EE382N (20): Computer Architecture - Parallelism and Locality
Spring 2015

Lecture 09 – GPUs (II)

Mattan Erez

The University of Texas at Austin

Recap

Streaming model

1. Use many "slimmed down cores" to run in parallel

2. Pack cores full of ALUs (by sharing instruction stream across groups of fragments)
   - Option 1: Explicit SIMD vector instructions
   - Option 2: Implicit sharing managed by hardware

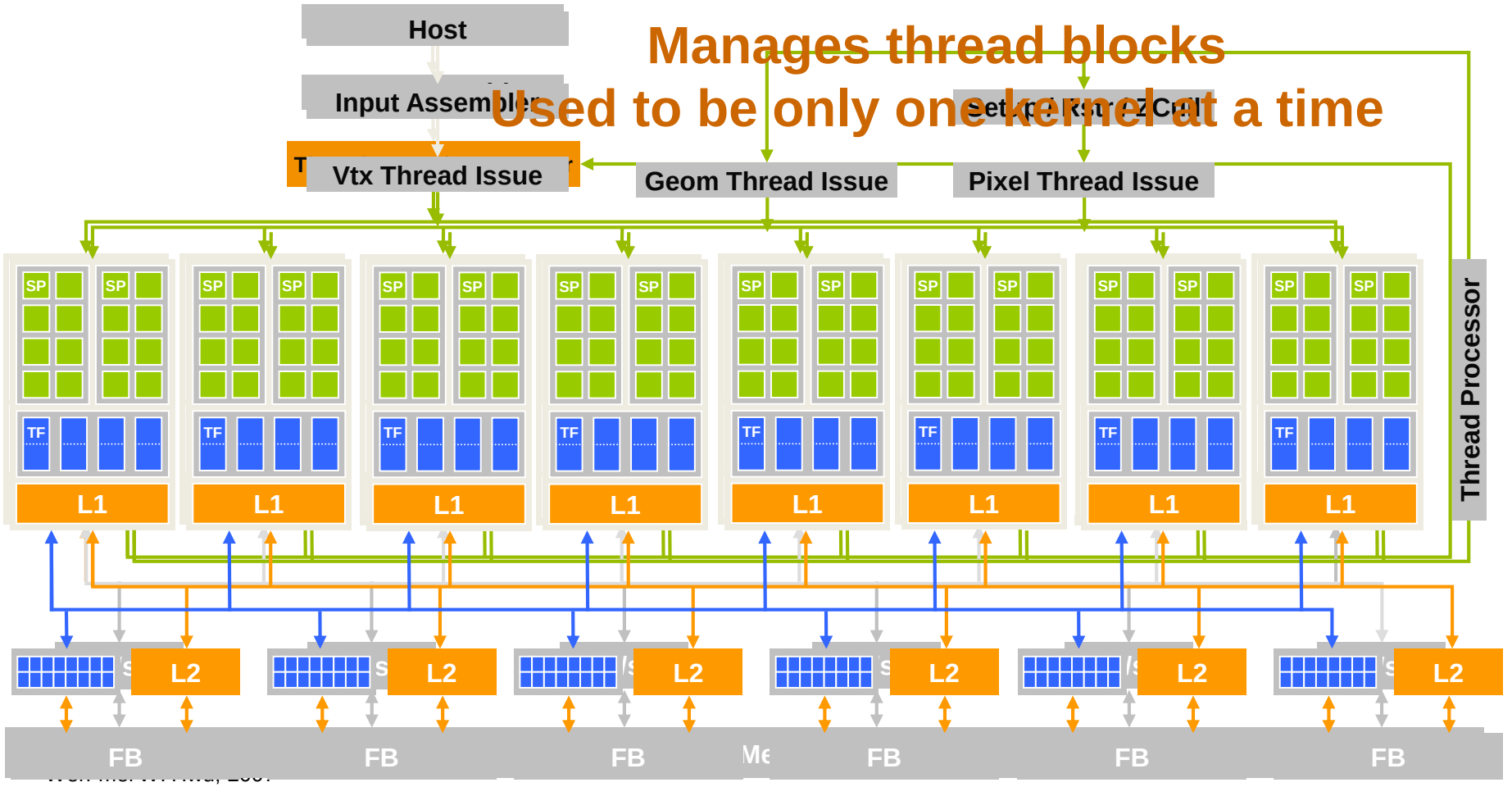
3. Avoid latency stalls by interleaving execution of many groups of fragments
   - When one group stalls, work on another group

Make the Compute Core The Focus of the Architecture

- The future of GPUs is programmable processing
- So – build the architecture around the processor

Processors execute computing threads

Alternative operating mode specifically for computing



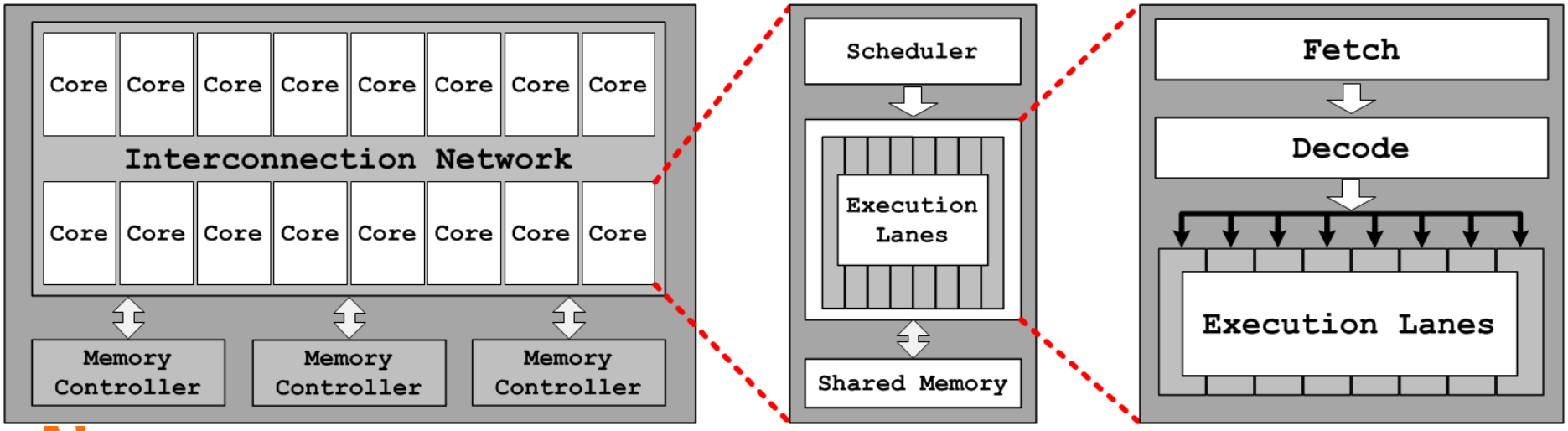
Manages thread blocks

Used to be only one kernel at a time

Graphic Processing Units (GPUs)

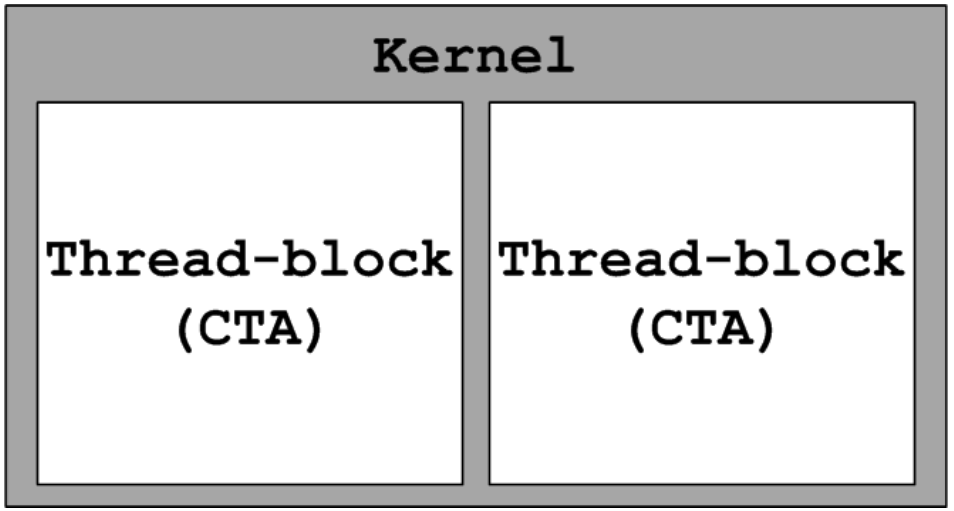
- General-purpose many-core accelerators
  - Use simple in-order shader cores in 10s / 100s numbers
  - Shader core == **S**treaming **M**ultiprocessor (**SM**) in NVIDIA GPUs

- Scalar frontend (fetch & decode) + parallel backend
  - Amortizes the cost of frontend and control



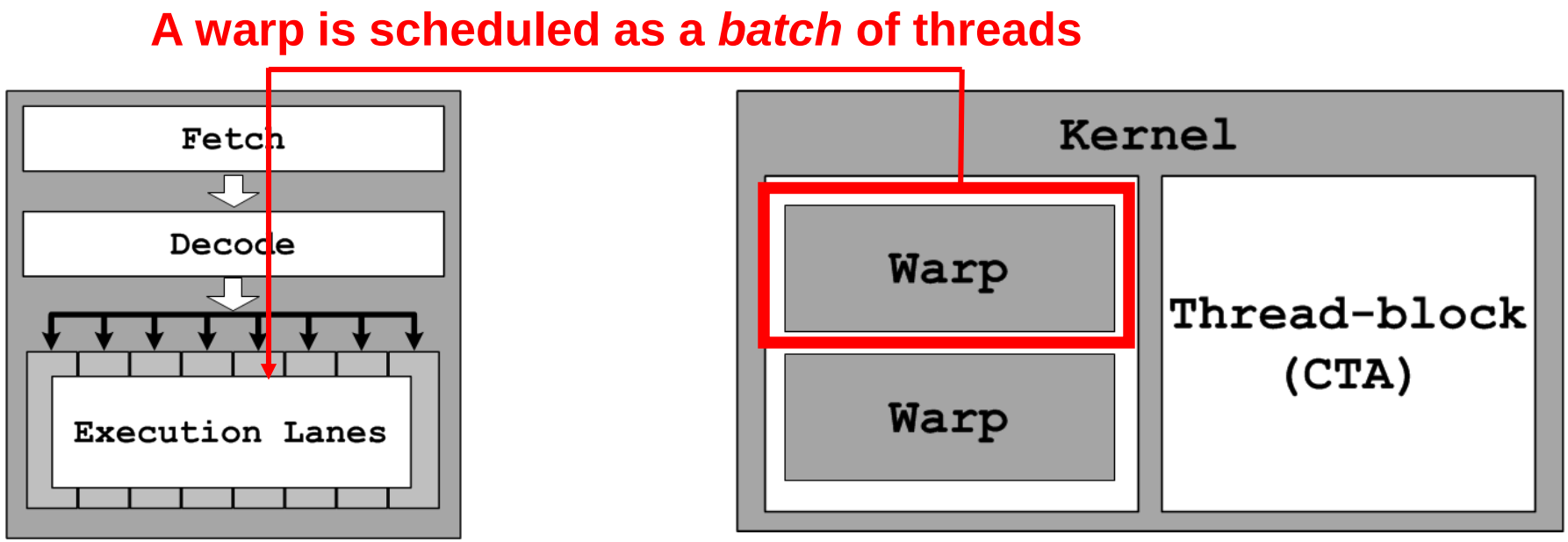
CUDA exposes hierarchy of data-parallel threads

- **SPMD model**: single ***kernel*** executed by all *scalar* threads
- **Kernel / Thread-block**
  - Multiple **thread-blocks** (cooperative-thread-arrays (**CTA**s)) compose a **kernel**



CUDA exposes hierarchy of data-parallel threads

- **SPMD model**: single ***kernel*** executed by all *scalar* threads
- **Kernel / Thread-block / *Warp* / *Thread***
  - Multiple **warps** compose a **thread-block**
  - Multiple **threads (32)** compose a warp



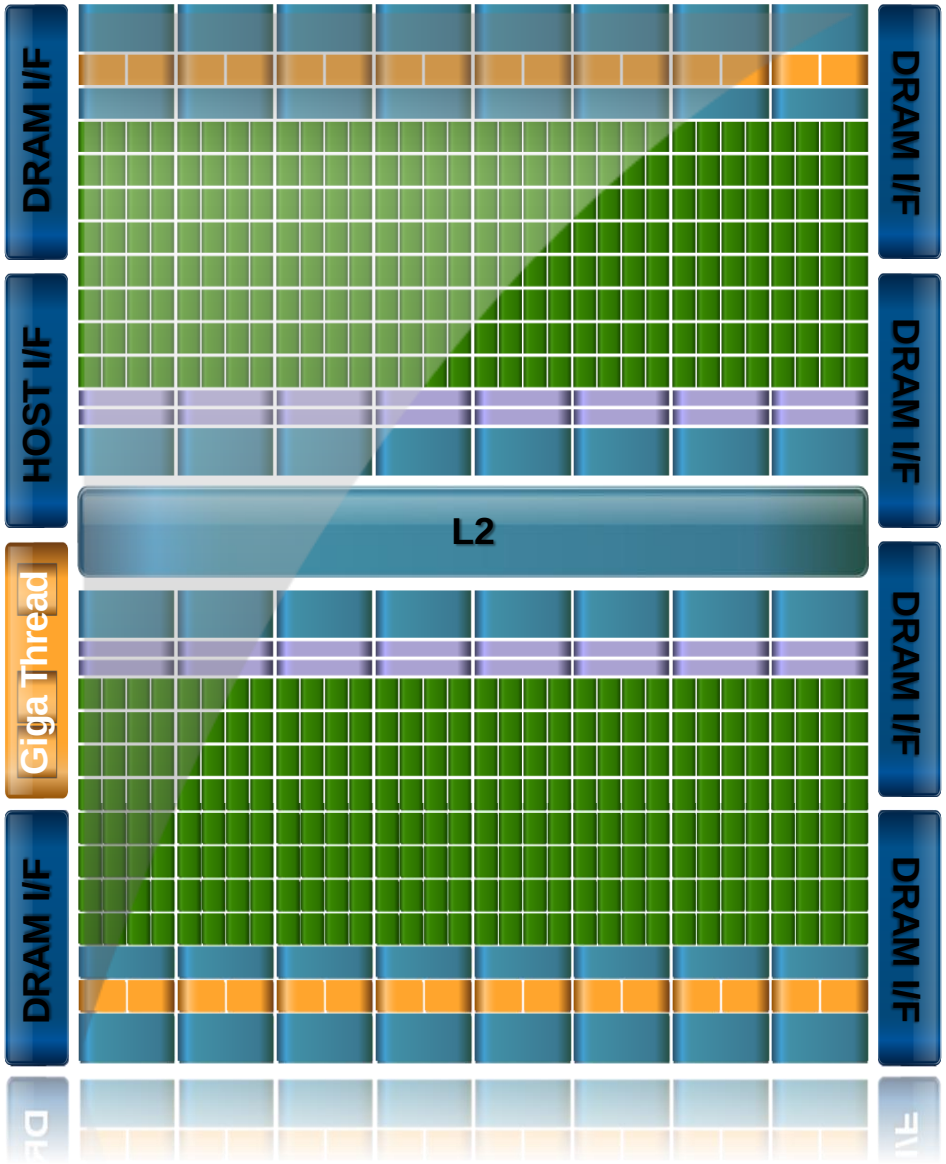
SIMT for balanced *programmability* and *HW-eff.*

- **SIMT**: Single-Instruction Multiple-**Thread**
  - Programmer writes code to be executed by ***scalar*** threads
  - Underlying hardware uses ***vector*** SIMD pipelines (lanes)
    - HW/SW groups ***scalar*** threads to execute in ***vector*** lanes

- Enhanced programmability using SIMT
  - Hardware/software supports ***conditional branches***
    - Each thread can follow its *own control flow*
  - ***Per-thread load/store*** instructions are also supported
    - Each thread can reference *arbitrary address regions*

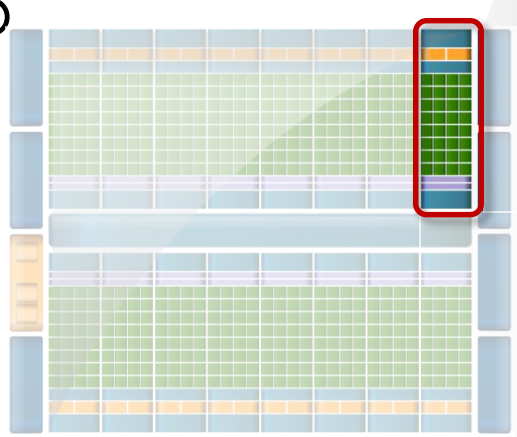
Older GPU, but high-level same

- Expand performance sweet spot of the GPU
  - Caching
  - Concurrent kernels
  - FP64
  - 512 cores
  - GDDR5 memory
- Bring more users, more applications to the GPU
  - C++
  - Visual Studio Integration
  - ECC



Streaming Multiprocessor (SM)

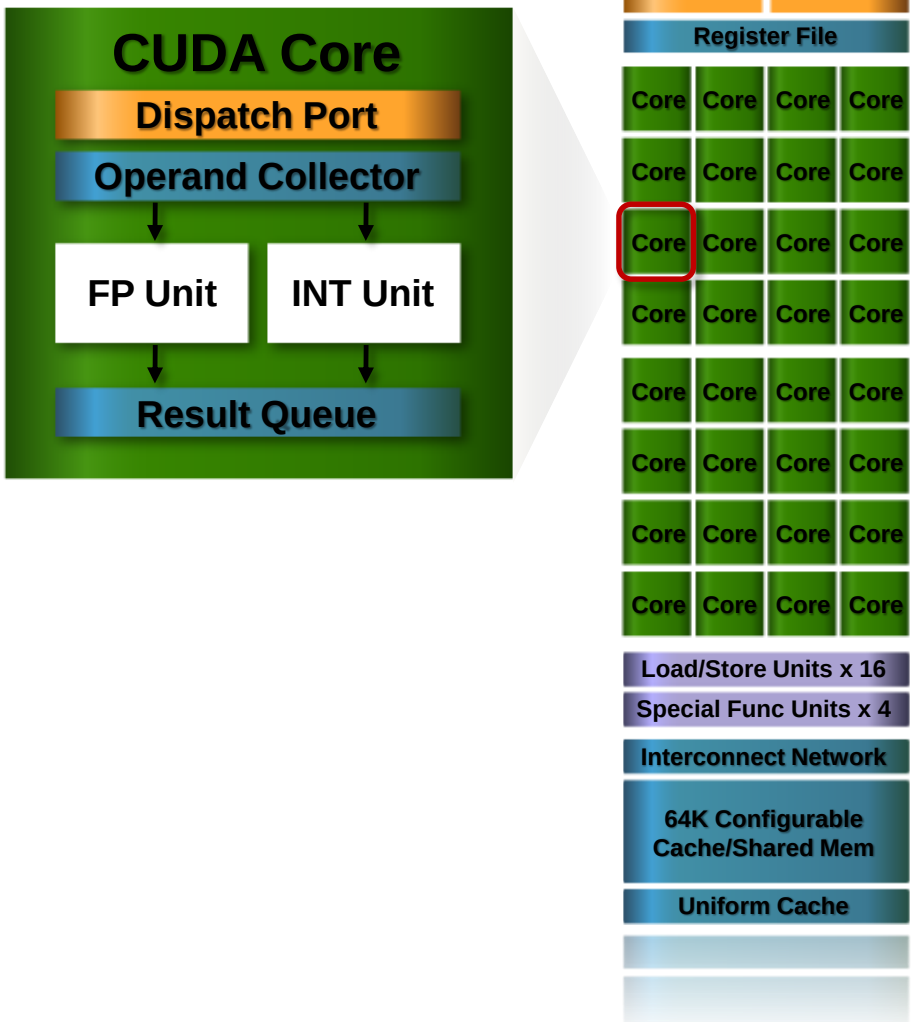
- Objective – optimize for GPU computing
  - New ISA
  - Revamp issue / control flow
  - New CUDA core architecture
- 16 SMs per Fermi chip
- 32 cores per SM (512 total)
- 64KB of configurable L1$ / shared memory



	FP32	FP64	INT	SFU	LD/ST
Ops / clk	32	16	32	4	16

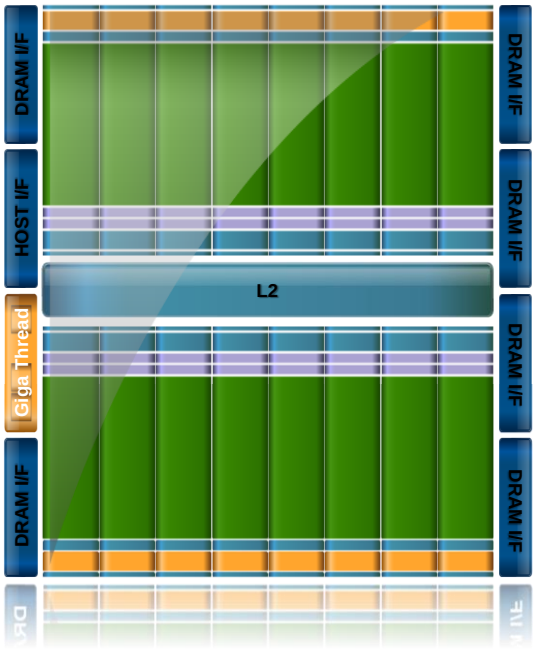
SM Microarchitecture

- New IEEE 754-2008 arithmetic standard
- Fused Multiply-Add (FMA) for SP & DP
- New integer ALU optimized for 64-bit and extended precision ops



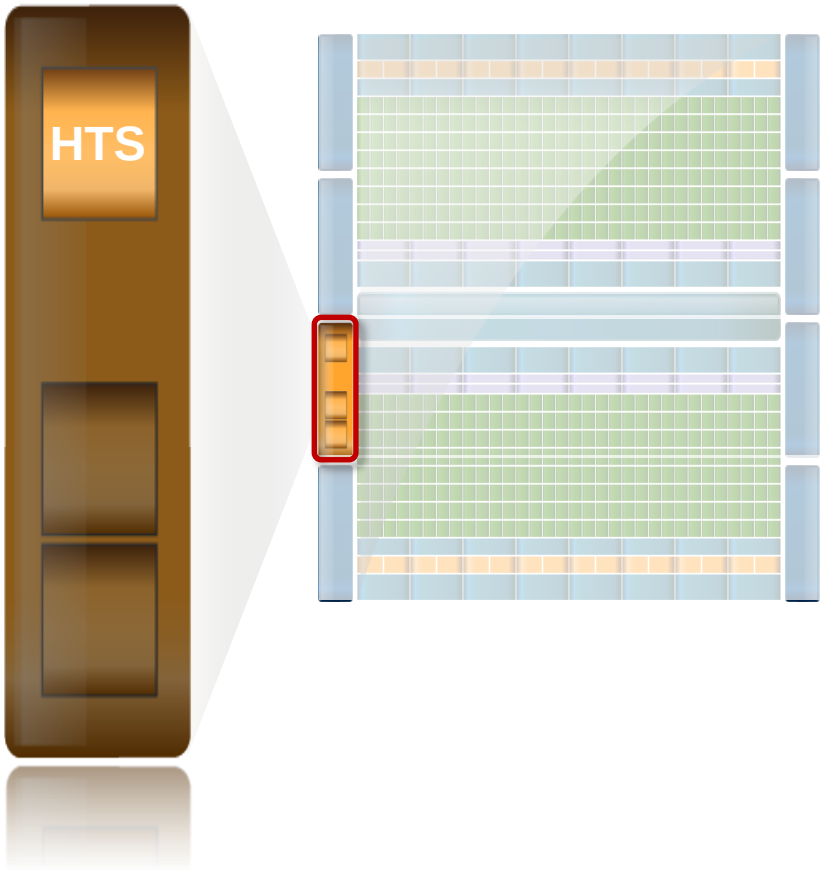
Memory Hierarchy

- True cache hierarchy + on-chip shared RAM
  - On-chip shared memory: good fit for regular memory access
    - dense linear algebra, image processing, …
  - Caches: good fit for irregular or unpredictable memory access
    - ray tracing, sparse matrix multiply, physics …
- Separate L1 Cache for each SM (16/48 KB)
  - Improves bandwidth and reduces latency
- Unified L2 Cache for all SMs (768 KB)
  - Fast, coherent data sharing across all cores in the GPU



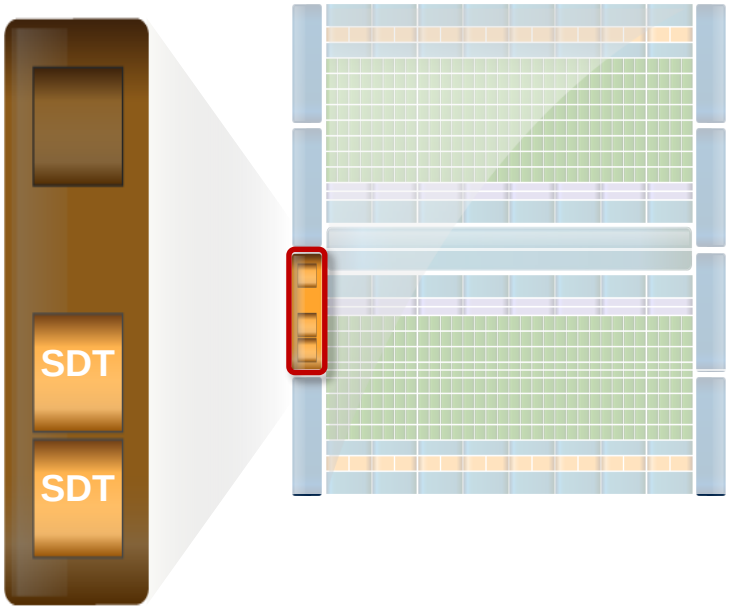
GigaThread™ Hardware Thread Scheduler

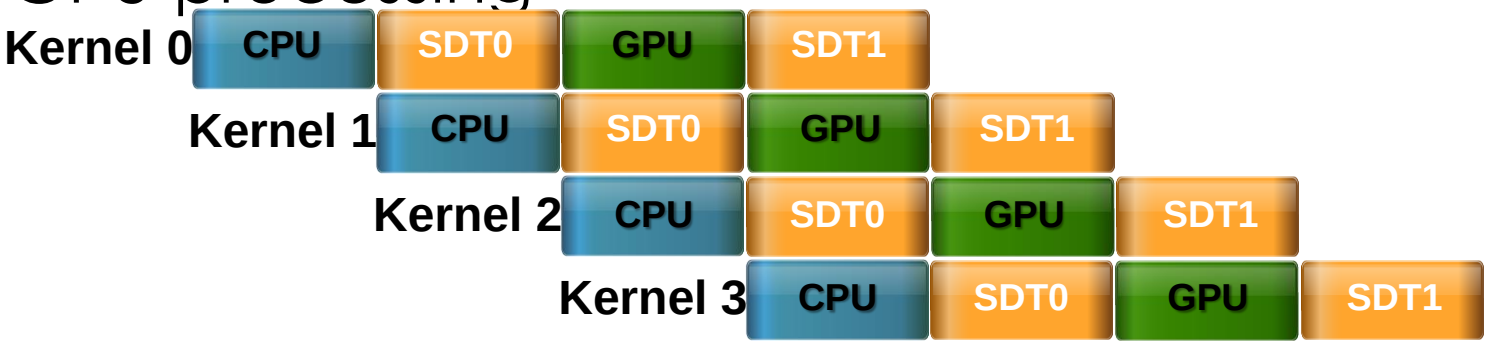
- Hierarchically manages tens of thousands of simultaneously active threads
- 10x faster context switching on Fermi
- Overlapping kernel execution



GigaThread Streaming Data Transfer Engine

- Dual DMA engines
- Simultaneous CPU→GPU and GPU→CPU data transfer
- Fully overlapped with CPU/GPU processing





Thread Life Cycle in HW

- Kernel is launched on the SPA
  - Kernels known as ***grids*** of thread blocks
- Thread Blocks are serially distributed to all the SM's
  - Potentially >1 Thread Block per SM
  - At least 96 threads per block
- Each SM launches Warps of Threads
  - 2 levels of parallelism
- SM schedules and executes Warps that are ready to run
- As Warps and Thread Blocks complete, resources are freed
  - SPA can distribute more Thread Blocks