King Abdullah University of Science and Technology

KAUST

CS 380 - GPU and GPGPU Programming
Lecture 12: GPU Compute APIs 1

Markus Hadwiger, KAUST

Reading Assignment #7 (until Oct 19)

Read (required):

- Read **`https://en.wikipedia.org/wiki/Instruction_pipelining`**
- CUDA NVCC doc (CUDA SDK: **`CUDA_Compiler_Driver_NVCC.pdf`**)
  Read Chapters 1 – 3; Chapter 5; get an overview of the rest
- PTX Instruction Set Architecture 7.1 in CUDA SDK (**`ptx_isa_7.1.pdf`**)
  Read Chapters 1 – 3; get an overview of Chapter 12;
  browse through the other chapters to get a feeling for what PTX looks like
- Look at CUDA SASS in CUDA SDK: **`CUDA_Binary_Utilities.pdf`**, Chapter 4

Read (optional):

- Inline PTX Assembly in CUDA (CUDA SDK: **`Inline_PTX_Assembly.pdf`**)
- Dissecting GPU Architecture through Microbenchmarking:

  Volta: **`https://arxiv.org/abs/1804.06826`**
  Turing: **`https://arxiv.org/abs/1903.07486`**

`https://developer.download.nvidia.com/video/gputechconf/gtc/2019/presentation/s9839-discovering-the-turing-t4-gpu-architecture-with-microbenchmarks.pdf`

NVIDIA Volta Architecture

2017/2018

NVIDIA Volta SM

Multiprocessor: SM

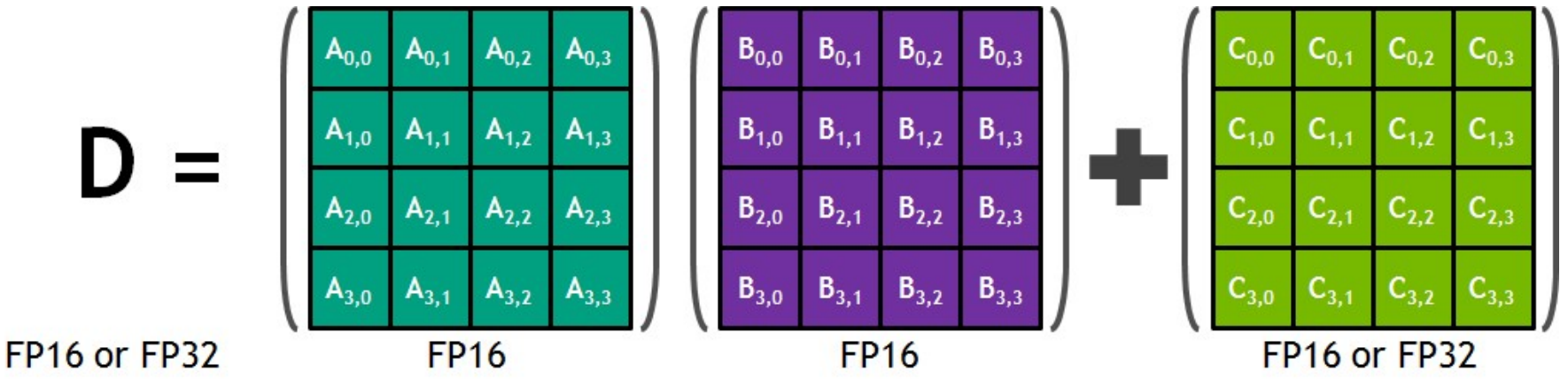
- 64 FP32 + INT32 cores
- 32 FP64 cores
- 8 tensor cores (FP16/FP32 mixed-precision)

4 partitions inside SM

- 16 FP32 + INT32 cores each
- 8 FP64 cores each
- 8 LD/ST units each
- 2 tensor cores each
- Each has: warp scheduler, dispatch unit, register file

Tensor Cores

Mixed-precision, fast matrix-matrix multiply and accumulate



From this, build larger sizes, higher dimensionalities, ...

[+Tensor cores on later architectures add more data types/precisions!]

NVIDIA Ampere Architecture
2020

GA100, GA102, GA104, ...
(A100, RTX 3070, RTX 3080, RTX 3090, ...)

NVIDIA GA100 SM

Multiprocessor: SM

- 64 FP32 + 64 INT32 cores
- 32 FP64 cores
- 4 3rd gen tensor cores
- 1 2nd gen RT (ray tracing) core

4 partitions inside SM

- 16 FP32 + 16 INT32 cores
- 8 FP64 cores
- 8 LD/ST units each
- 1 3rd gen tensor core each
- Each has: warp scheduler, dispatch unit, 16K register file

NVIDIA GA10x SM

Multiprocessor: SM

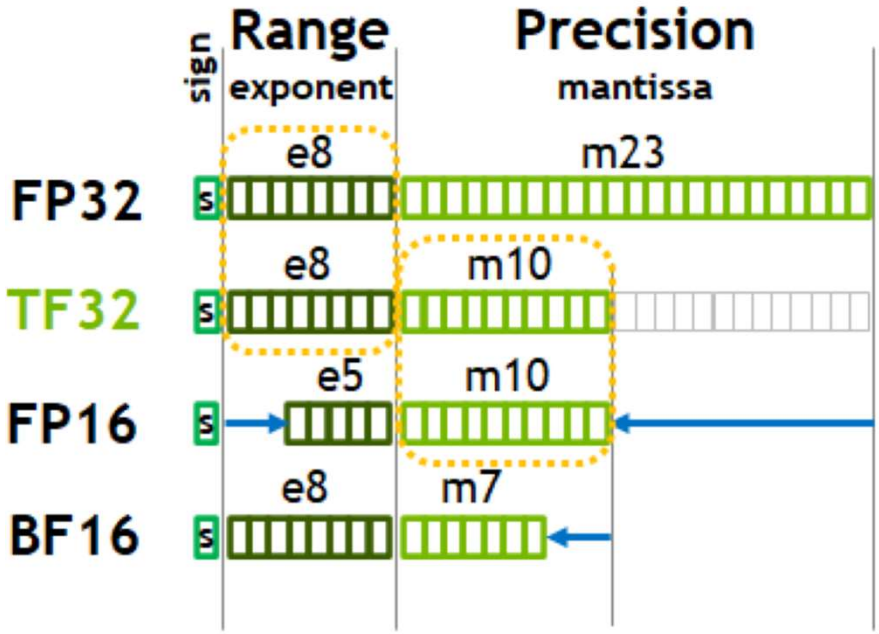
- 128 (64+64) FP32 + 64 INT32 cores
- 2 (!) FP64 cores
- 4 3rd gen tensor cores
- 1 2nd gen RT (ray tracing) core

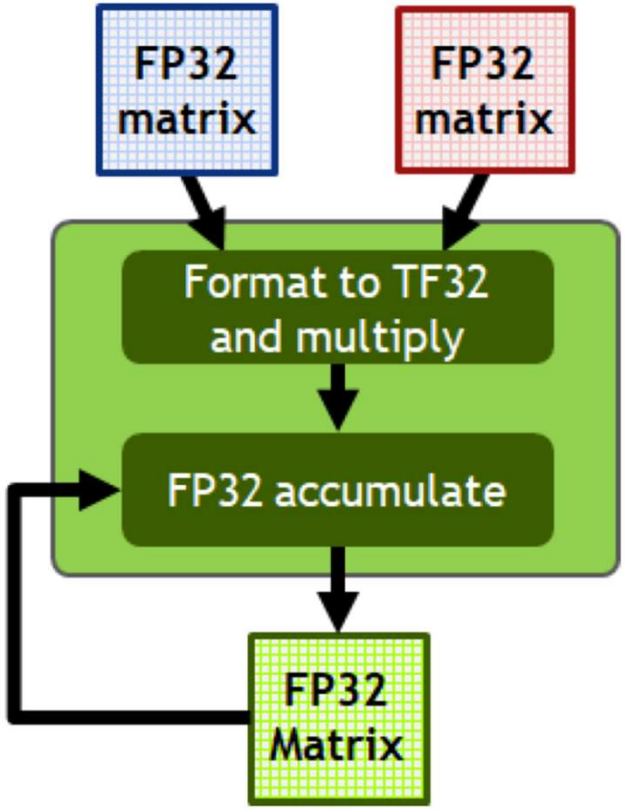
4 partitions inside SM

- 16+16 FP32 + 16 INT32 cores
- 4 LD/ST units each
- 1 3rd gen tensor core each
- Each has: warp scheduler, dispatch unit, 16K register file

Tensor Cores: Many Mixed Precision Options

New in Ampere: TF32, BF16, FP64





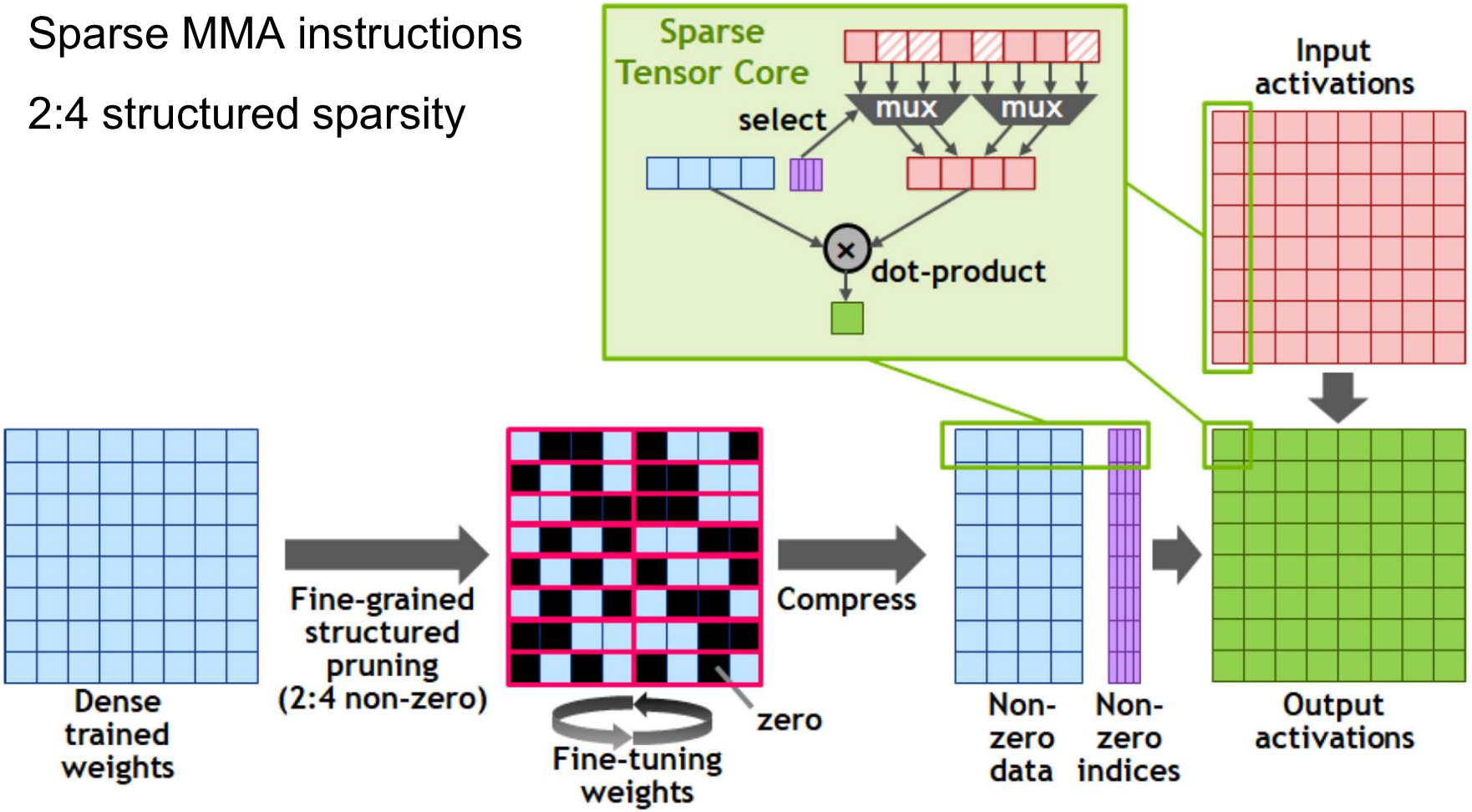
plus FP64 (new in Ampere; GA100 only)

plus INT4/INT8/binary data types (already introduced in Turing)

Tensor Cores: Sparsity Support

Sparse MMA instructions

2:4 structured sparsity



CUDA
Compute Capabilities

Compute Capab. – 2.0

- 1024 threads / block
- More threads / SM
- 32K registers / SM
- New synchronization functions

	Compute Capability				
Feature Support (Unlisted features are supported for all compute capabilities)	**1.0**	**1.1**	**1.2**	**1.3**	**2.0**
Integer atomic functions operating on 32-bit words in global memory (Section B.10)	No	yes			
Integer atomic functions operating on 64-bit words in global memory (Section B.10)	No		Yes		
Integer atomic functions operating on 32-bit words in shared memory (Section B.10)					
Warp vote functions (Section B.11)					
Double-precision floating-point numbers	No			Yes	
Floating-point atomic addition operating on 32-bit words in global and shared memory (Section B.10)	No				Yes
__ballot() (Section B.11)					
__threadfence_system() (Section B.5)					
__syncthreads_count(), __syncthreads_and(), __syncthreads_or() (Section B.6)					

	Compute Capability				
Technical Specifications	**1.0**	**1.1**	**1.2**	**1.3**	**2.0**
Maximum x- or y-dimension of a grid of thread blocks	65535				
Maximum number of threads per block	512				1024
Maximum x- or y-dimension of a block	512				1024
Maximum z-dimension of a block	64				
Warp size	32				
Maximum number of resident blocks per multiprocessor	8				
Maximum number of resident warps per multiprocessor	24		32		48
Maximum number of resident threads per multiprocessor	768		1024		1536
Number of 32-bit registers per multiprocessor	8 K		16 K		32 K
Maximum amount of shared memory per multiprocessor	16 KB				48 KB
Number of shared memory banks	16				32
Amount of local memory per thread	16 KB				512 KB
Constant memory size	64 KB				
Cache working set per multiprocessor for constant memory	8 KB				
Cache working set per multiprocessor for texture memory	Device dependent, between 6 KB and 8 KB				
Maximum width for a 1D texture reference bound to a CUDA array	8192				32768
Maximum width for a 1D texture reference bound to linear memory	2^{27}				
Maximum width and height for a 2D texture reference bound to linear memory or a CUDA array	65536 x 32768				65536 x 65536
Maximum width, height, and depth for a 3D texture reference bound to linear memory or a CUDA array	2048 x 2048 x 2048				4096 x 4096 x 4096
Maximum number of instructions per kernel	2 million				

Compute Capabilities 2.0 – 3.5 (Fermi – Kepler)

	FERMI GF100	FERMI GF104	KEPLER GK104	KEPLER GK110
Compute Capability	2.0	2.1	3.0	3.5
Threads / Warp	32	32	32	32
Max Warps / Multiprocessor	48	48	64	64
Max Threads / Multiprocessor	1536	1536	2048	2048
Max Thread Blocks / Multiprocessor	8	8	16	16
32-bit Registers / Multiprocessor	32768	32768	65536	65536
Max Registers / Thread	63	63	63	255
Max Threads / Thread Block	1024	1024	1024	1024
Shared Memory Size Configurations (bytes)	16K 48K	16K 48K	16K 32K 48K	16K 32K 48K
Max X Grid Dimension	2^16-1	2^16-1	2^32-1	2^32-1
Hyper-Q	No	No	No	Yes
Dynamic Parallelism	No	No	No	Yes

Compute Capability of Fermi and Kepler GPUs

Compute Capab. 5.x (Maxwell, Part 1)

Maxwell

- GM107: 5.0
- GM204: 5.2

	Compute Capability					
Technical Specifications	**2.x**	**3.0, 3.2**	**3.5**	**3.7**	**5.0**	**5.2**
Maximum dimensionality of grid of thread blocks	3					
Maximum x-dimension of a grid of thread blocks	65535	$2^{31}-1$				
Maximum y- or z-dimension of a grid of thread blocks	65535					
Maximum dimensionality of thread block	3					
Maximum x- or y-dimension of a block	1024					
Maximum z-dimension of a block	64					
Maximum number of threads per block	1024					
Warp size	32					
Maximum number of resident blocks per multiprocessor	8	16			32	
Maximum number of resident warps per multiprocessor	48	64				
Maximum number of resident threads per multiprocessor	1536	2048				

Compute Capab. 5.x (Maxwell, Part 2)

Maxwell

- GM107: 5.0
- GM204: 5.2

	Compute Capability					
Technical Specifications	**2.x**	**3.0, 3.2**	**3.5**	**3.7**	**5.0**	**5.2**
Number of 32-bit registers per multiprocessor	32 K	64 K		128 K	64 K	
Maximum number of 32-bit registers per thread block	32 K	64 K				
Maximum number of 32-bit registers per thread	63		255			
Maximum amount of shared memory per multiprocessor	48 KB			112 KB	64 KB	96 KB
Maximum amount of shared memory per thread block	48 KB					
Number of shared memory banks	32					
Amount of local memory per thread	512 KB					
Constant memory size	64 KB					
Cache working set per multiprocessor for constant memory	8 KB				10 KB	
Cache working set per multiprocessor for texture memory	12 KB	Between 12 KB and 48 KB				

Compute Capabilities 3.5 – 7.0 (Kepler – Volta)

GPU	Kepler GK180	Maxwell GM200	Pascal GP100	Volta GV100
Compute Capability	3.5	5.2	6.0	7.0
Threads / Warp	32	32	32	32
Max Warps / SM	64	64	64	64
Max Threads / SM	2048	2048	2048	2048
Max Thread Blocks / SM	16	32	32	32
Max 32-bit Registers / SM	65536	65536	65536	65536
Max Registers / Block	65536	32768	65536	65536
Max Registers / Thread	255	255	255	255*
Max Thread Block Size	1024	1024	1024	1024
FP32 Cores / SM	192	128	64	64
# of Registers to FP32 Cores Ratio	341	512	1024	1024
Shared Memory Size / SM	16 KB/32 KB/48 KB	96 KB	64 KB	Configurable up to 96 KB

Compute Capabilities – 8.0 (Ampere)

GPU Features	NVIDIA Tesla P100	NVIDIA Tesla V100	NVIDIA A100
GPU Codename	GP100	GV100	GA100
GPU Architecture	NVIDIA Pascal	NVIDIA Volta	NVIDIA Ampere
Compute Capability	6.0	7.0	8.0
Threads / Warp	32	32	32
Max Warps / SM	64	64	64
Max Threads / SM	2048	2048	2048
Max Thread Blocks / SM	32	32	32
Max 32-bit Registers / SM	65536	65536	65536
Max Registers / Block	65536	65536	65536
Max Registers / Thread	255	255	255
Max Thread Block Size	1024	1024	1024
FP32 Cores / SM	64	64	64
Ratio of SM Registers to FP32 Cores	1024	1024	1024
Shared Memory Size / SM	64 KB	Configurable up to 96 KB	Configurable up to 164 KB

GPU Compute APIs

NVIDIA CUDA

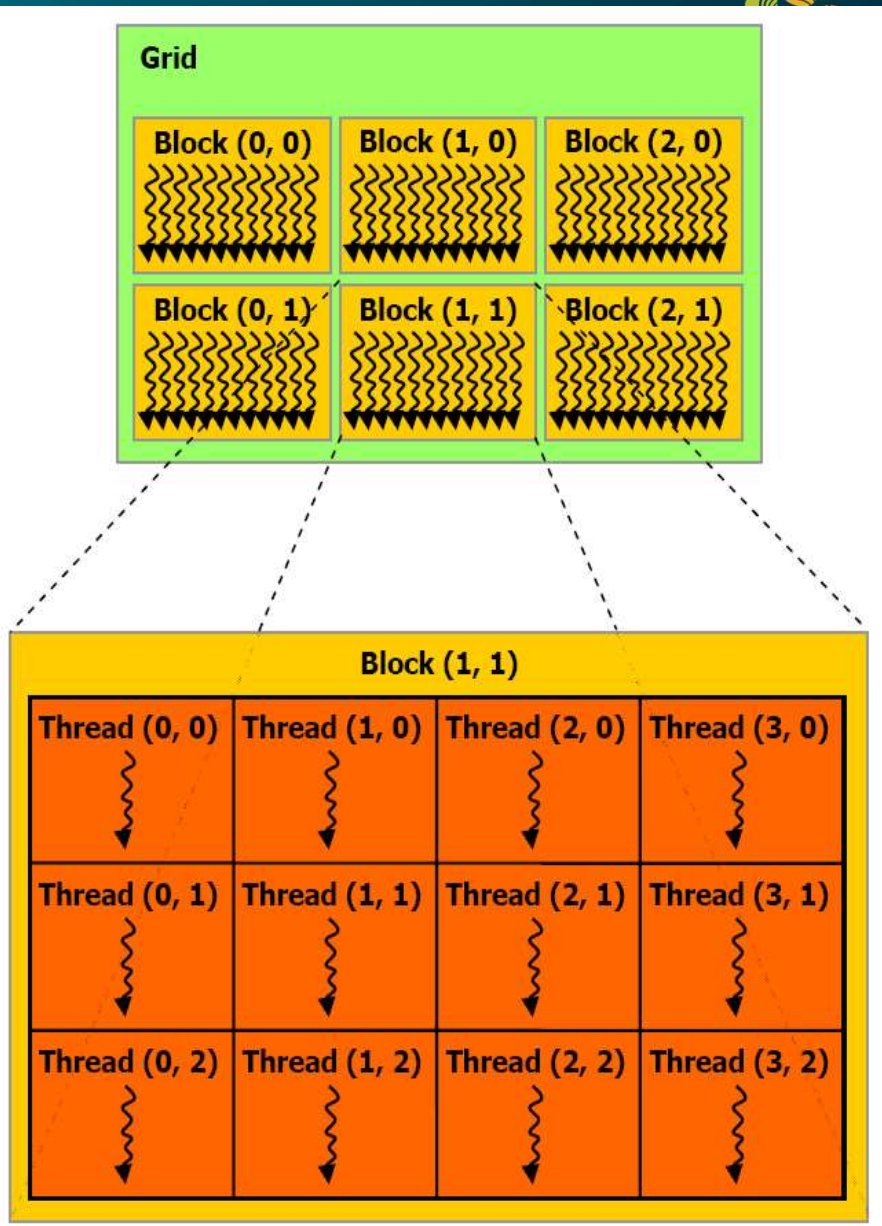
- "Compute Unified Device Architecture"
- Extensions to C(++) programming language
- **`__host__`**, **`__global__`**, and **`__device__`** functions
- Heavily multi-threaded
- Synchronize threads with **`__syncthreads`**(), ...
- Atomic functions
  (before compute capability 2.0 only integer, from 2.0 on also float)

- Compile **`.cu`** files with NVCC
- Uses general C compiler (Visual C, gcc, ...)
- Link with CUDA run-time (**`cudart.lib`**) and cuda core (**`cuda.lib`**)

CUDA Multi-Threading

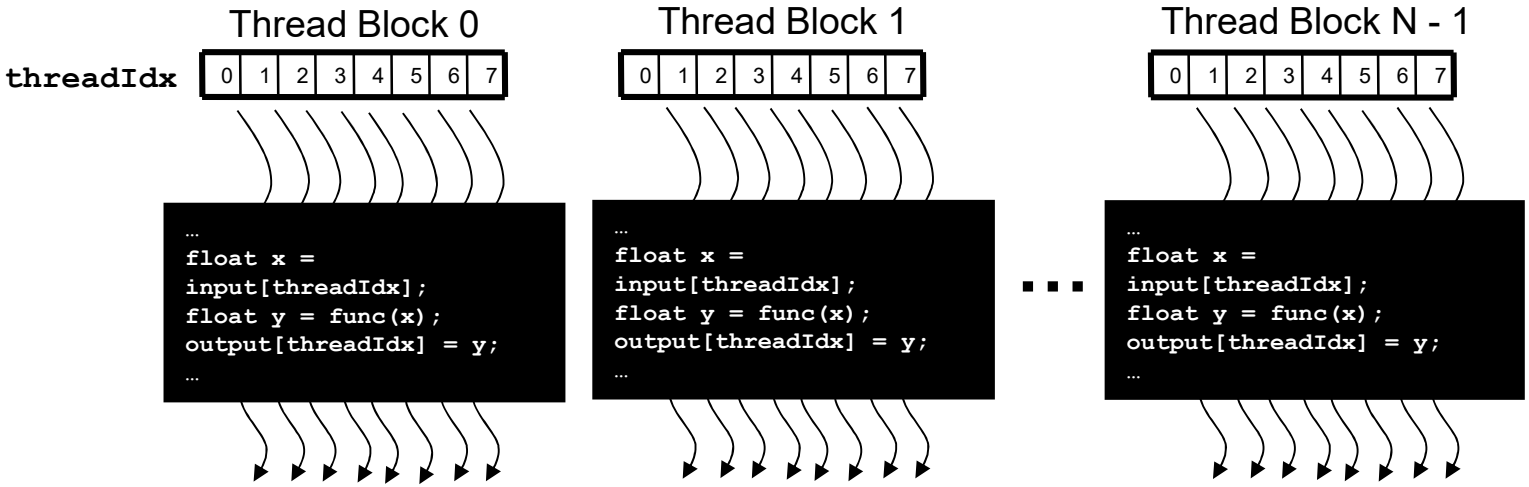
- CUDA model groups threads into blocks; blocks into grid

- Execution on actual hardware:
  - Block assigned to SM (up to 8, 16, or 32 blocks per SM; depending on compute capability)
  - 32 threads grouped into warp



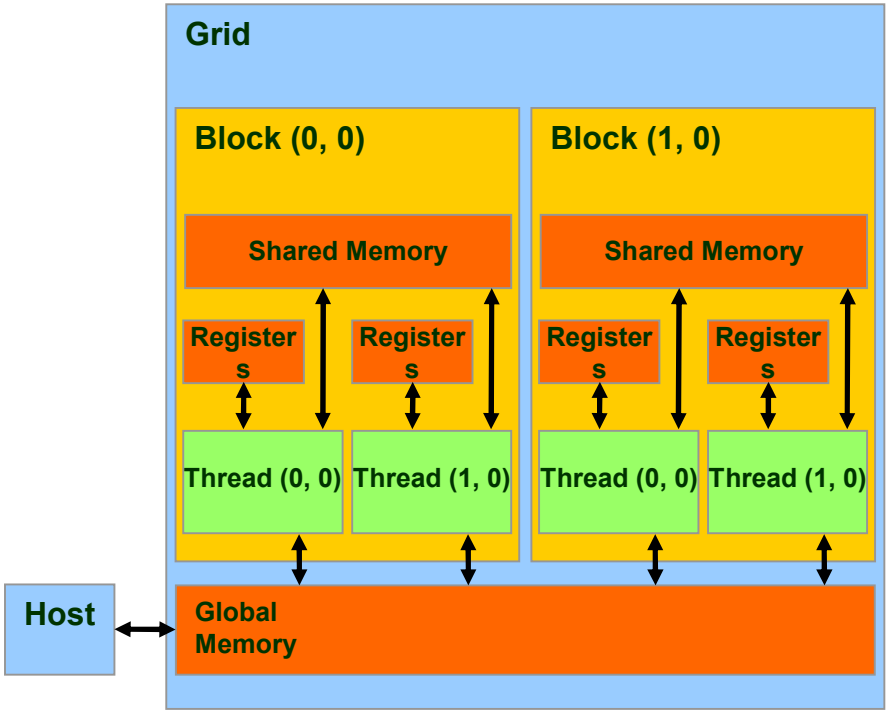
Threads in Block, Blocks in Grid

- Identify work of thread via
  - `threadIdx`
  - `blockIdx`



CUDA Memory Model and Usage

- `cudaMalloc(), cudaFree()`
- `cudaMallocArray(), cudaMalloc2DArray(), cudaMalloc3DArray()`
- `cudaMemcpy()`
- `cudaMemcpyArray()`
- Host ↔ host
  Host ↔ device
  Device ↔ device
- Asynchronous transfers possible (DMA)



CUDA Software Development

CUDA Optimized Libraries:
math.h, FFT, BLAS, …

Integrated CPU + GPU
C Source Code

NVIDIA C Compiler

NVIDIA Assembly
for Computing (PTX)

CPU Host Code

CUDA
Driver

Profiler

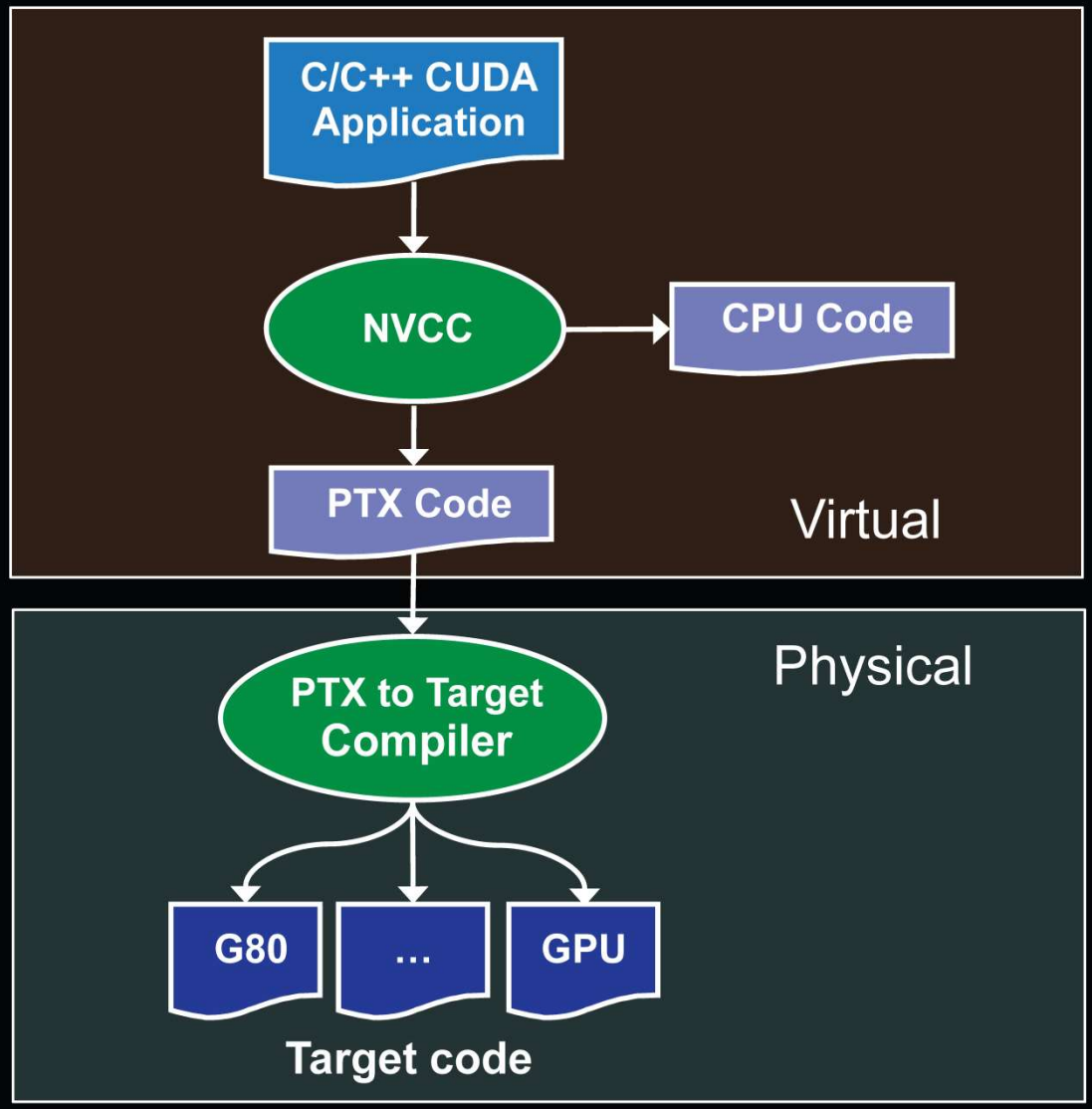
Standard C Compiler

GPU

CPU

NVIDIA

Compiling CUDA Code



CUDA Kernels and Threads

- Parallel portions of an application are executed on the device as kernels
  - One kernel is executed at a time
  - Many threads execute each kernel
- Differences between CUDA and CPU threads
  - CUDA threads are extremely lightweight
    - Very little creation overhead
    - Instant switching
  - CUDA uses 1000s of threads to achieve efficiency
    - Multi-core CPUs can use only a few

Definitions

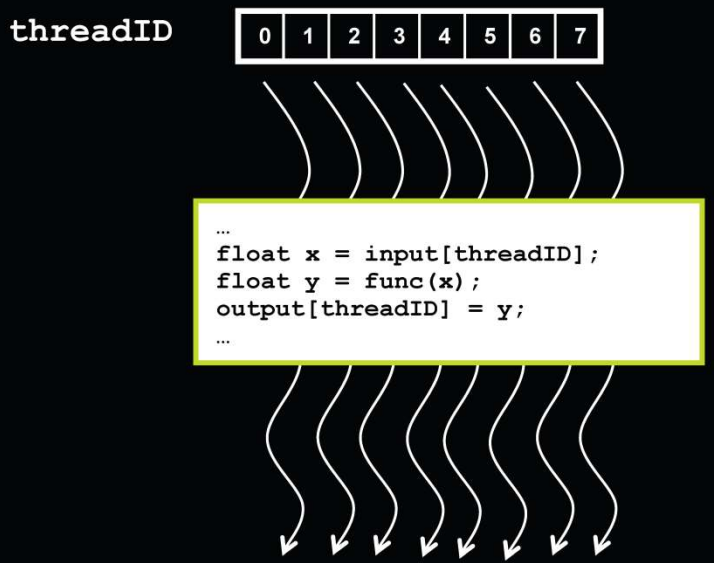
Device = GPU

Host = CPU

Kernel = function that runs on the device

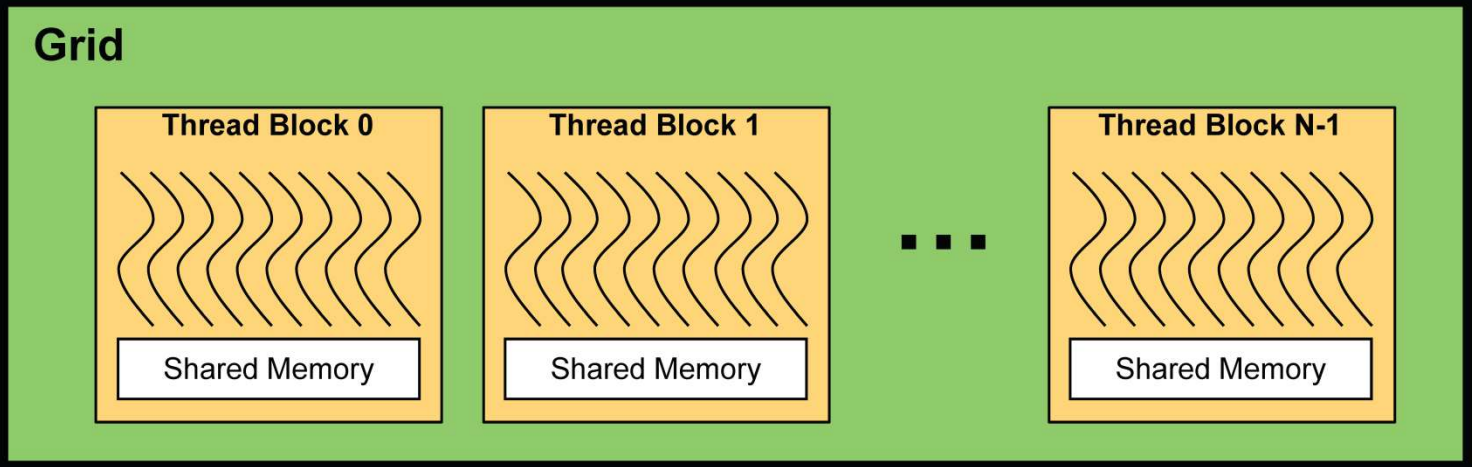
Arrays of Parallel Threads

- A CUDA kernel is executed by an array of threads
  - All threads run the same code
  - Each thread has an ID that it uses to compute memory addresses and make control decisions



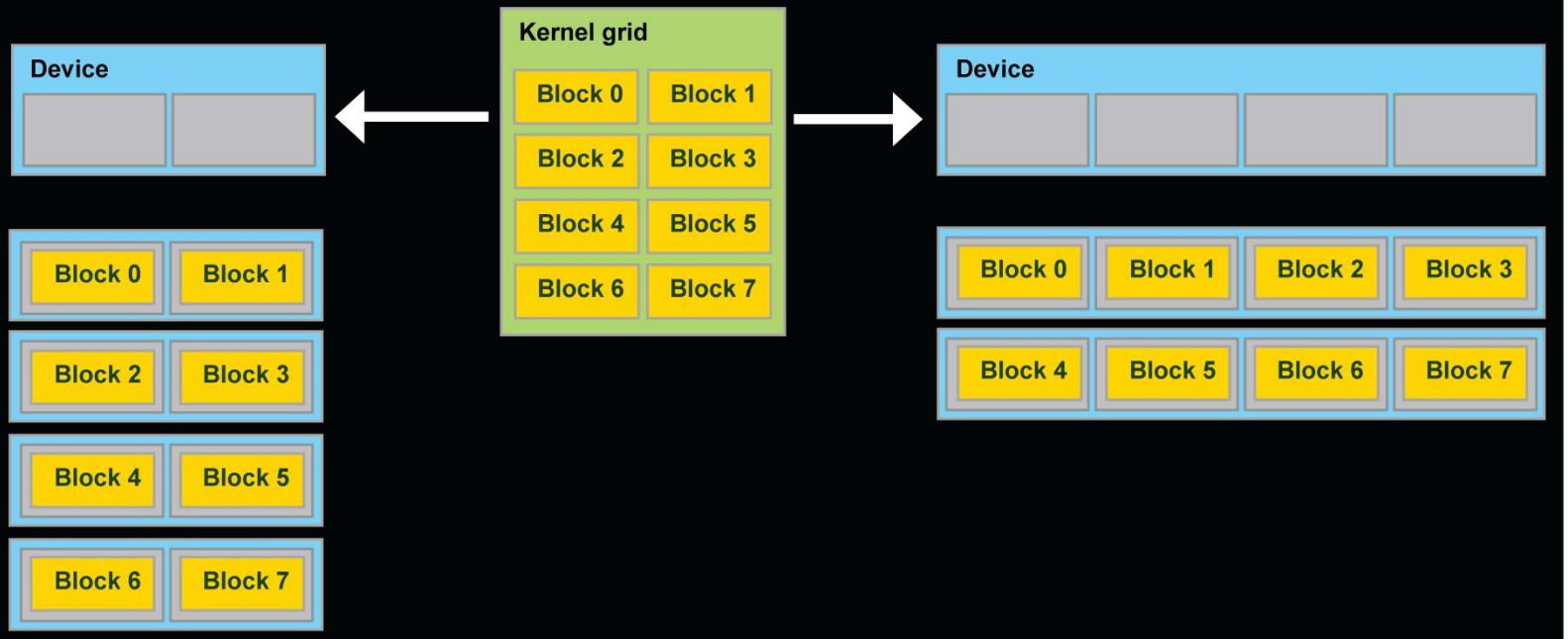
Thread Batching

- Kernel launches a grid of thread blocks
  - Threads within a block cooperate via shared memory
  - Threads within a block can synchronize
  - Threads in different blocks cannot cooperate
- Allows programs to *transparently scale* to different GPUs

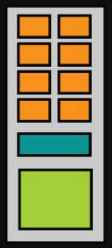
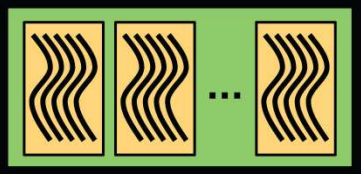
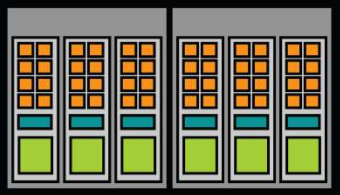


Transparent Scalability

- Hardware is free to schedule thread blocks on any processor
  - A kernel scales across parallel multiprocessors



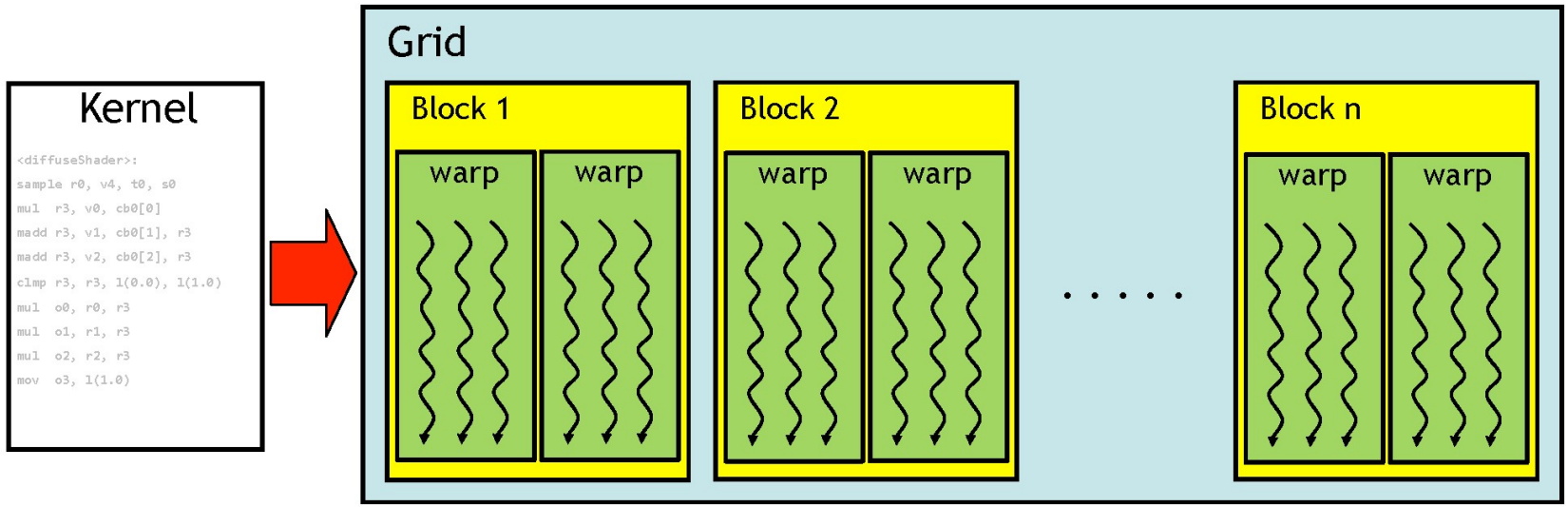
Execution Model

Software	Hardware	
 Thread	 Thread Processor	Threads are executed by thread processors
 Thread Block	 Multiprocessor	Thread blocks are executed on multiprocessors Thread blocks do not migrate Several concurrent thread blocks can reside on one multiprocessor - limited by multiprocessor resources (shared memory and register file)
 Grid	 Device	A kernel is launched as a grid of thread blocks Only one kernel can execute on a device at one time

NVIDIA

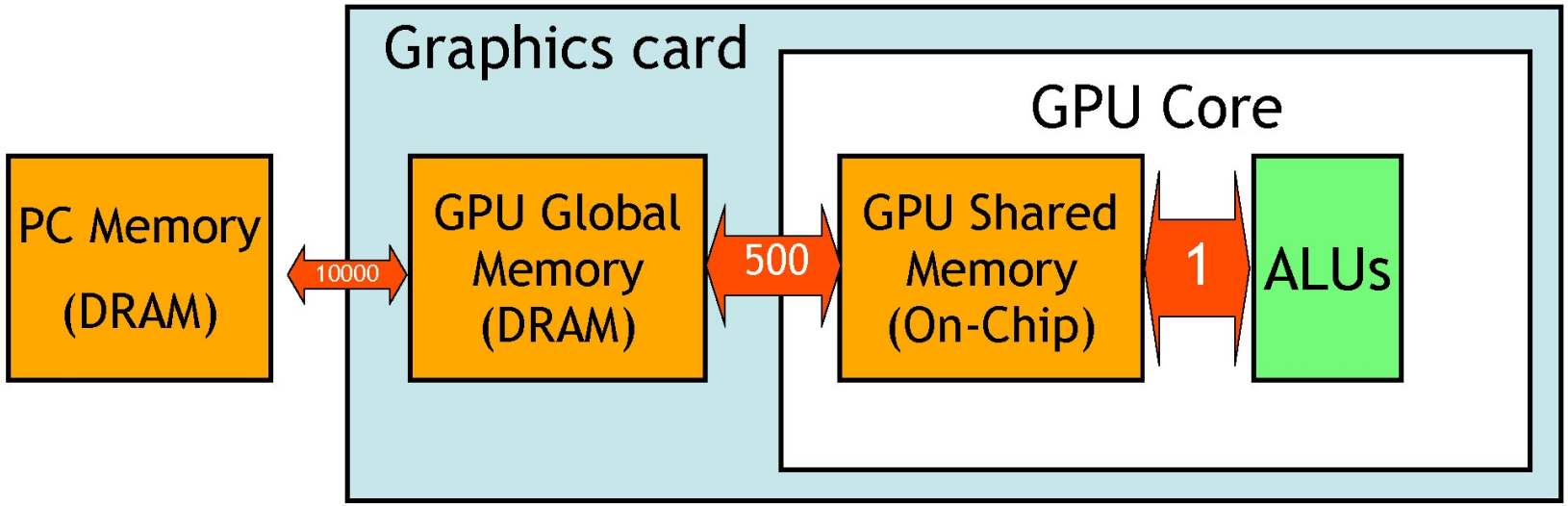
CUDA Programming Model

- Kernel
  - GPU program that runs on a thread grid
- Thread hierarchy
  - Grid : a set of blocks
  - Block : a set of warps
  - Warp : a SIMD group of 32 threads
  - Grid size * block size = total # of threads



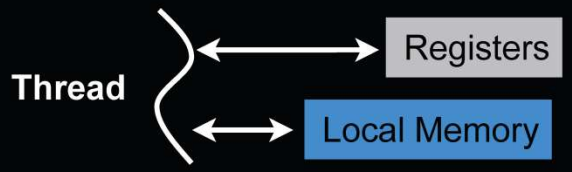
CUDA Memory Structure

- Memory hierarchy
  - PC memory : off-card
  - GPU global : off-chip / on-card
  - GPU shared/register/cache : on-chip
- The host can read/write global memory
- Each thread communicates using shared memory



Kernel Memory Access

- **Per-thread**

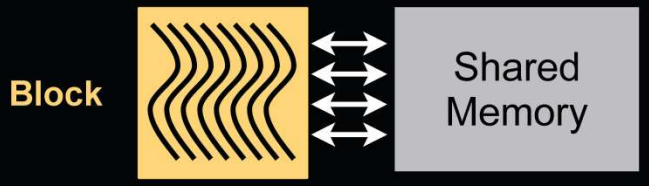


On-chip

Off-chip, uncached

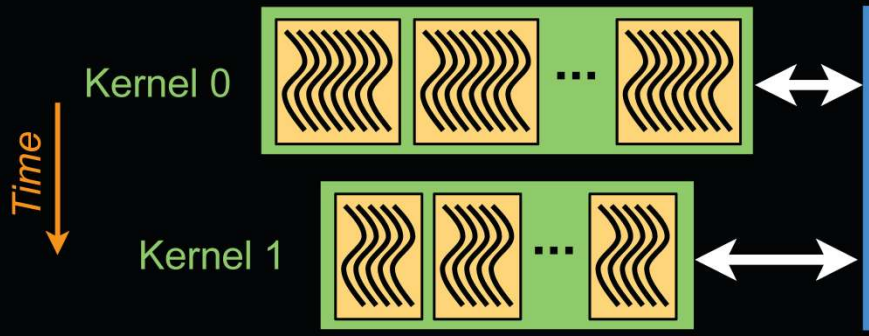
cached on Fermi or newer

- **Per-block**



- On-chip, small
- Fast

- **Per-device**



- Off-chip, large
- Uncached
- Persistent across kernel launches
- Kernel I/O

cached on Fermi or newer

Memory Architecture

Memory	Location	Cached	Access	Scope	Lifetime
Register	On-chip	N/A	R/W	One thread	Thread
Local	Off-chip	No*	R/W	One thread	Thread
Shared	On-chip	N/A	R/W	All threads in a block	Block
Global	Off-chip	No*	R/W	All threads + host	Application
Constant	Off-chip	Yes	R	All threads + host	Application
Texture	Off-chip	Yes	R	All threads + host	Application

* cached on Fermi or newer

(Memory) State Spaces

PTX ISA 7.1 (Chapter 5)

Name	Addressable	Initializable	Access	Sharing
`.reg`	No	No	R/W	per-thread
`.sreg`	No	No	RO	per-CTA
`.const`	Yes	Yes[1]	RO	per-grid
`.global`	Yes	Yes[1]	R/W	Context
`.local`	Yes	No	R/W	per-thread
`.param` (as input to kernel)	Yes[2]	No	RO	per-grid
`.param` (used in functions)	Restricted[3]	No	R/W	per-thread
`.shared`	Yes	No	R/W	per-CTA
`.tex`	No[4]	Yes, via driver	RO	Context

Notes:

[1] Variables in `.const` and `.global` state spaces are initialized to zero by default.

[2] Accessible only via the `ld.param` instruction. Address may be taken via `mov` instruction.

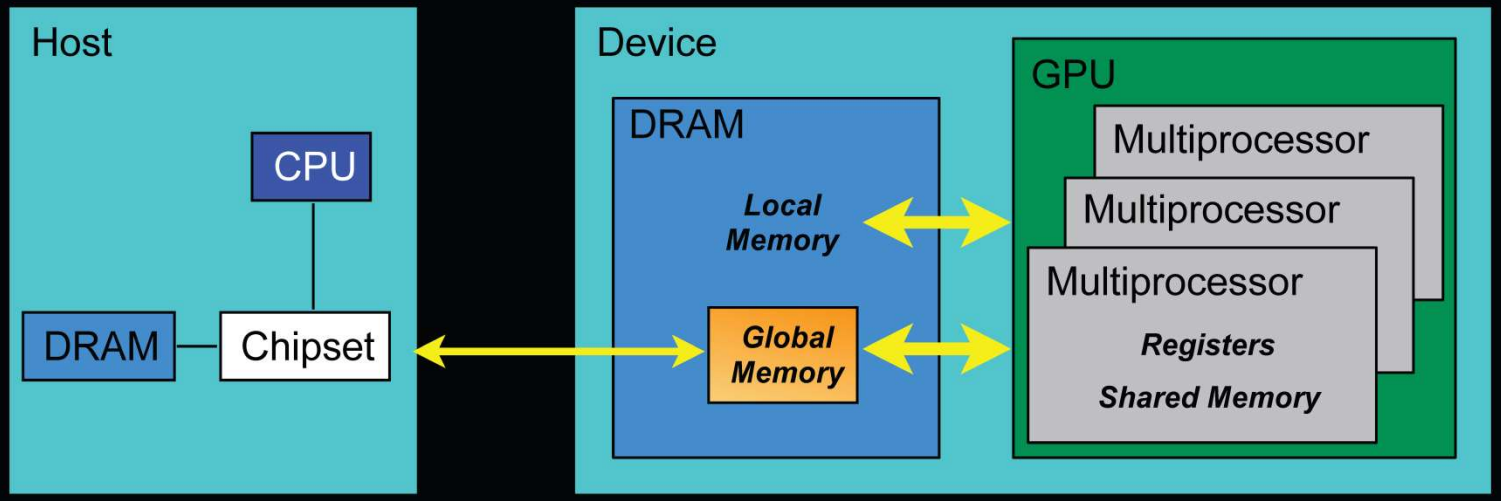
[3] Accessible via `ld.param` and `st.param` instructions. Device function input and return parameters may have their address taken via `mov`; the parameter is then located on the stack frame and its address is in the `.local` state space.

[4] Accessible only via the `tex` instruction.

Managing Memory

Unified memory space can be enabled on Fermi / CUDA 4.x and newer

- CPU and GPU have separate memory spaces
- Host (CPU) code manages device (GPU) memory:
  - Allocate / free
  - Copy data to and from device
  - Applies to *global* device memory (DRAM)



Thank you.