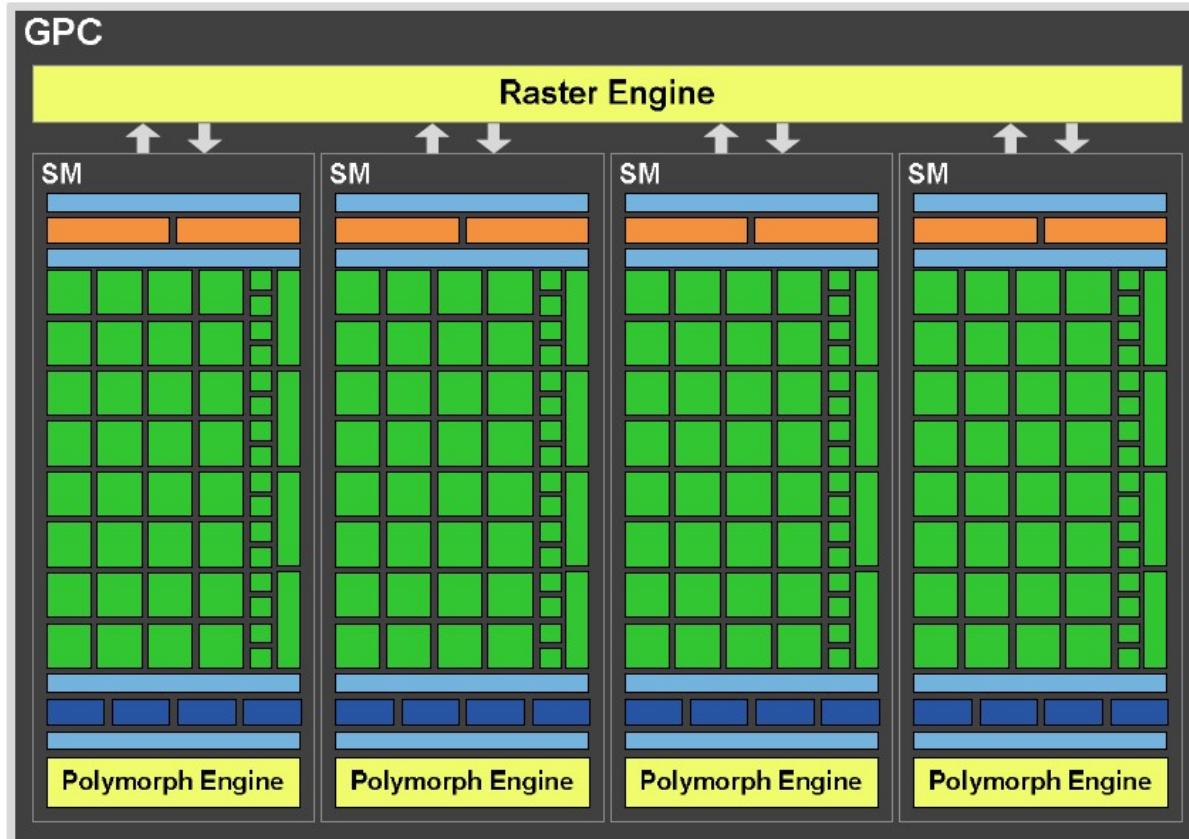
4 SMs

32 CUDA cores / SM

4 SMs / GPC =
128 cores / GPC

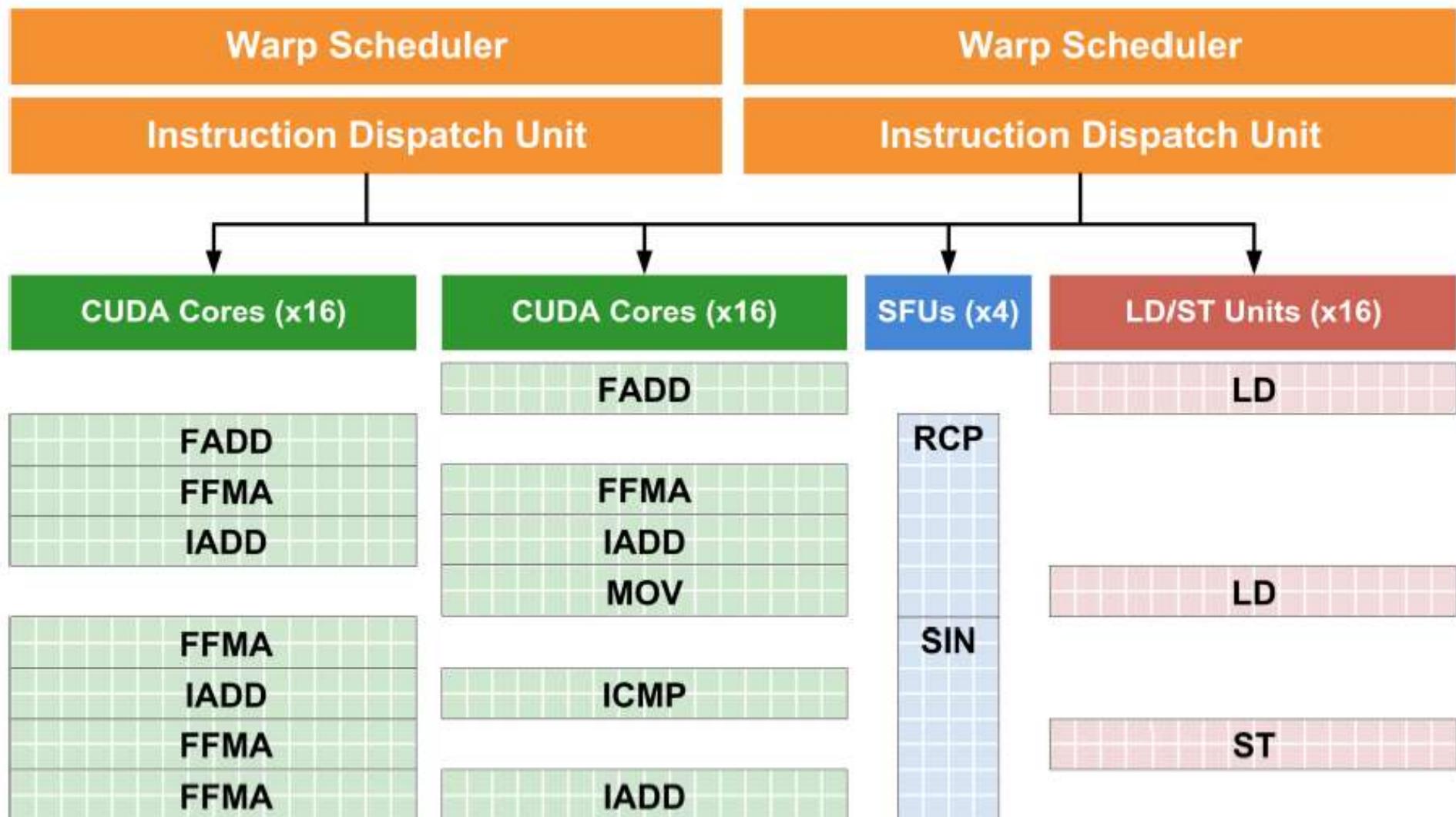
Decentralized rasterization
and geometry

- 4 raster engines
- 16 "PolyMorph" engines

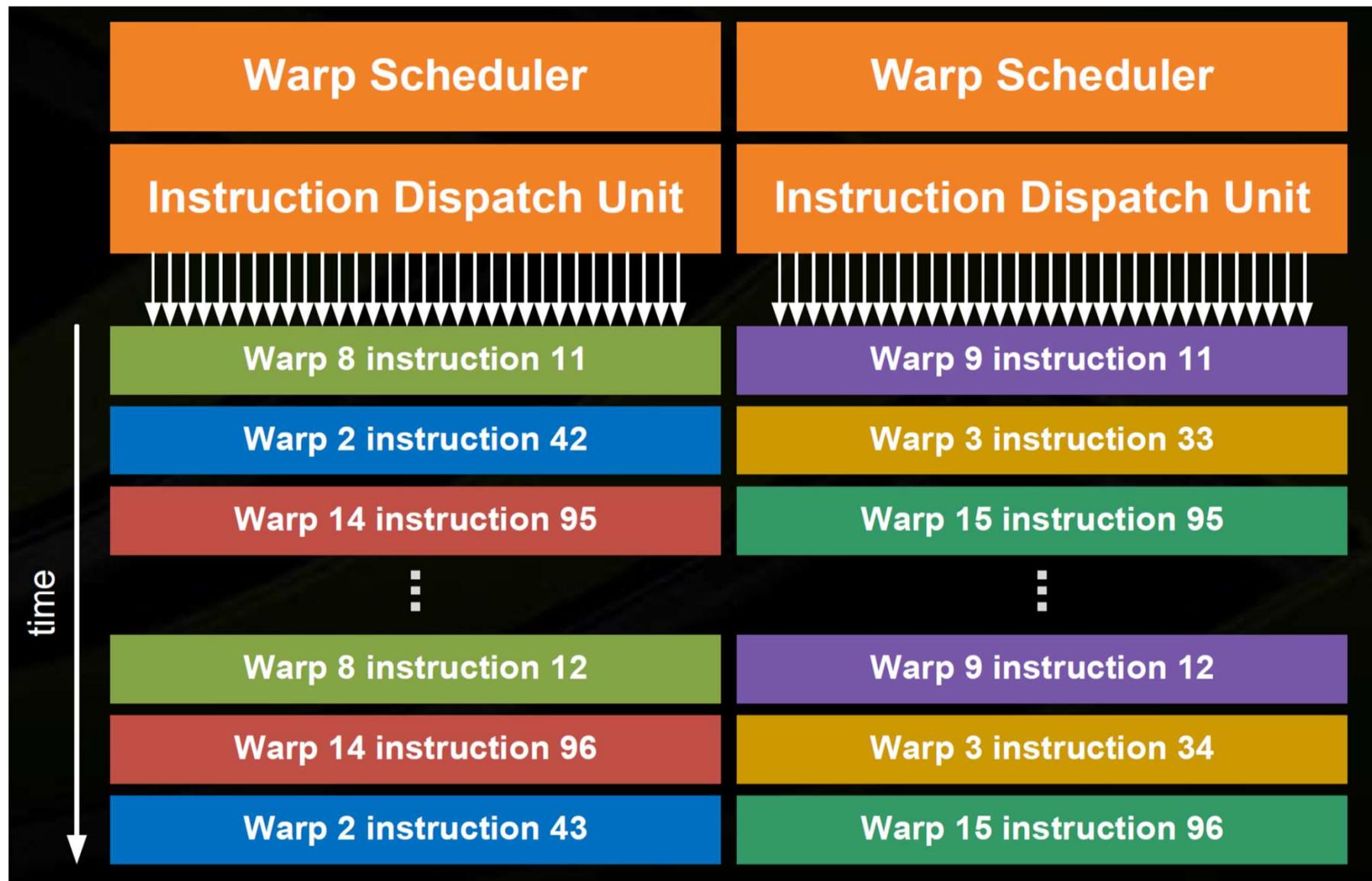




Dual Warp Schedulers



Dual Warp Schedulers

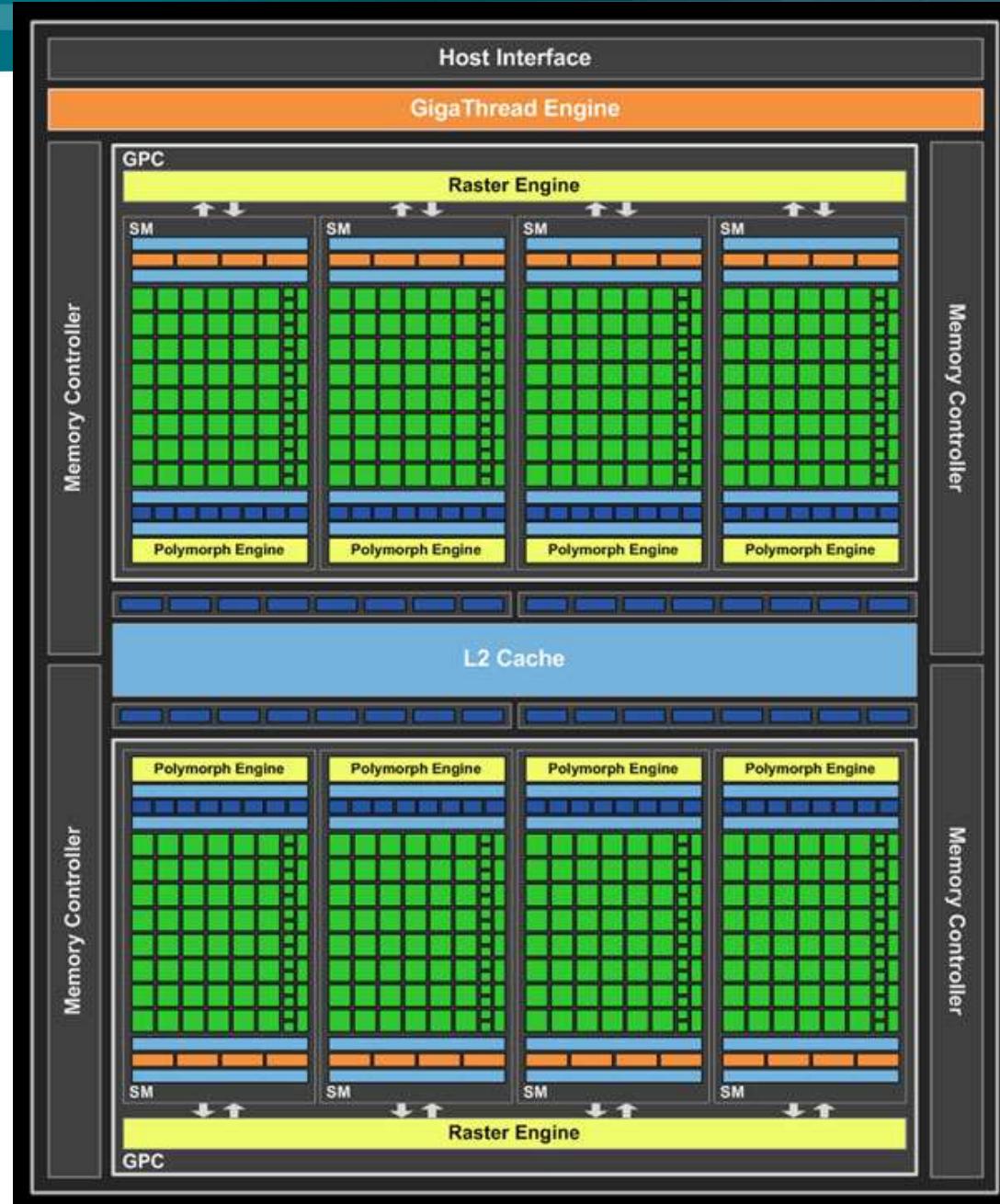


NVIDIA Fermi (GF104) Architecture (2010)



Full size GF104

- 2 GPCs
- 4 SMs each
- SM design different from GF100 / GF110 !
- Fewer total SMs, but each SM is “superscalar”



ALU Instruction Latencies and Instructs. / SM



| CC | 2.0 (Fermi) | 2.1 (Fermi) | 3.x (Kepler) | 5.x (Maxwell) | 6.0 (Pascal) | 6.1/6.2 (Pascal) | 7.x (Volta, Turing) | 8.0/8.6 (Ampere) | 8.9/9.0 (Ada/Hopper) |
|------------------------------------|----------------------|----------------------|-----------------|------------------|-----------------|---------------------|------------------------|---------------------|-------------------------|
| # warp sched. / SM | 2 | 2 | 4 | 4 | 2 | 4 | 4 | 4 | 4 |
| # ALU dispatch / warp sched. | 1 (over 2 clocks) | 2 (over 2 clocks) | 2 | 1 | 1 | 1 | 1 | 1 | 1 |
| SM busy with # warps + inst | L | 2L | 8L | 4L | 2L | 4L | 4L | 4L | 4L |
| inst. pipe latency (L) | 22 | 22 | 11 | 9 | 6 | 6 | 4 | 4 | 4 |
| SM busy with # warps | 22 | 22 + ILP | 44 + ILP | 36 | 12 | 24 | 16 | 16 | 16 |

see NVIDIA CUDA C Programming Guides (different versions)
 performance guidelines/multiprocessor level; compute capabilities

NVIDIA GF104 SM (2010)



Multiprocessor: SM (CC 2.1)

Streaming processors now called
CUDA cores

48 CUDA cores per Fermi GF104
streaming multiprocessor (SM)

Example GPU with 7 SMs = 336 CUDA cores (GTX 460)

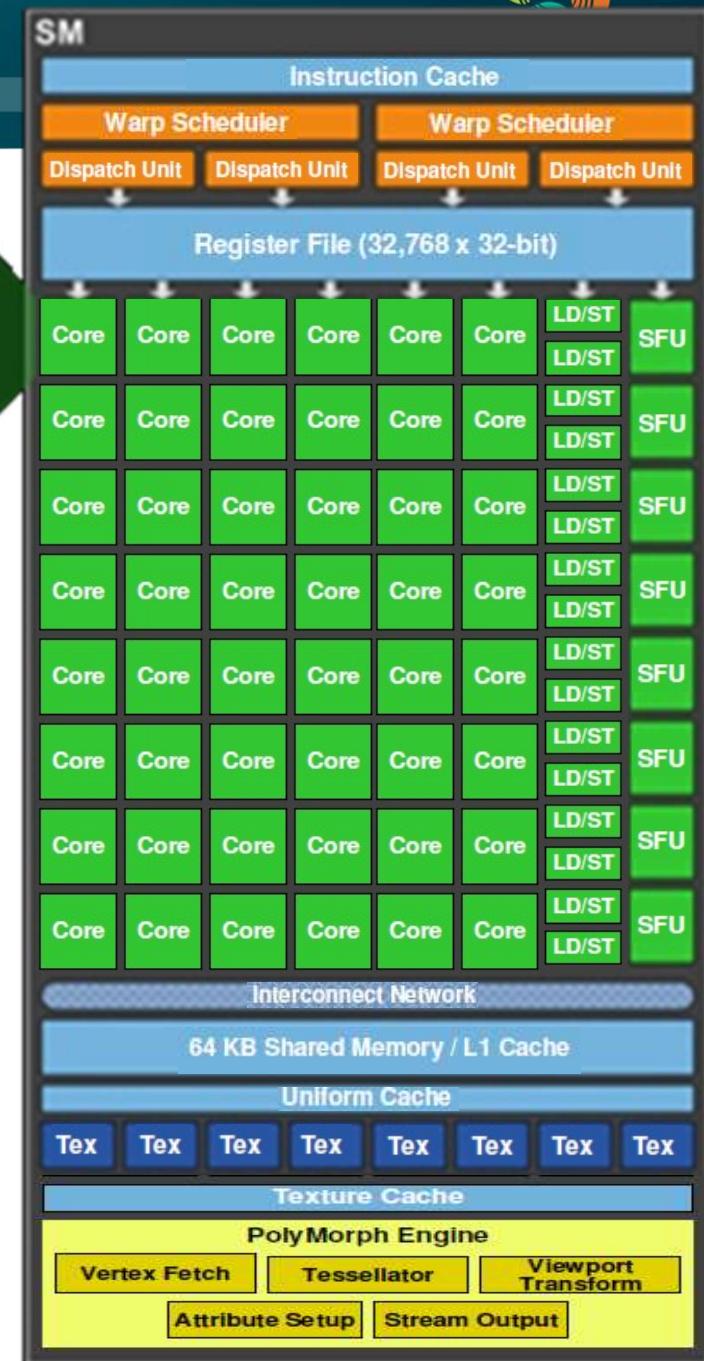
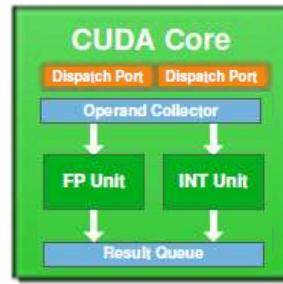
2 dispatch units / warp scheduler: “superscalar”

CPU-like cache hierarchy

- L1 cache / shared memory
- L2 cache

Texture units and caches now in SM

(instead of with TPC=multiple SMs in G80/GT200)

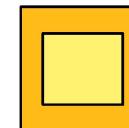
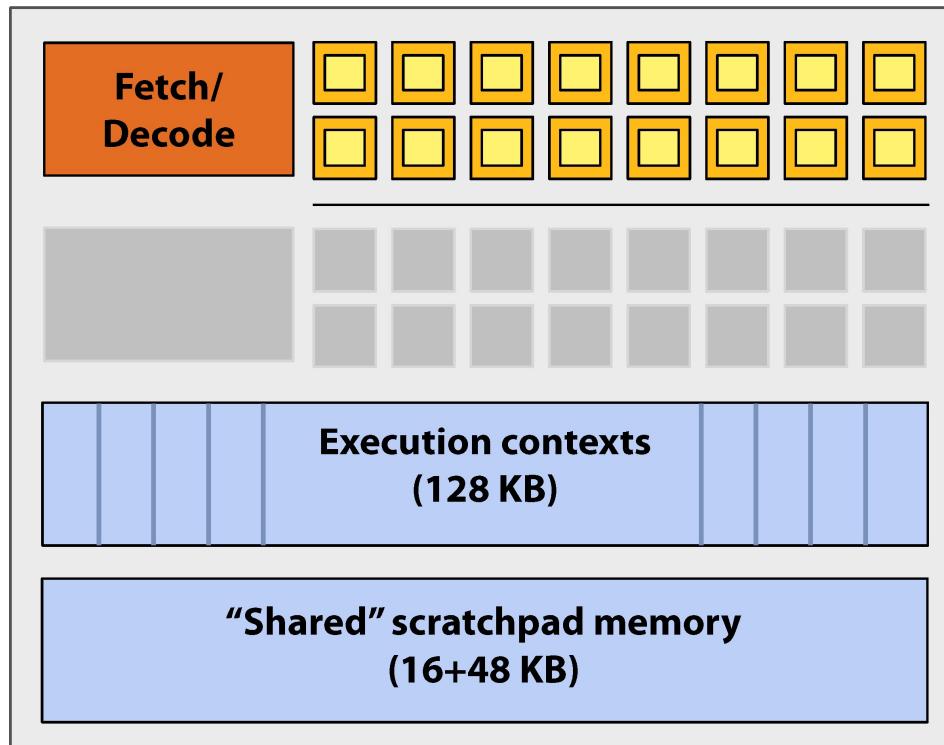


NVIDIA Fermi GF100 Architecture (2010)



NVIDIA GeForce GTX 480 “core”

CC 2.0, not 2.1 !



= SIMD function unit,
control shared across 16 units
(1 MUL-ADD per clock)

- Groups of 32 fragments share an instruction stream
- Up to 48 groups are simultaneously interleaved
- Up to 1536 individual contexts can be stored

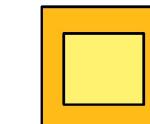
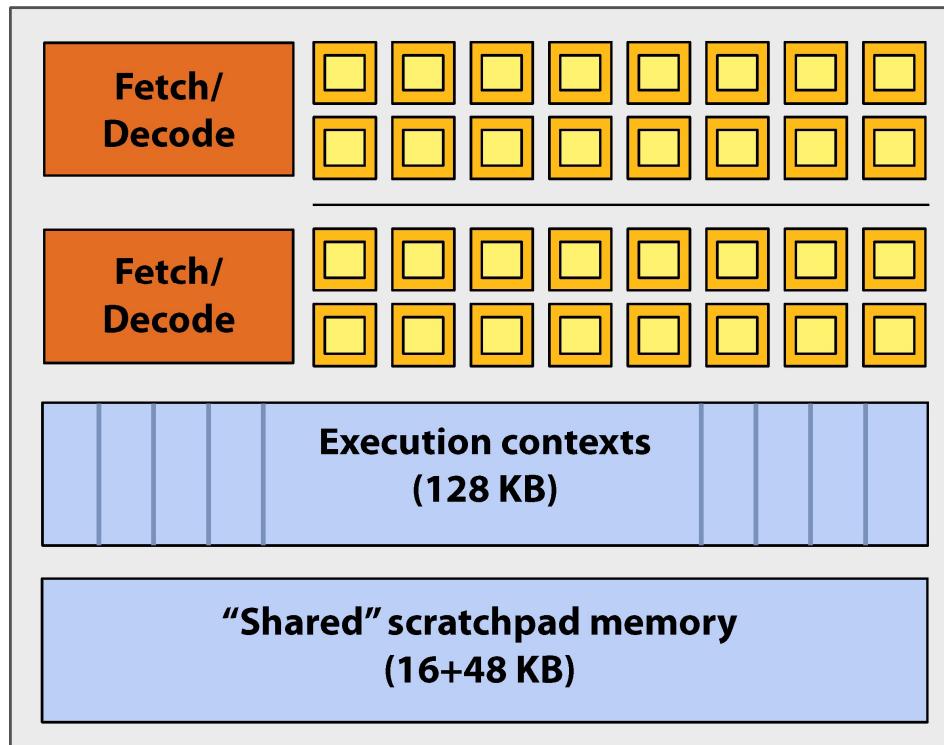
Source: Fermi Compute Architecture Whitepaper
CUDA Programming Guide 3.1, Appendix G

NVIDIA Fermi GF100 Architecture (2010)



NVIDIA GeForce GTX 480 “core”

CC 2.0, not 2.1 !



= SIMD function unit,
control shared across 16 units
(1 MUL-ADD per clock)

- The core contains 32 functional units
- Two groups are selected each clock
(decode, fetch, and execute two instruction streams in parallel)

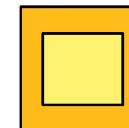
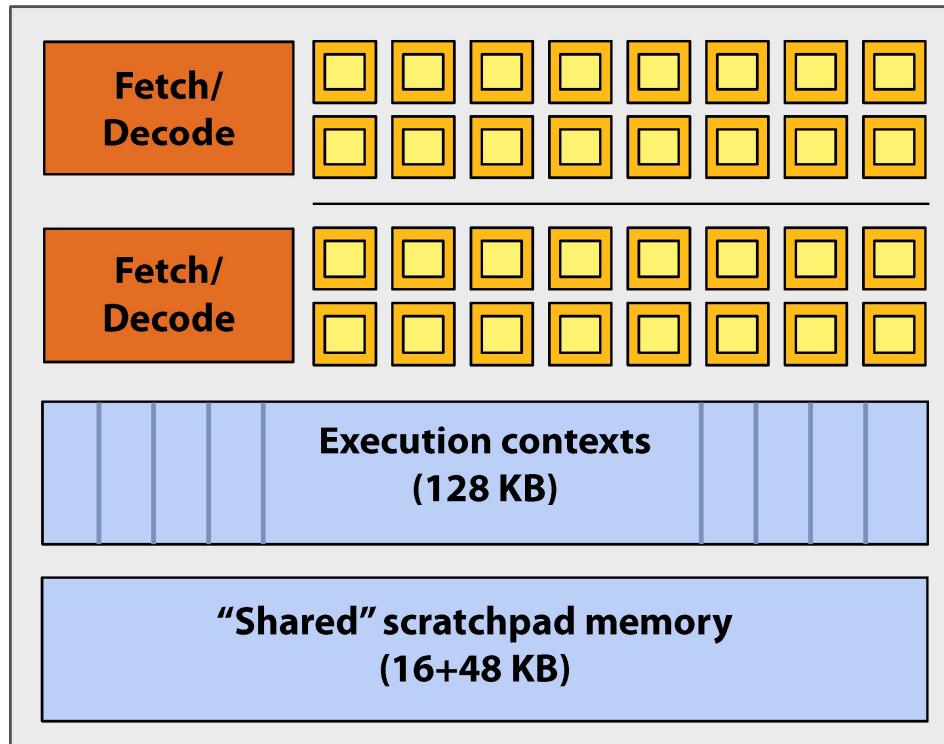
Source: Fermi Compute Architecture Whitepaper
CUDA Programming Guide 3.1, Appendix G

NVIDIA Fermi GF100 Architecture (2010)



NVIDIA GeForce GTX 480 "SM"

CC 2.0, not 2.1 !



= CUDA core
(1 MUL-ADD per clock)

- The **SM** contains 32 **CUDA cores**
- Two **warps** are selected each clock (decode, fetch, and execute two **warps** in parallel)
- Up to 48 warps are interleaved, totaling 1536 **CUDA threads**

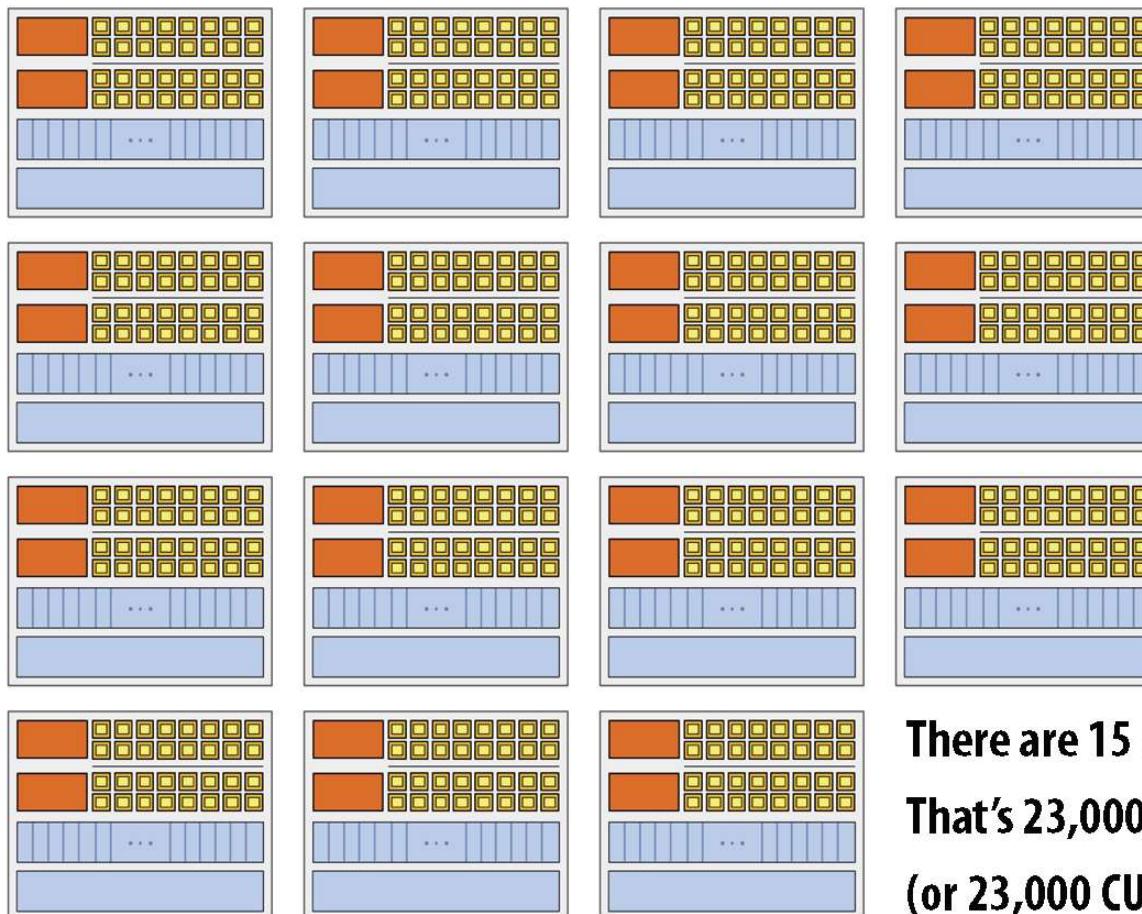
Source: Fermi Compute Architecture Whitepaper
CUDA Programming Guide 3.1, Appendix G

NVIDIA Fermi GF100 Architecture (2010)



NVIDIA GeForce GTX 480

CC 2.0, not 2.1 !



**There are 15 of these things on the GTX 480:
That's 23,000 fragments!
(or 23,000 CUDA threads!)**

Thank you.