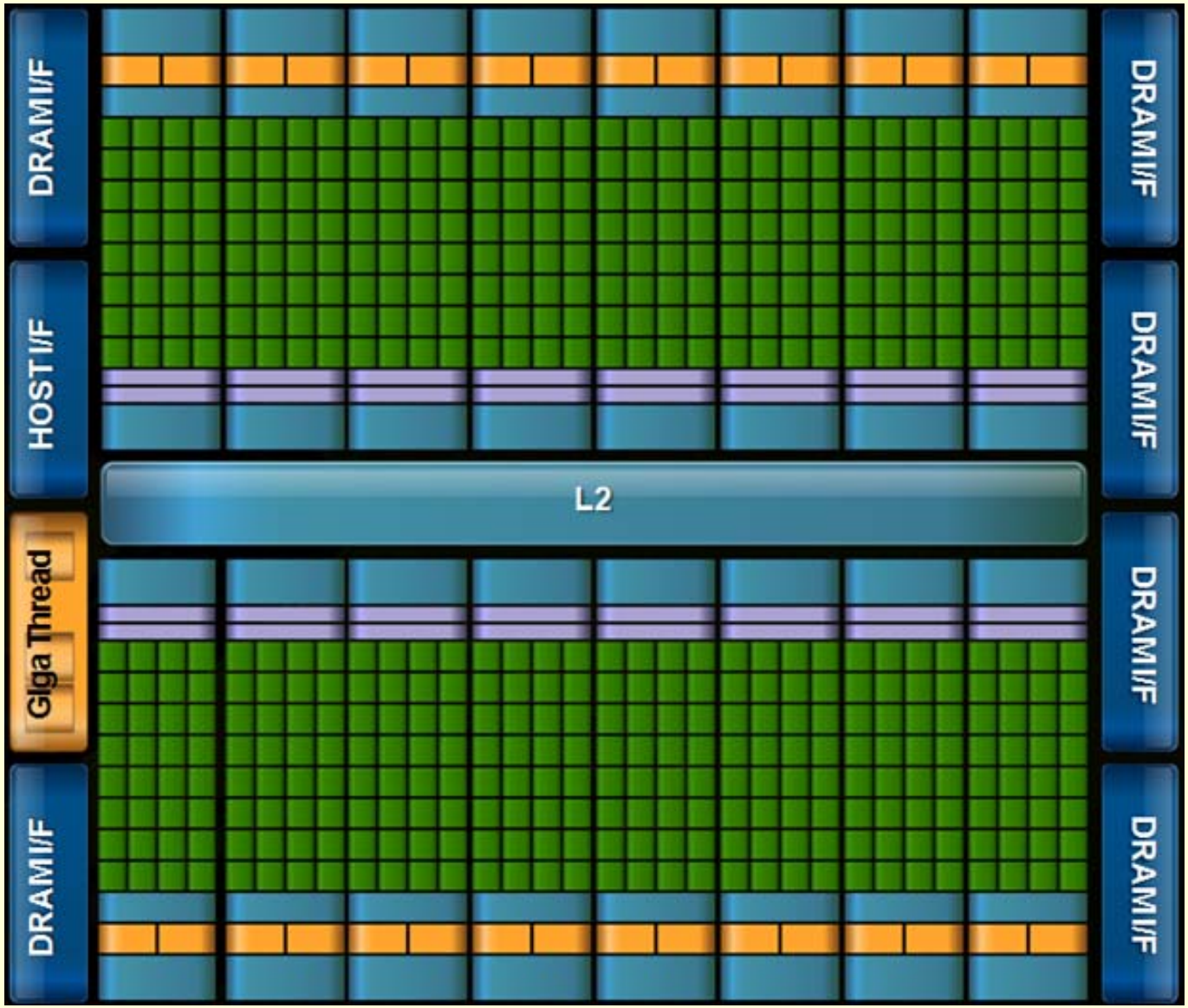
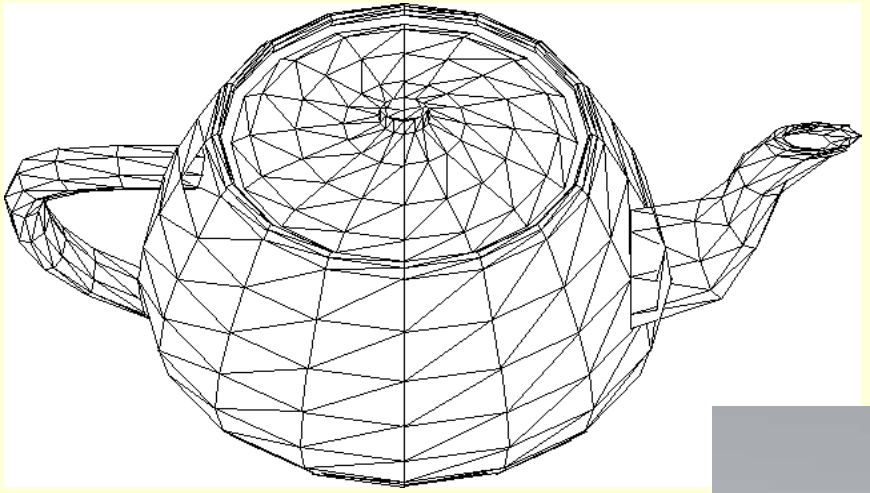
Introduction to GP-GPUs

GPU Architectures: How do we reach here?

NVIDIA Fermi, 512 Processing Elements (PEs)



What Can It Do?

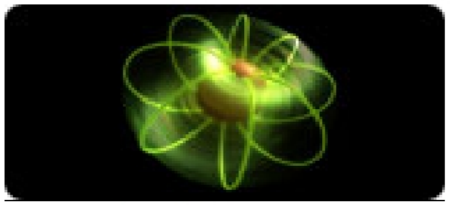


Render triangles.

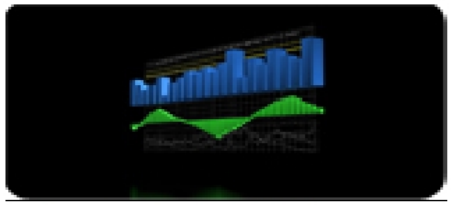
NVIDIA GTX480 can render 1.6 billion triangles per second!

General Purpose Computing

Bio-Informatics and Life Sciences

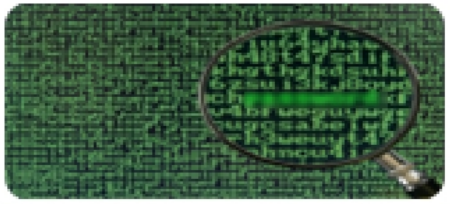


Computational Electromagnetics and Electrodynamics



Computational Finance

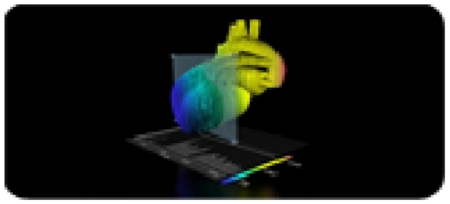
Computational Fluid Dynamics



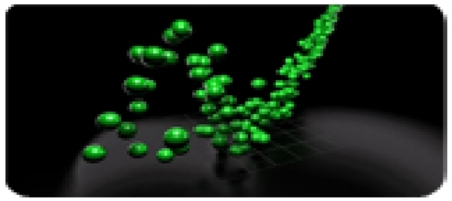
Data Mining, Analytics, and Databases

Imaging and Computer Vision

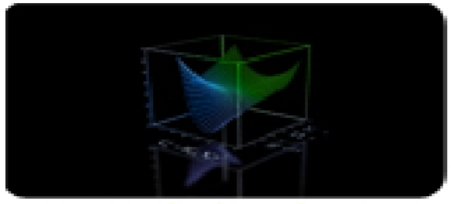
MATLAB Acceleration



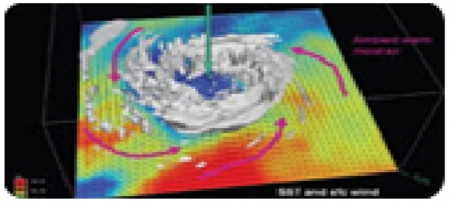
Medical Imaging



Molecular Dynamics



Numerical Packages



Weather, Atmospheric, Ocean Modeling, and Space Sciences

ref: http://www.nvidia.com/object/tesla_computing_solutions.html

General-Purpose GPUs (GP-GPUs)

- In 2006, NVIDIA introduced GeForce 8800 GPU supporting a new programming language:
  - ***CUDA,*** "Compute Unified Device Architecture"
  - Subsequently, broader industry pushing for ***OpenCL***, a vendor-neutral version of same ideas for multiple platforms.
- ***Basic 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 device for execution

Notice: This lecture has a simplified version of Nvidia CUDA-style model and only considers GPU execution for computational kernels, not graphics (Would probably need another course to describe graphics processing)

General-Purpose GPUs (GP-GPUs)

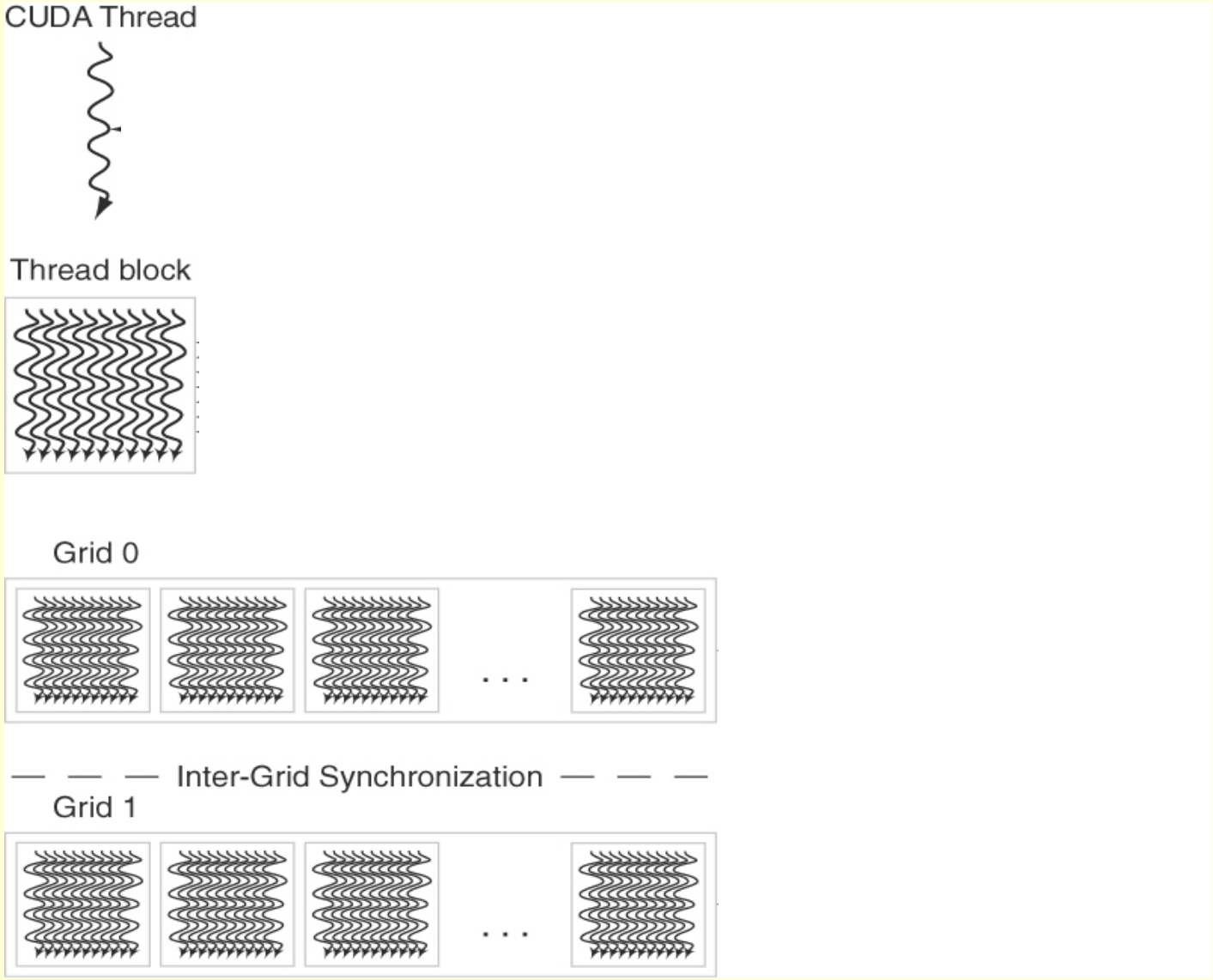
- Given the hardware invested to do graphics well, how can be supplement it to improve performance of a wider range of general purpose applications?
- ***Basic concepts:***
  - ***Heterogeneous*** execution model
    - CPU is the ***host***, GPU is the ***device***
  - Develop a C-like programming language for GPUs
  - Unify all forms of GPU parallelism as *CUDA thread*
  - Programming model is ***"Single Instruction Multiple Thread" (SIMT):*** massive number of ***light-weight*** threads
  - Programmer unware of number of parallel cores

Threads, Blocks and Grids

- ***Thread:*** lower level of parallelism as programming primitive.
  - Each *thread is composed of SIMD instructions*
  - A thread is associated with each data element
- Threads are organized into ***Blocks***
- Blocks are organized into ***Grids***

- GPU hardware handles thread management, not applications or OS

Threads, Blocks and Grids (2)



Threads, Blocks and Grids (3)

- Each *thread of SIMD instructions* calculates 32 elements for each instruction.
- Each thread block is executed by a multi-threaded SIMD processor.
- Example: Multiply two vectors of length 8192
  - Each SIMD instruction executes 32 elements at a time
  - Each thread block contains 16 threads of SIMD instructions
  - Grid size = 16 thread blocks
  - Total 16 x 16 x 32 = 8192 elements

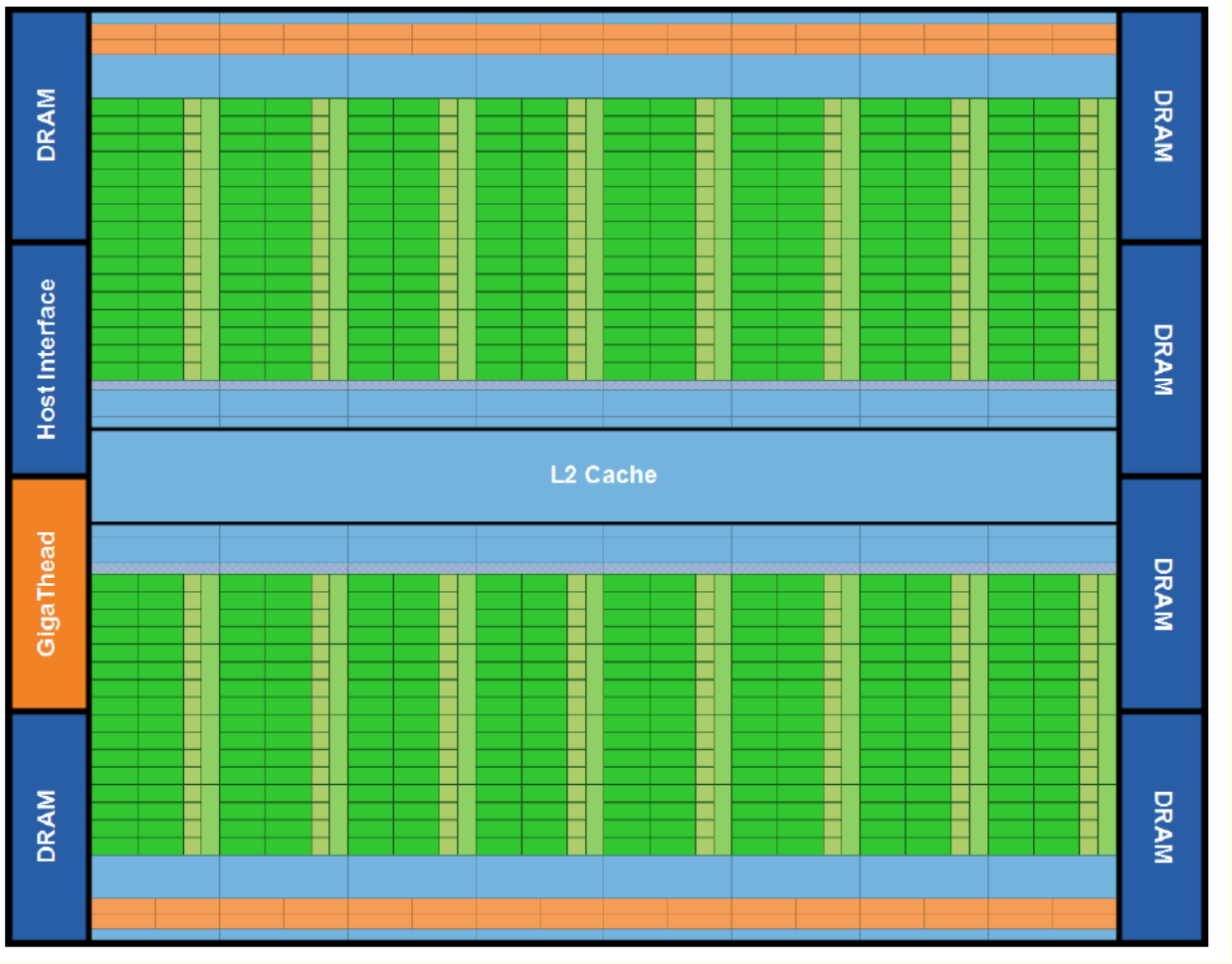
Scheduling Threads

- Two levels of HW Schedulers:
  1. The *Thread Block Scheduler* assigns thread blocks to multithreaded SIMD processors
  2. The *Thread Scheduler*, within a SIMD Processor, assigns each thread to run each clock cycle.

- Fermi architecture:
  1. Giga Thread Scheduler
  2. Dual Warp Scheduler

NVIDIA Fermi GPU (2010)

- Each thread block (warp) is composed of 32 CUDA threads for Fermi (16 CUDA threads for Tesla).
- Fermi GPUs have 16 multithreaded SIMD Processors (called SMs, Streaming Multiprocessors).
- Each SM has:
  - two Warp Schedulers
  - two sets of 16 CUDA cores also called SPs
  - 16 load/store units
  - 4 special function units (SFUs) to execute transcendental instructions (such as sin, cosine, reciprocal and square root).
- Globally there are 512 CUDA cores (512 SPs).

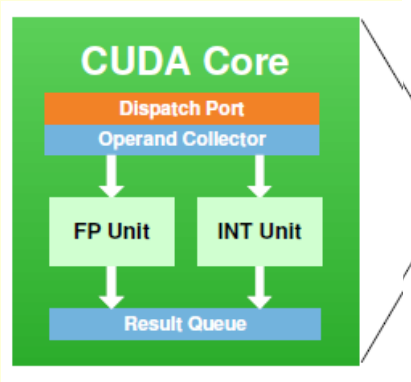


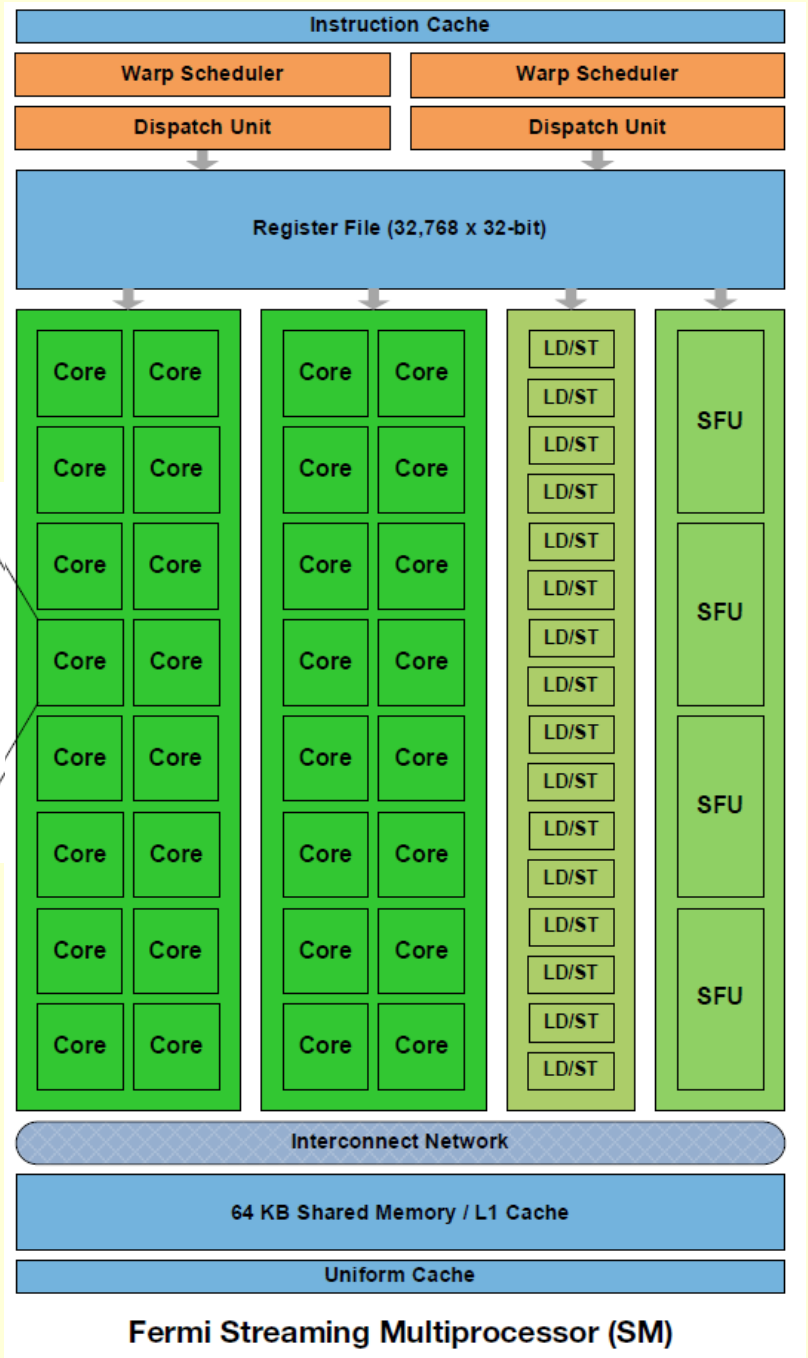
Floorplan of the Fermi GPU composed of **16 multithreaded SIMD Processors** (called SMs, Streaming Multiprocessors) and a common **L2 Cache**.
The **Thread Block Scheduler (Giga Thread)** distributes thread blocks to SMs.
Each SM has its own dual SIMD Thread Scheduler **(Dual Warp Scheduler).**

Fermi Streaming Multiprocessor (SM)

Each SM has:

- two Warp Schedulers
- two sets of 16 CUDA cores (2 x 16 SPs)
- 16 load/store units
- 4 Special Function Units
- 32K registers of 32-bit
- 64 KB shared memory /L1 cache



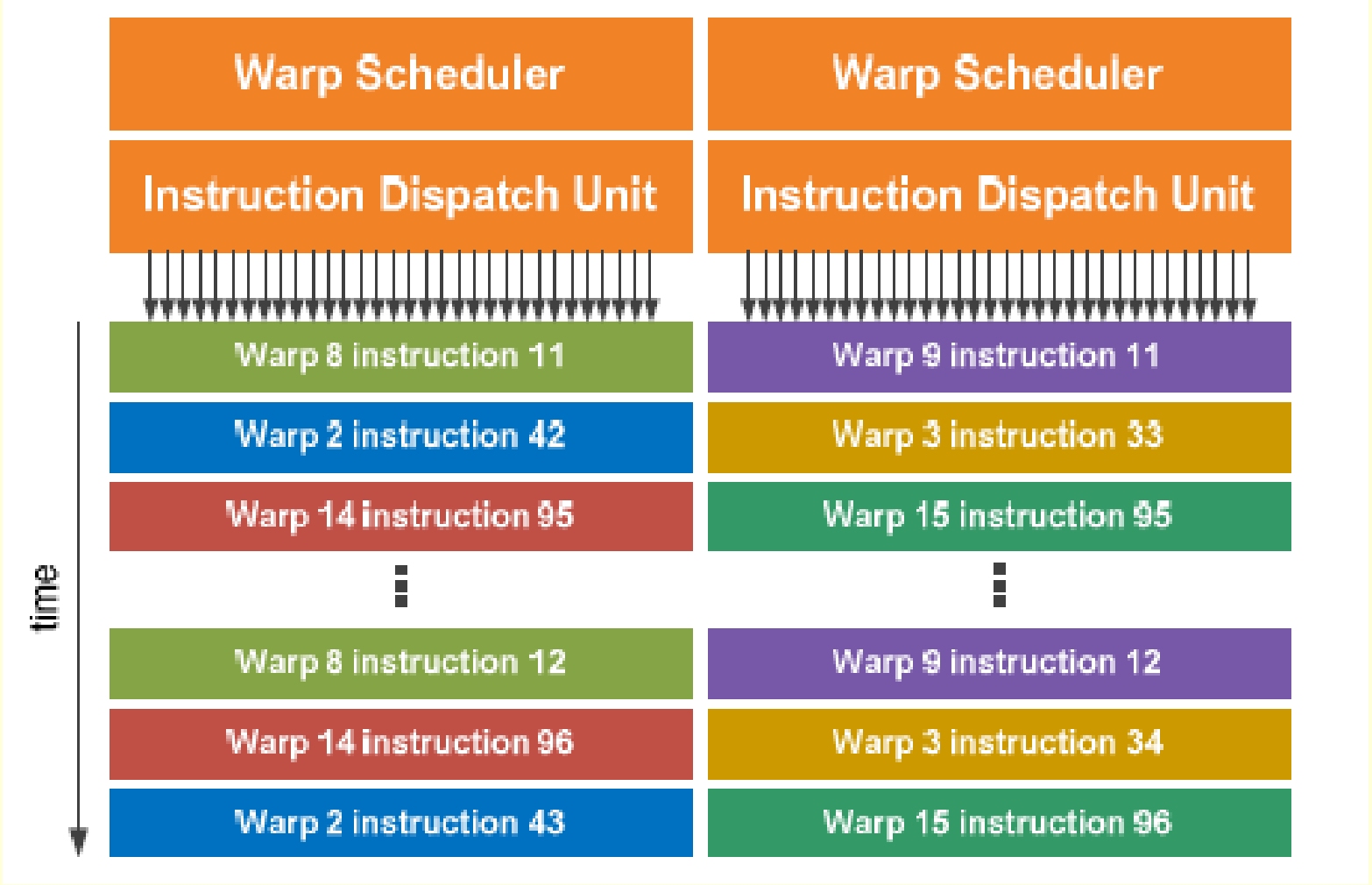


Fermi Streaming Multiprocessor (SM)

NVIDIA Fermi GPU: Scheduling Threads

- Two levels of HW Schedulers:
  1. The ***Giga Thread Engine*** schedules thread blocks to various SMs.
  2. The ***Dual Warp Schedulers*** selects two warps of 32 threads of SIMD instructions and issue one instruction from each warp to its execution units
- The threads are independent so there is no need to check for dependencies within the instruction stream.
- Analogous to a multi-threaded vector processor that can issue vector instructions from two independent threads

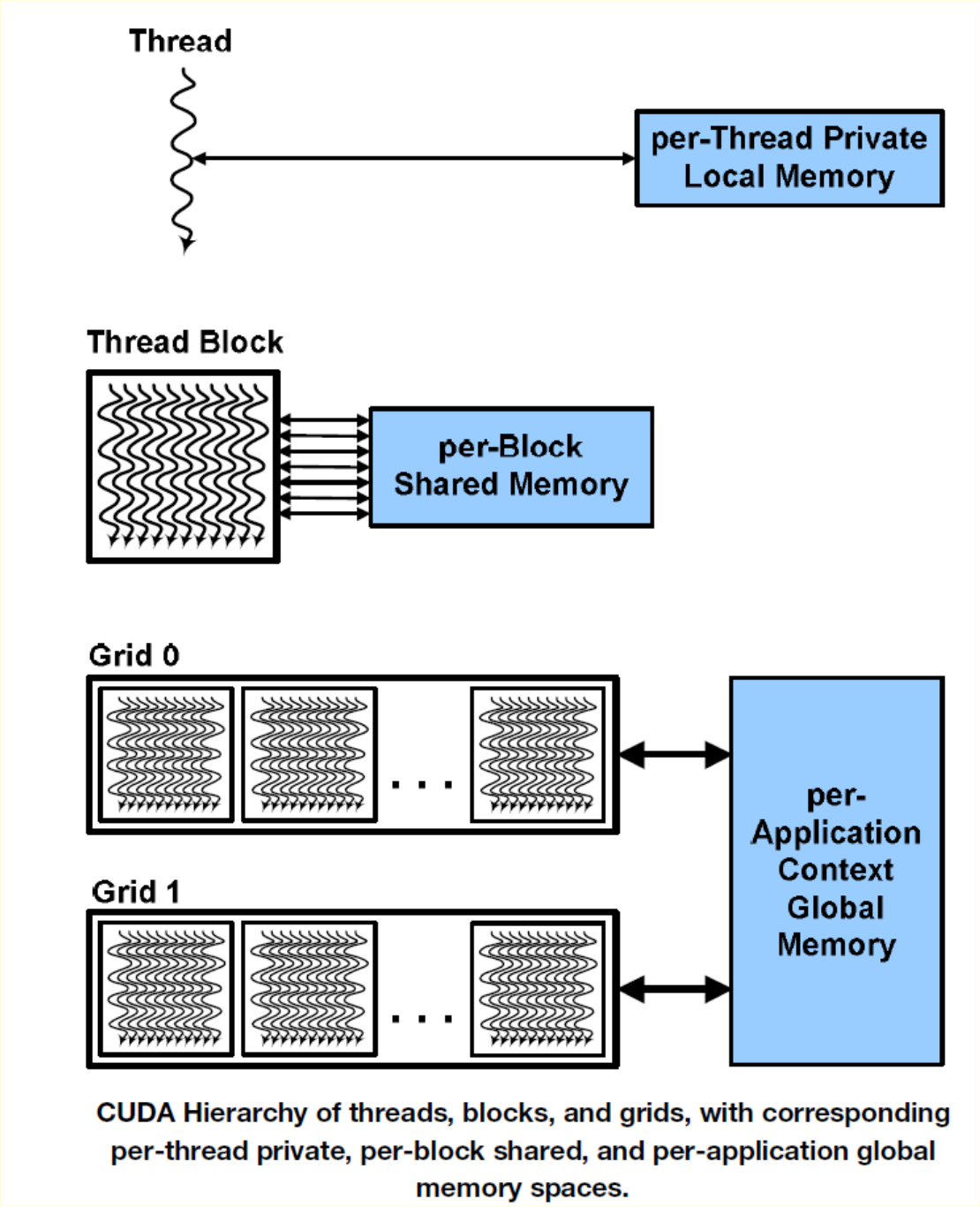
NVIDIA Fermi GPU: Dual Warp Scheduler



Fermi GPU: Fused Multiply-Add

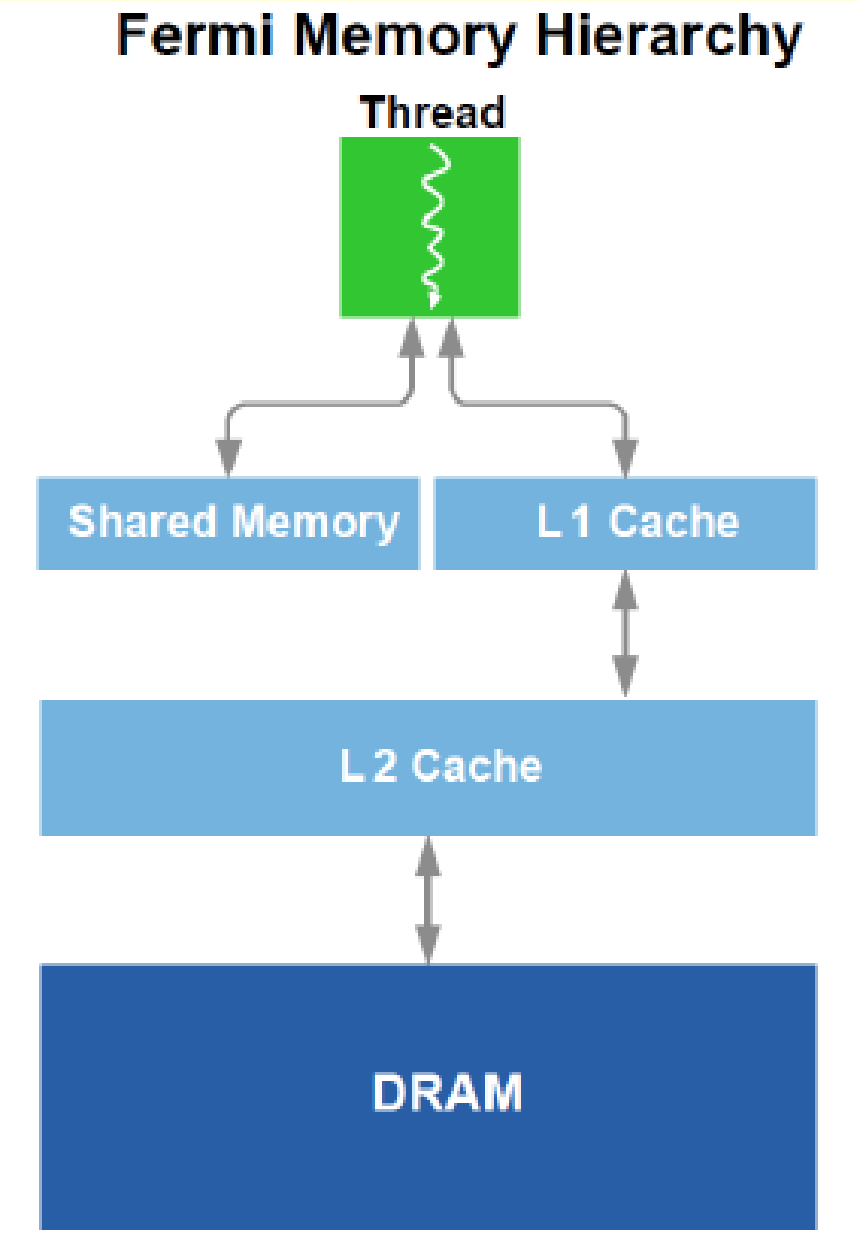
- The Fermi architecture implements the new IEE 754-2008 floating point standard.
- Fused Multiply-Add (FMA) performs multiplication and addition a <= a + (b x c) with a single final rounding step (more accurate than performing the two operations separately).
- Each SP can fulfill up to two single precision FMAs per clock
- Each SM up to 32 single precision (32-bit) FP operations or 16 double precision (64-bit) FP operations

Fermi GPU Memory Hierarchy



CUDA Hierarchy of threads, blocks, and grids, with corresponding per-thread private, per-block shared, and per-application global memory spaces.

Fermi GPU Memory Hierarchy



Fermi GPU Memory Hierarchy

- Each SM has 32,768 registers of 32-bit
  - Divided into lanes
  - Each SIMD thread has access to its own registers and not those of other threads.
  - Each SIMD thread is limited to 64 registers
- Each SM has 64 KB shared memory/L1 cache to be configured as
  - either 48 KB of shared memory among threads (within the same block) + 16 KB of L1 to cache data to individual threads
  - or 16 KB of shared memory and 48 KB of L1 cache.

Fermi GPU Memory Hierarchy

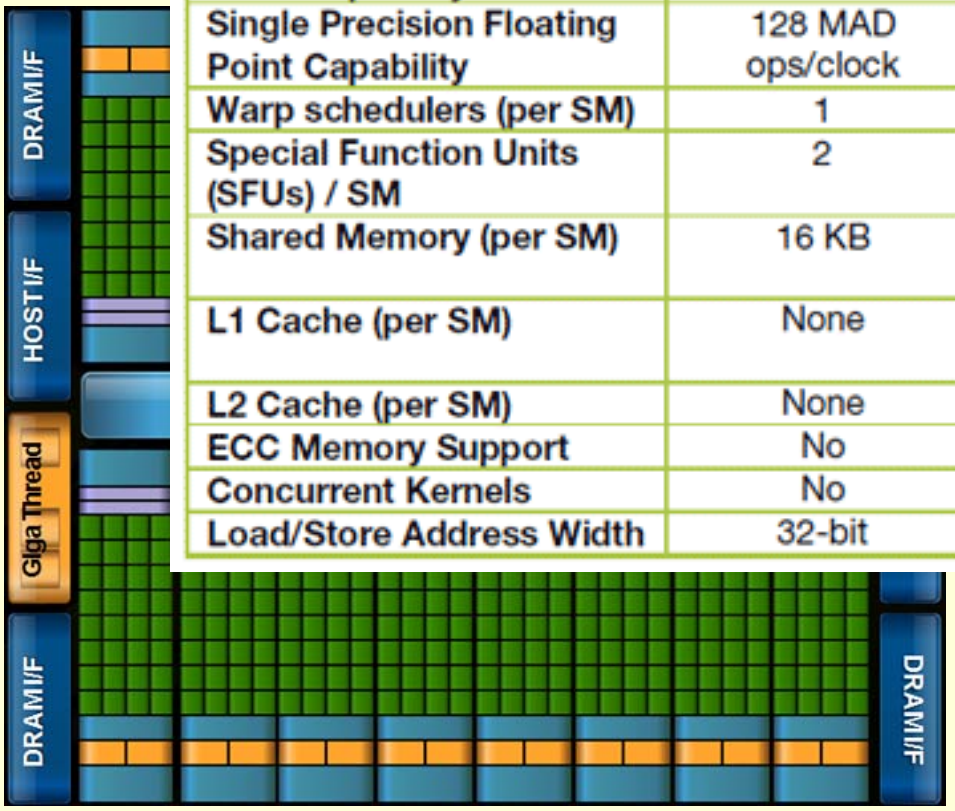
- Local memory can be used to hold ***spilled*** registers, when a thread block requires more register storage than is available on SM registers.
- L2 cache 768 KB unified among the 16 SMs that services all load/store from/to global memory (including copies to/from CPU host) and used for managing access to data shared across thread blocks.
- Global Memory accessible by all threads as well as host CPU. High latency.

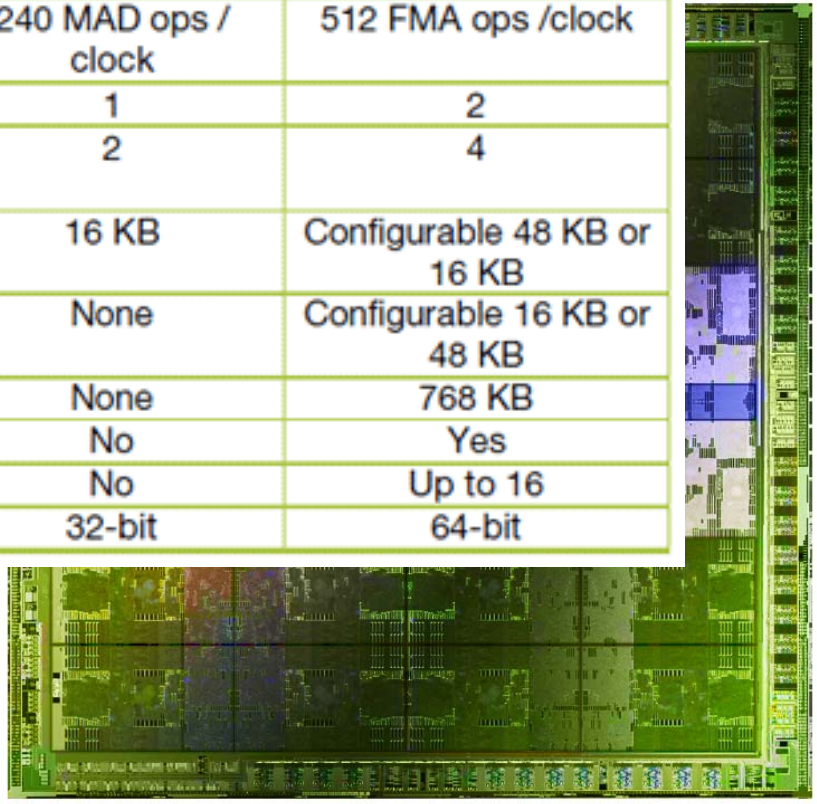
NVIDIA GPU vs. Vector Architectures

- Similarities :
  - Works well with data-level parallel problems
  - Scatter-gather transfers
  - Mask registers
  - Branch hardware uses internal masks
  - 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

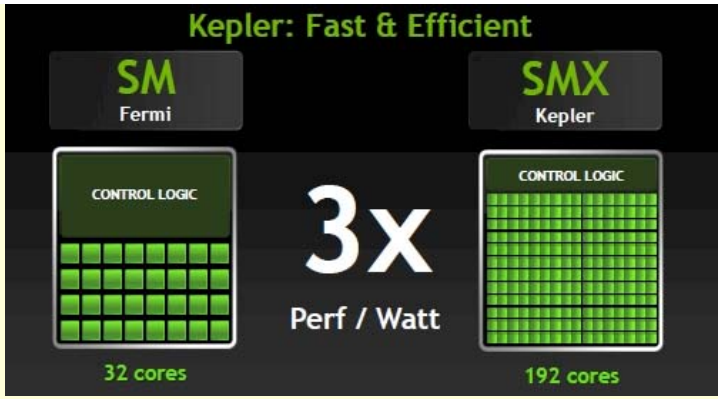
NVIDIA Fermi GPU

GPU	G80	GT200	Fermi
Transistors	681 million	1.4 billion	3.0 billion
CUDA Cores	128	240	512
Double Precision Floating Point Capability	None	30 FMA ops / clock	256 FMA ops /clock
Single Precision Floating Point Capability	128 MAD ops/clock	240 MAD ops / clock	512 FMA ops /clock
Warp schedulers (per SM)	1	1	2
Special Function Units (SFUs) / SM	2	2	4
Shared Memory (per SM)	16 KB	16 KB	Configurable 48 KB or 16 KB
L1 Cache (per SM)	None	None	Configurable 16 KB or 48 KB
L2 Cache (per SM)	None	None	768 KB
ECC Memory Support	No	No	Yes
Concurrent Kernels	No	No	Up to 16
Load/Store Address Width	32-bit	32-bit	64-bit



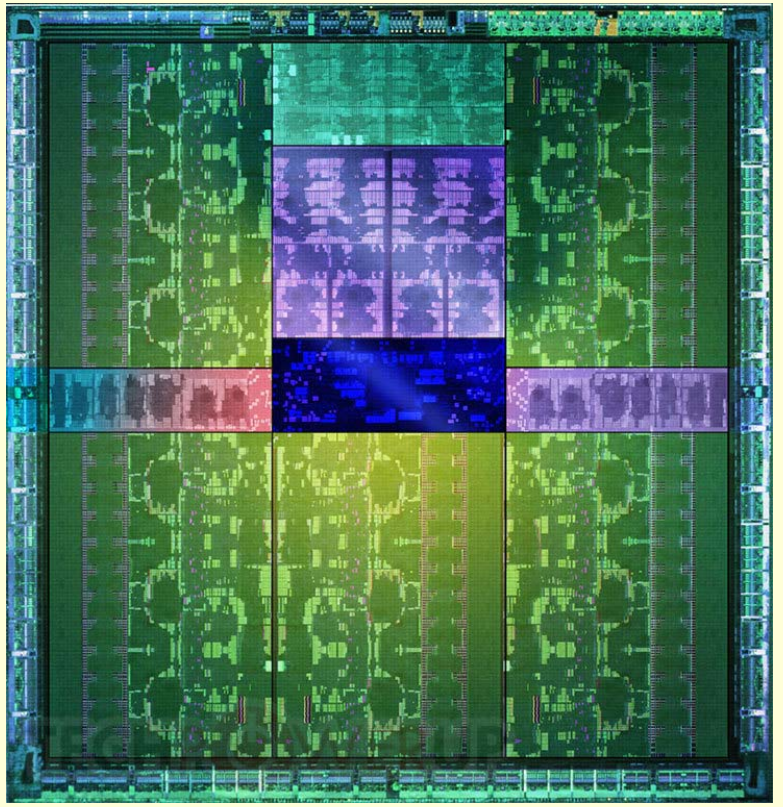


NVIDIA Kepler GPU



Kepler GK110 Architecture

- 7.1B Transistors
- 15 SMX units (2880 cores)
- >1TFLOP FP64
- 1.5MB L2 Cache
- 384-bit GDDR5
- PCI Express Gen3

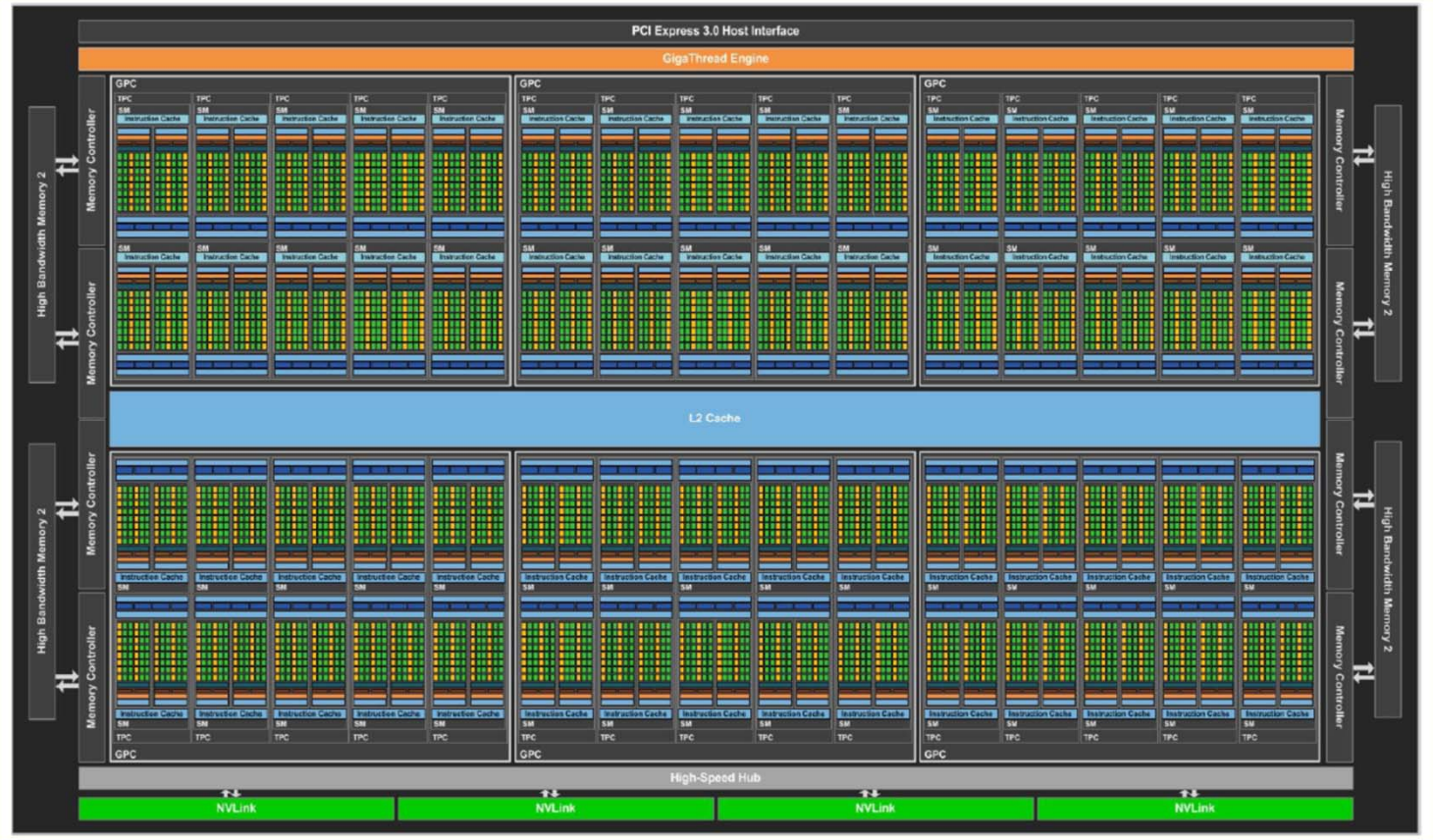


NVIDIA Maxwell GeForce GTX 980

GeForce GTX 980 (Maxwell) vs GTX 680 (Kepler)

GPU	GeForce GTX 680 (Kepler)	GeForce GTX 980 (Maxwell)
SMs	8	16
CUDA Cores	1536	2048
Base Clock	1006 MHz	1126 MHz
GPU Boost Clock	1058 MHz	1216 MHz
GFLOPs	3090	4612[1]
Texture Units	128	128
Texel fill-rate	128.8 Gigatexels/sec	144.1 Gigatexels/sec
Memory Clock	6000 MHz	7000 MHz
Memory Bandwidth	192 GB/sec	224 GB/sec
ROPs	32	64
L2 Cache Size	512KB	2048KB
TDP	195 Watts	165 Watts
Transistors	3.54 billion	5.2 billion
Die Size	294 mm^2	398 mm^2
Manufacturing Process	28-nm	28-nm

NVIDIA Tesla P100 with Pascal GP100 GPU



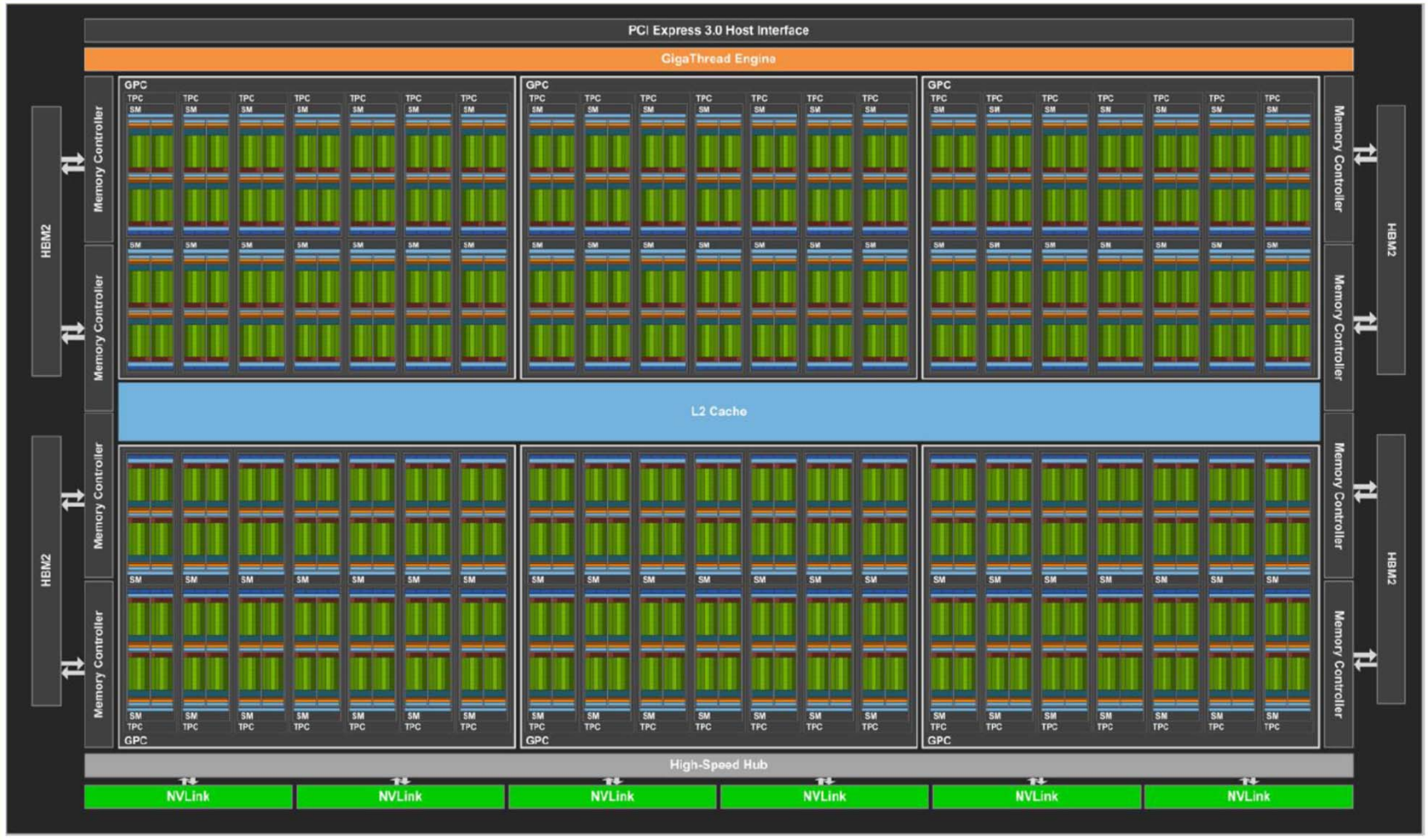
NVIDIA Tesla GP100 (Pascal) with 3584 FP32 CUDA Cores/GPU and 1792 FP64 CUDA Cores/GPUs
Total 15.3 billion transistors in 16 nmFinFET process technology

NVIDIA Tesla V100 SXM2 Module with Volta GV100 GPU

- New Streaming Multiprocessor (SM) Architecture Optimized for **Deep Learning**
- New **Tensor Cores** designed specifically for Deep Learning deliver up to 12x higher peak TFLOPS for training and 6x higher peak TFLOPS for inference
- Volta is 50% more energy efficient than the previous generation Pascal design, enabling major boosts in FP32 and FP64 performance in the same power envelope.
- References:

https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf
https://www.csm.ornl.gov/workshops/openshmem2017/presentations/Rees%20-%20A%20Deep%20Drive%20Into%20NVIDIA%27s%20Vola%20Architecture.PDF

NVIDIA Volta GV100 Full GPU with 84 SM units



The GV100 GPU includes 21.1 billion transistors

Comparison of NVIDIA Tesla GPUs

Tesla Product	Tesla K40	Tesla M40	Tesla P100	Tesla V100
GPU	GK180 (Kepler)	GM200 (Maxwell)	GP100 (Pascal)	GV100 (Volta)
SMs	15	24	56	80
TPCs	15	24	28	40
FP32 Cores / SM	192	128	64	64
FP32 Cores / GPU	2880	3072	3584	5120
FP64 Cores / SM	64	4	32	32
FP64 Cores / GPU	960	96	1792	2560
Tensor Cores / SM	NA	NA	NA	8
Tensor Cores / GPU	NA	NA	NA	640
GPU Boost Clock	810/875 MHz	1114 MHz	1480 MHz	1530 MHz
Peak FP32 TFLOPS[1]	5	6.8	10.6	15.7
Peak FP64 TFLOPS[1]	1.7	.21	5.3	7.8
Peak Tensor TFLOPS[1]	NA	NA	NA	125
Texture Units	240	192	224	320
Memory Interface	384-bit GDDR5	384-bit GDDR5	4096-bit HBM2	4096-bit HBM2
Memory Size	Up to 12 GB	Up to 24 GB	16 GB	16 GB
L2 Cache Size	1536 KB	3072 KB	4096 KB	6144 KB
Shared Memory Size / SM	16 KB/32 KB/48 KB	96 KB	64 KB	Configurable up to 96 KB
Register File Size / SM	256 KB	256 KB	256 KB	256KB
Register File Size / GPU	3840 KB	6144 KB	14336 KB	20480 KB
TDP	235 Watts	250 Watts	300 Watts	300 Watts
Transistors	7.1 billion	8 billion	15.3 billion	21.1 billion
GPU Die Size	551 mm²	601 mm²	610 mm²	815 mm²
Manufacturing Process	28 nm	28 nm	16 nm FinFET+	12 nm FFN

Top500 ranking of the world's most powerful supercomputers (Nov. 2018)

1. **Summit**, IBM-built supercomputer at Dept. of Energy's Oak Ridge National Lab. reached **143.5 PetaFLOPS** Linpack performance with **2,397,824 cores:** Processor **IBM Power9** 22C at 3.1 GHz, **NVIDIA Volta GV100 GPUs,** Mellanox dual-rail EDR InfiniBand network.
2. **Sierra**, IBM-built supercomputer at DOE's Lawrence Livermore National Lab. reached **94.6 PetaFLOPS** with **1,572,480 cores.** Sierra is quite similar to Summit with **IBM Power9** 22C at 3.1 GHz, **NVIDIA Volta GV100 GPUs,** Mellanox dual-rail EDR InfiniBand network.

www.top500.org