

GPU Fundamentals

Jeff Larkin <jlarkin@nvidia.com>, November 14, 2016



Who Am I?

2002 - B.S. Computer Science - Furman University

2005 - M.S. Computer Science - UT Knoxville

2002 - Graduate Teaching Assistant

2005 - Graduate Research Assistant (ICL)

2005 - 2013 - Cray, Inc

Worked on porting & optimizing HPC apps @ ORNL, User Training

2013 - Present - NVIDIA Corp.

Porting & optimizing HPC apps @ ORNL , User Training,

Representative to OpenACC & OpenMP

AGENDA

GPU Architecture

Speed v. Throughput

Latency Hiding

Memory Coalescing

SIMD v. SIMT

GPU Architecture

Two Main Components

Global memory

Analogous to RAM in a CPU server

Accessible by both GPU and CPU

Currently up to **16 GB** in Tesla products

Streaming Multiprocessors (SM)

Perform the actual computation

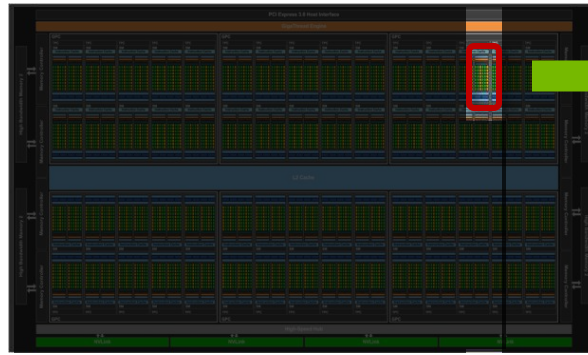
Each SM has its own: Control units, registers, execution pipelines, caches



GPU Architecture

Streaming Multiprocessor (SM)

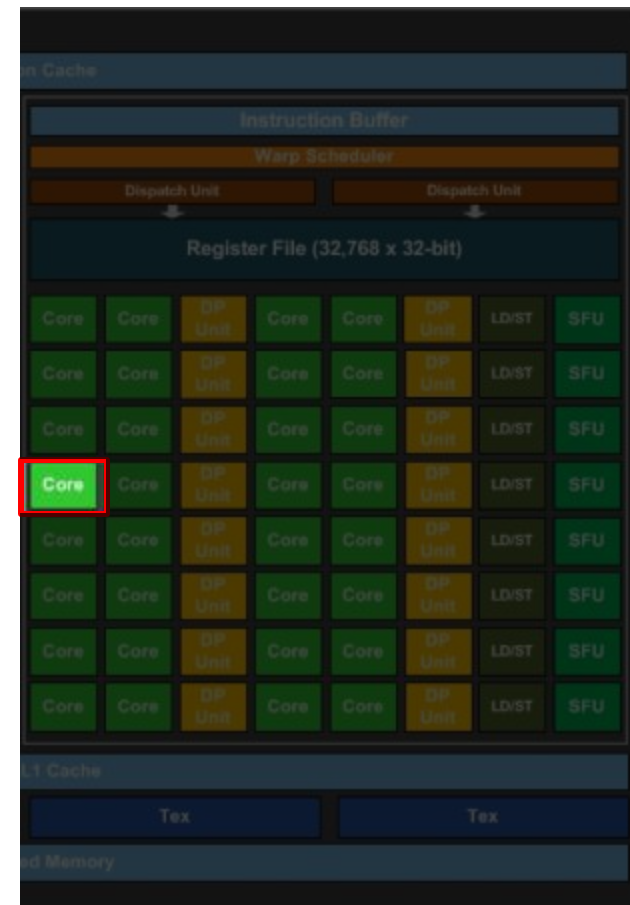
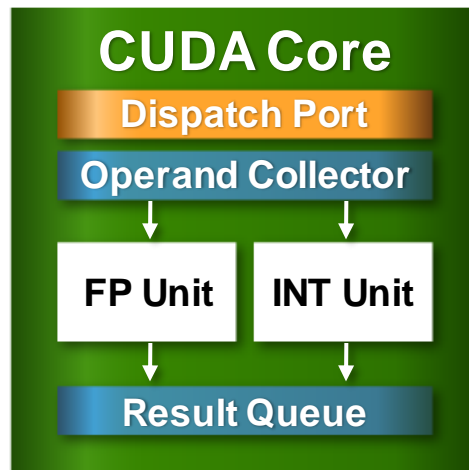
- ▶ Many CUDA Cores per SM
 - ▶ Architecture dependent
- ▶ Special-function units
 - ▶ cos/sin/tan, etc.
- ▶ Shared memory + L1 cache
- ▶ Thousands of 32-bit registers



GPU Architecture

CUDA Core

- ▶ Floating point & Integer unit
 - ▶ IEEE 754-2008 floating-point standard
 - ▶ Fused multiply-add (FMA) instruction for both single and double precision
- ▶ Logic unit
- ▶ Move, compare unit
- ▶ Branch unit




Execution Model

Software

Hardware

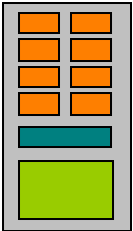
Thread


Scalar
Processor

Threads are executed by scalar processors



Thread
Block

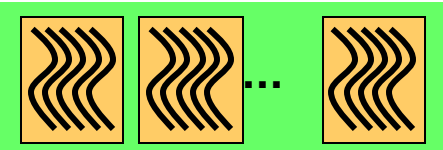


Multiprocessor

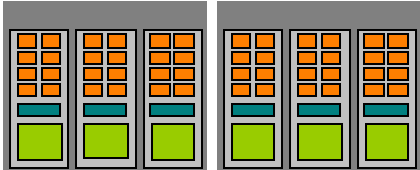
Thread blocks are executed on multiprocessors

Thread blocks do not migrate

Several concurrent thread blocks can reside on one multiprocessor - limited by multiprocessor resources (shared memory and register file)



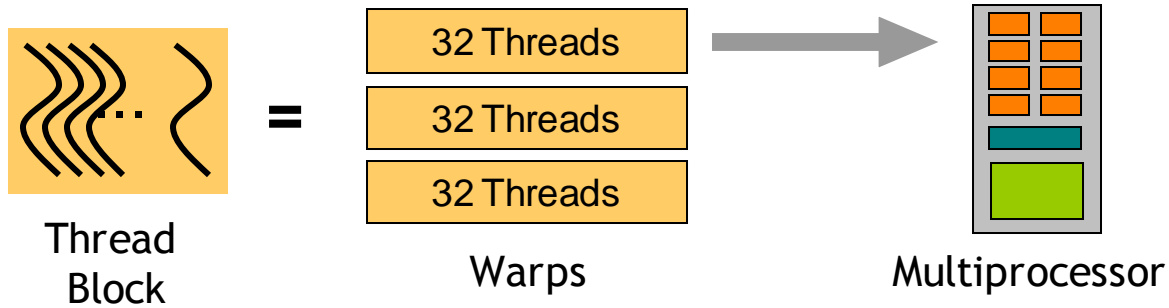
Grid



Device

A kernel is launched as a grid of thread blocks

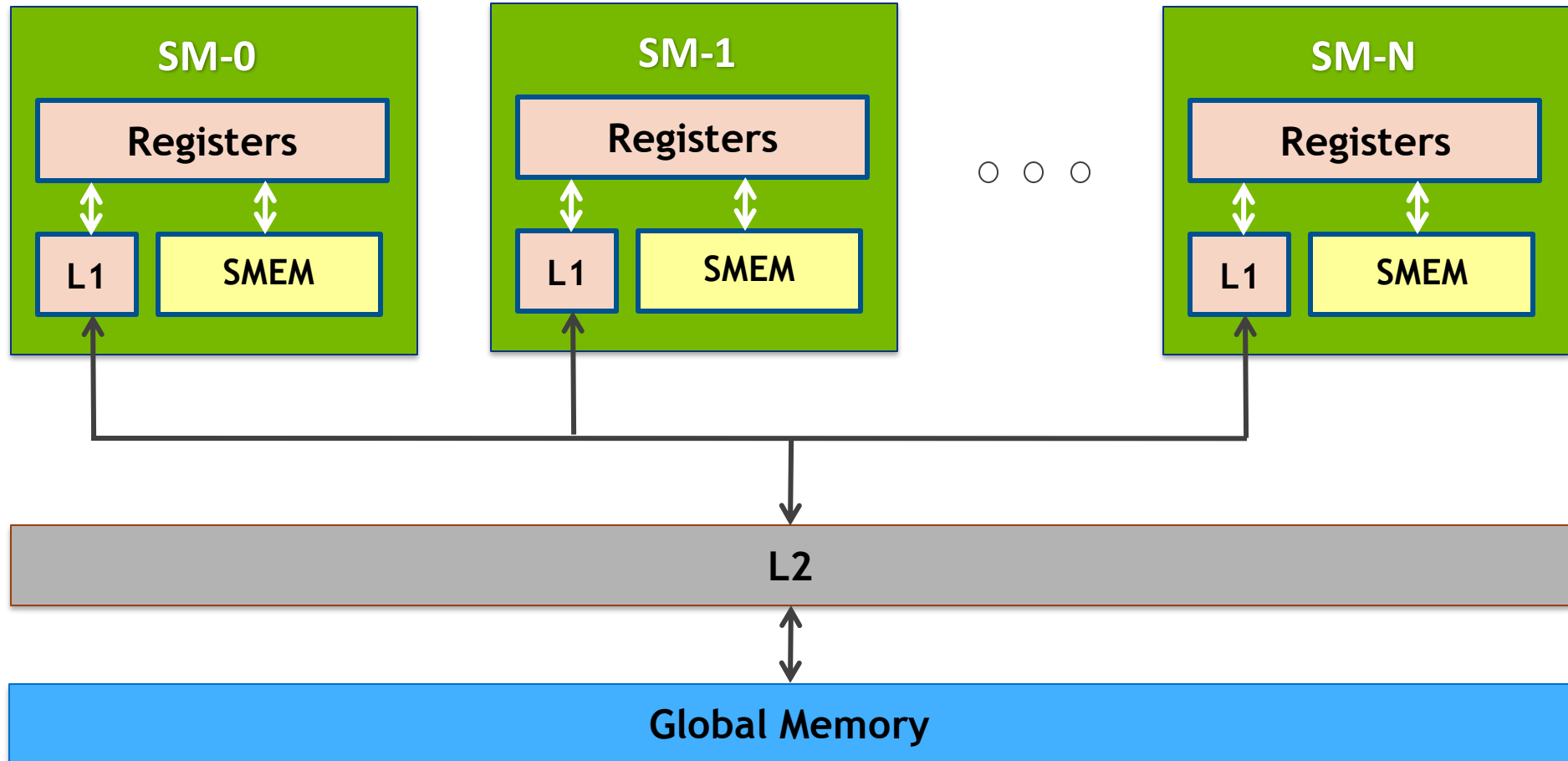
Warps



A thread block consists of 32-thread warps

A warp is executed physically in parallel (SIMT) on a multiprocessor

GPU Memory Hierarchy Review



GPU Architecture

Memory System on each SM

Extremely fast, but small, i.e., 10s of Kb

Programmer chooses whether to use cache as L1 or Shared Mem

L1

Hardware-managed—used for things like register spilling

Should NOT attempt to utilize like CPU caches

Shared Memory—programmer **MUST** synchronize data accesses!!!

User-managed scratch pad

Repeated access to same data or multiple threads with same data

GPU Architecture

Memory system on each GPU board

Unified L2 cache (100s of Kb)

Fast, coherent data sharing across all cores in the GPU

Unified/Managed Memory

Since CUDA6 it's possible to allocate 1 pointer (virtual address) whose physical location will be managed by the runtime.

Pre-Pascal GPUS - managed by software, limited to GPU memory size

Pascal & Beyond - Hardware can page fault to manage location, can oversubscribe GPU memory.

Speed v. Throughput

Speed

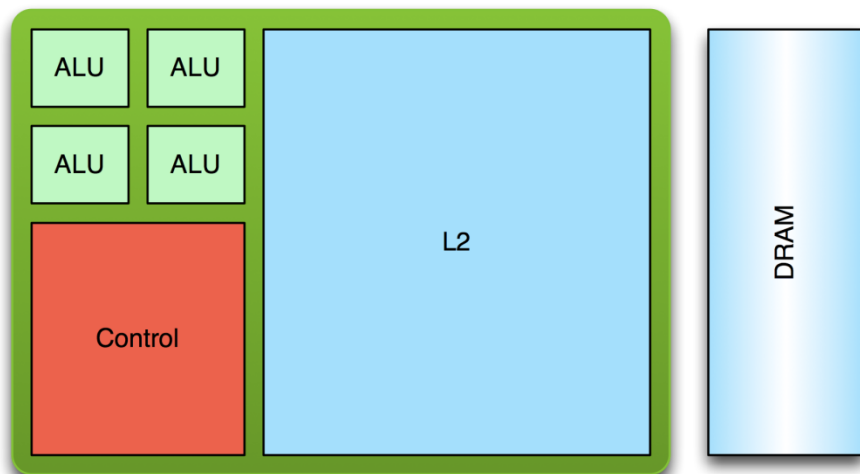


Throughput



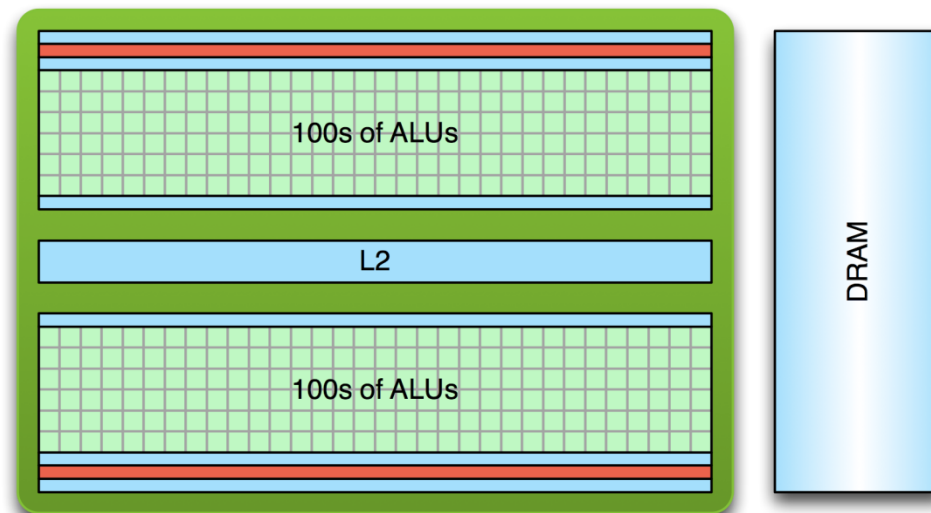
Which is better depends on your needs...

Low Latency or High Throughput?



CPU

- Optimized for low-latency access to cached data sets
- Control logic for out-of-order and speculative execution
- **10's of threads**



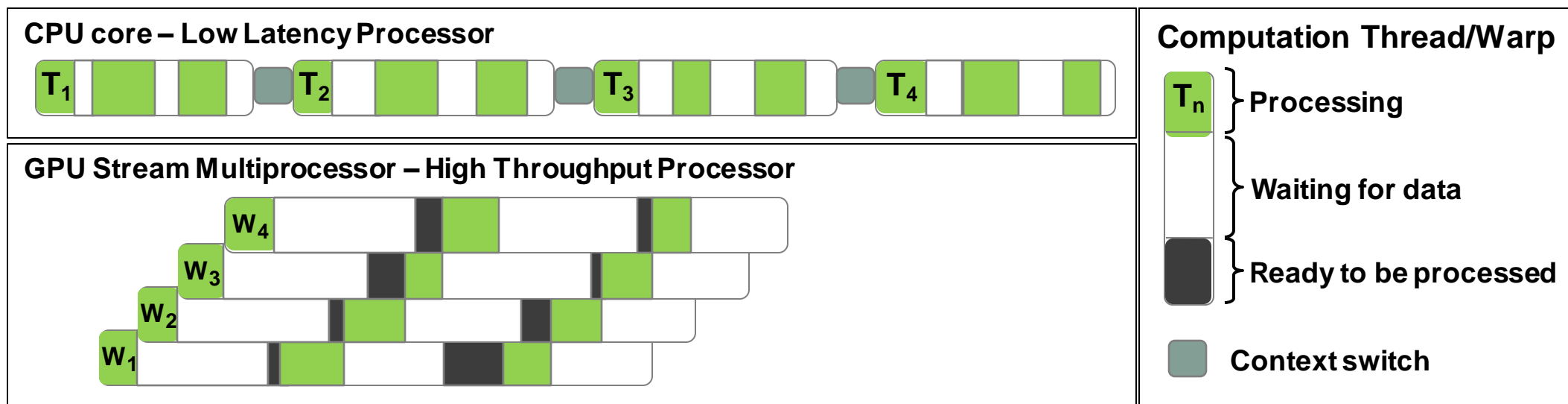
GPU

- Optimized for data-parallel, throughput computation
- Tolerant of memory latency
- More transistors dedicated to computation
- **10,000's of threads**

Low Latency or High Throughput?

CPU architecture must **minimize latency** within each thread

GPU architecture **hides latency** with computation from other thread warps



Memory Coalescing

Global memory access happens in transactions of 32 or 128 bytes

The hardware will try to reduce to as few transactions as possible

Coalesced access:

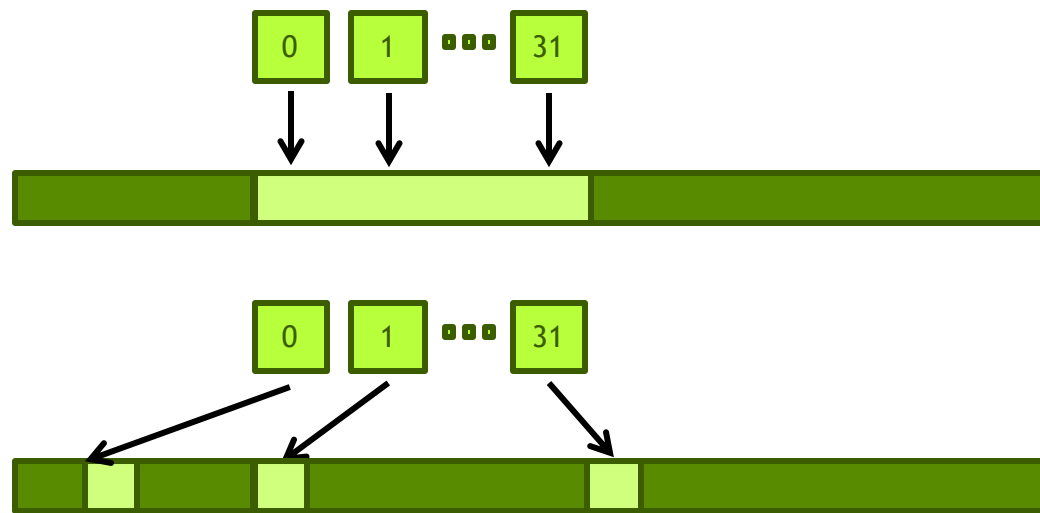
A group of 32 contiguous threads (“warp”) accessing adjacent words

Few transactions and high utilization

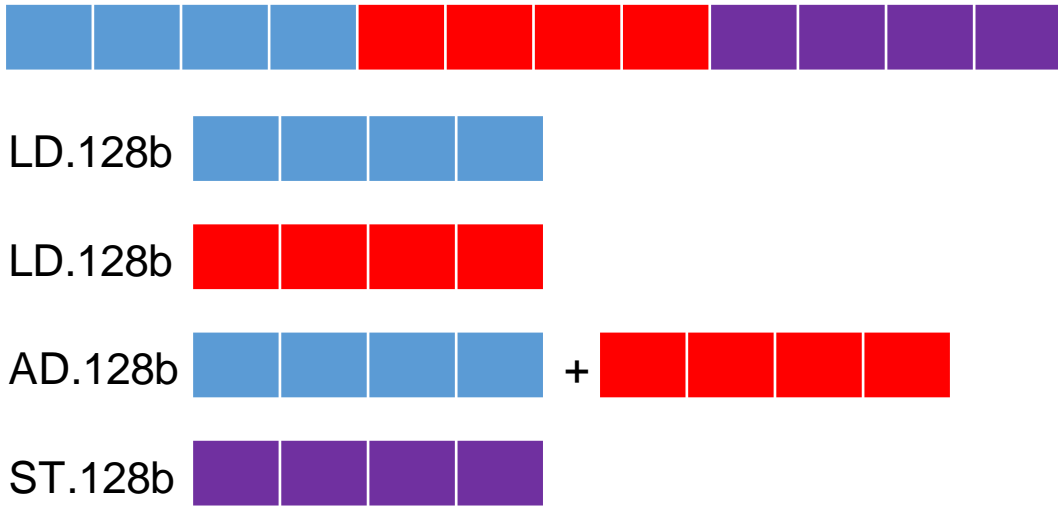
Uncoalesced access:

A warp of 32 threads accessing scattered words

Many transactions and low utilization

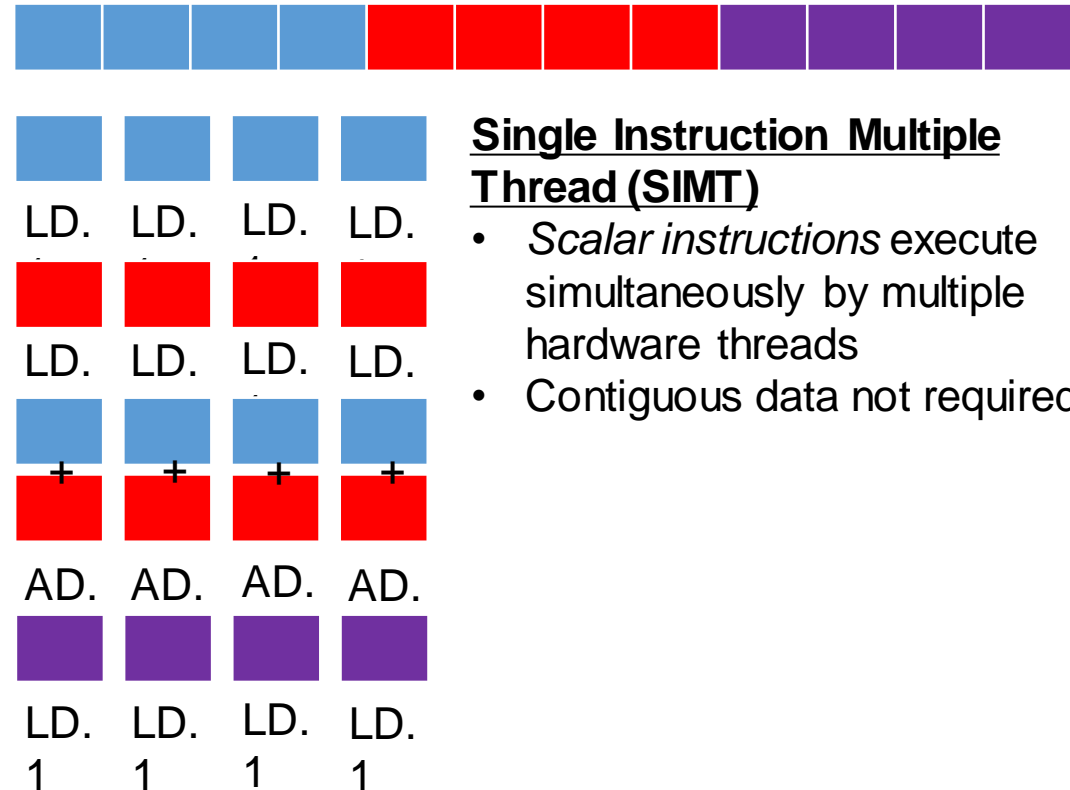


SIMD and SIMT



Single Instruction Multiple Data (SIMD)

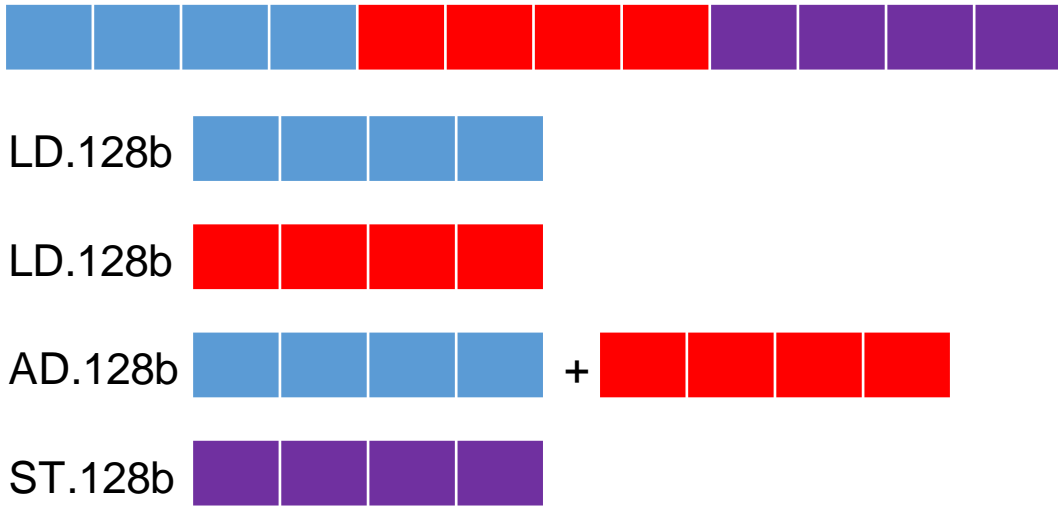
- *Vector instructions* perform the same operation on multiple data elements.
- Data must be loaded and stored in contiguous buffers



Single Instruction Multiple Thread (SIMT)

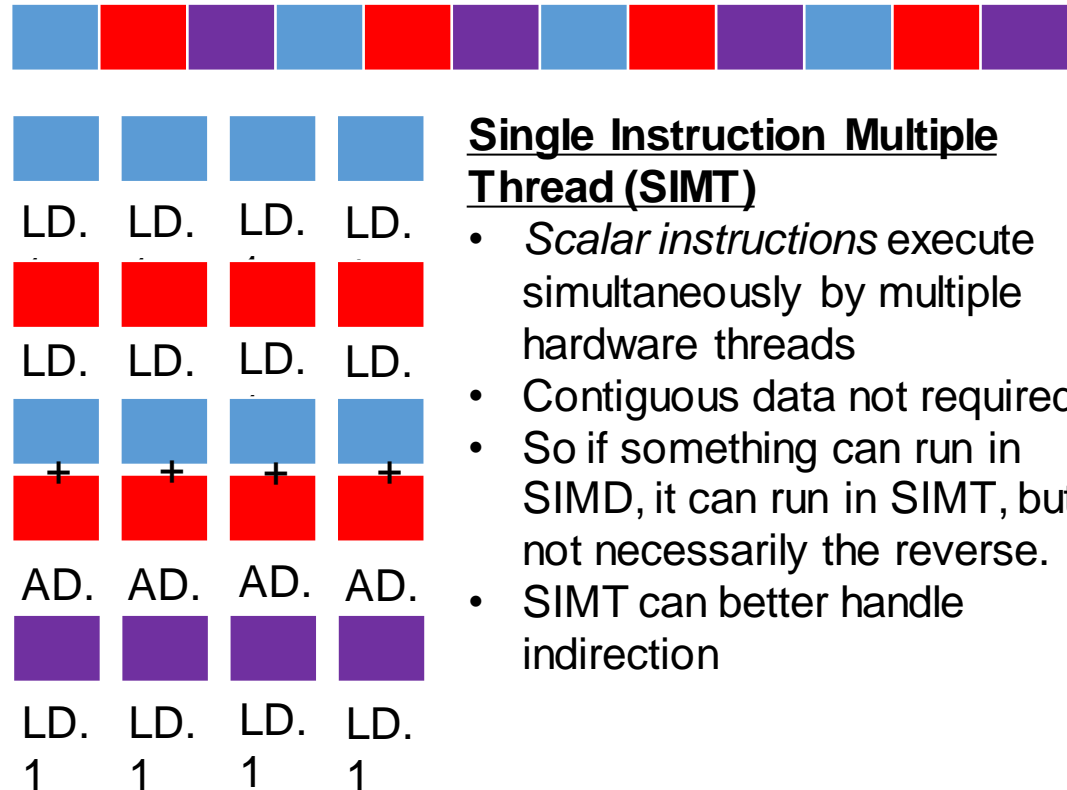
- *Scalar instructions* execute simultaneously by multiple hardware threads
- Contiguous data not required.

SIMD and SIMT



Single Instruction Multiple Data (SIMD)

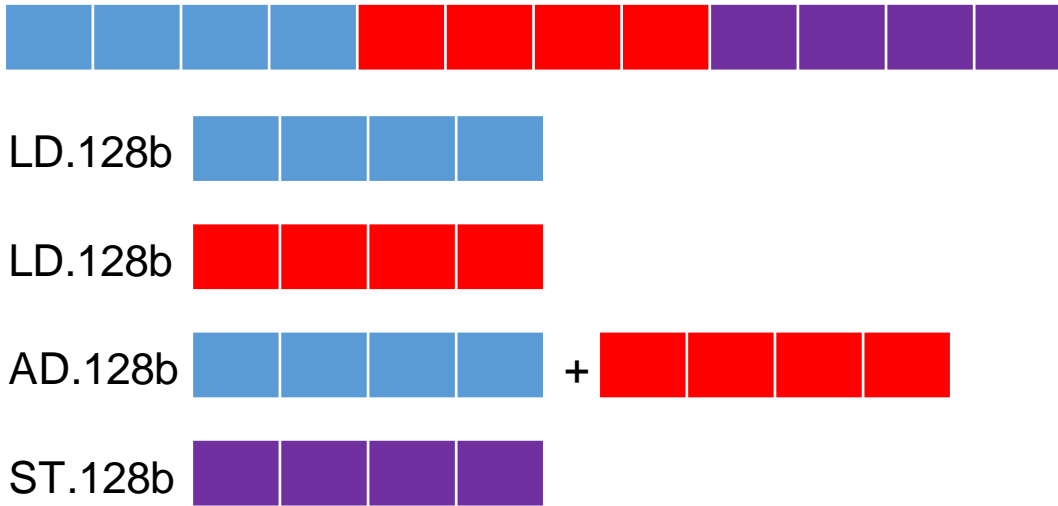
- *Vector instructions* perform the same operation on multiple data elements.
- Data must be loaded and stored in contiguous buffers



Single Instruction Multiple Thread (SIMT)

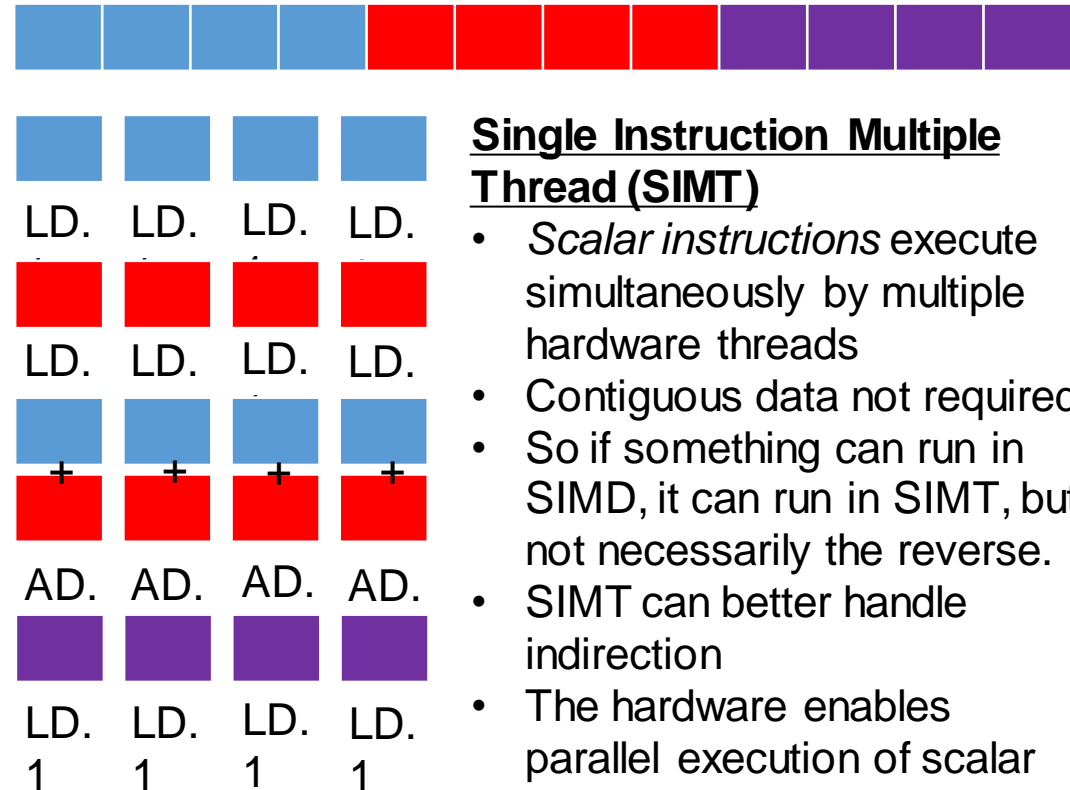
- *Scalar instructions* execute simultaneously by multiple hardware threads
- Contiguous data not required.
- So if something can run in SIMD, it can run in SIMT, but not necessarily the reverse.
- SIMT can better handle indirection

SIMD and SIMT



Single Instruction Multiple Data (SIMD)

- *Vector instructions* perform the same operation on multiple data elements.
- Data must be loaded and stored in contiguous buffers
- Either the programmer or the compiler must generate vector instructions

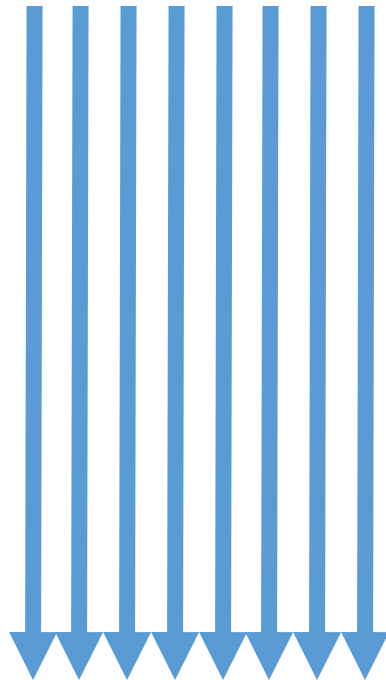


Single Instruction Multiple Thread (SIMT)

- *Scalar instructions* execute simultaneously by multiple hardware threads
- Contiguous data not required.
- So if something can run in SIMD, it can run in SIMT, but not necessarily the reverse.
- SIMT can better handle indirection
- The hardware enables parallel execution of scalar instructions

SIMD and SIMT Branching

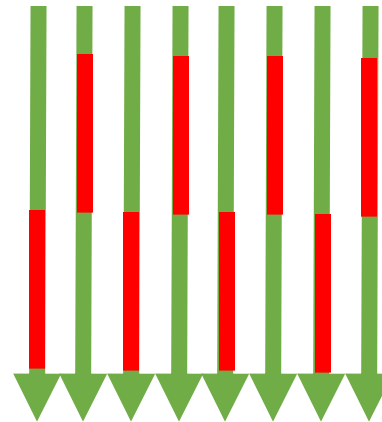
SIMD



1. Execute converged instructions
2. Generate vector mask for true
3. Execute masked vector instruction
4. Generate vector mask for false
5. Execute masked vector instruction
6. Continue to execute converged instructions

Divergence (hopefully) handled by compiler through masks and/or gather/scatter operations.

SIMT



1. Execute converged instructions
2. Executed true branch
3. Execute false branch
4. Continue to execute converged instructions

Divergence handle by hardware through predicated instructions.

Next 2 Lectures

Wednesday - OpenACC Basics

Friday - More OpenACC?