

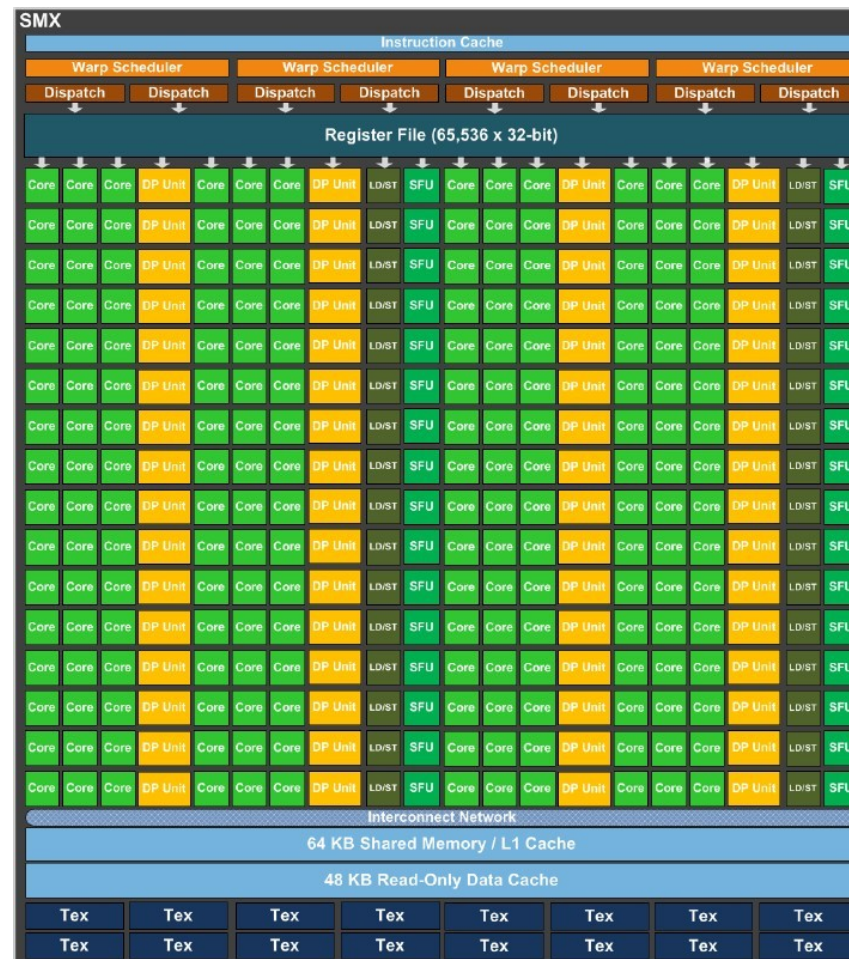
CUDA OPTIMIZATION, PART 1

NVIDIA Corporation

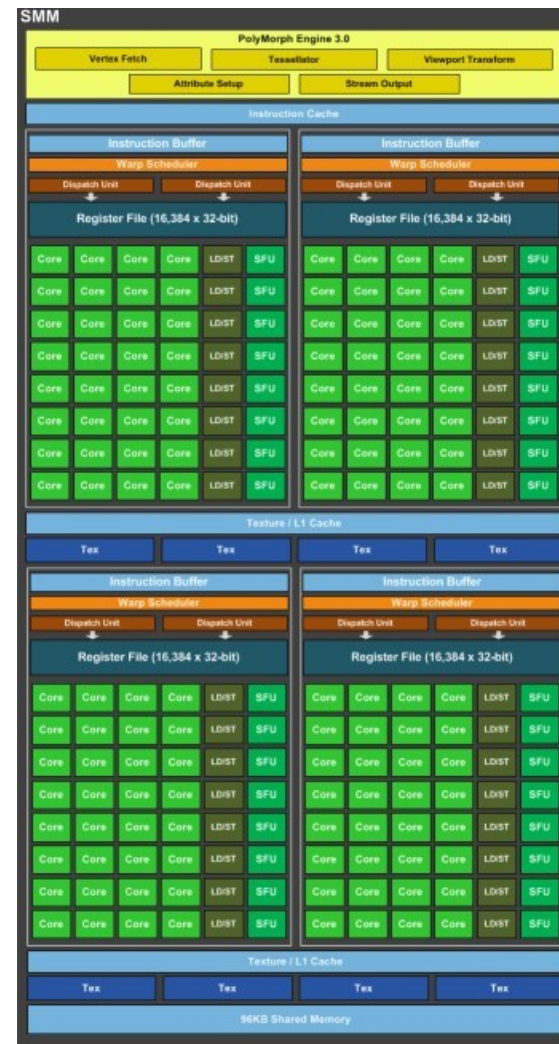
- Architecture:
 - Kepler/Maxwell/Pascal/Volta
- Kernel optimizations
 - Launch configuration (use lots of threads)
- Part 2 (next session):
 - Global memory throughput (use memory efficiently)
 - Shared memory access

Most concepts in this presentation apply to *any* language or API on NVIDIA GPUs

- “SMX” (enhanced SM) **SIMD processor**
- 192 SP(single precision) units (“cores”)
- 64 DP (double precision) units
- LD/ST units,
- 64K registers
- 4 warp scheduler
- Each warp scheduler is dual-issue capable
- K20: 13 SMX's, 5GB
- K20X: 14 SMX's, 6GB
- K40: 15 SMX's, 12GB



- “SMM” (enhanced SM)
- 128 SP units (“cores”)
- 4 DP units
- LD/ST units
- cc 6.1: INT8
- 4 warp schedulers
- Each warp scheduler is dual-issue capable
- M40: 24 SMM’s, 12/24GB
- P40: 30 SM’s, 24GB
- P4: 20 SM’s, 8GB



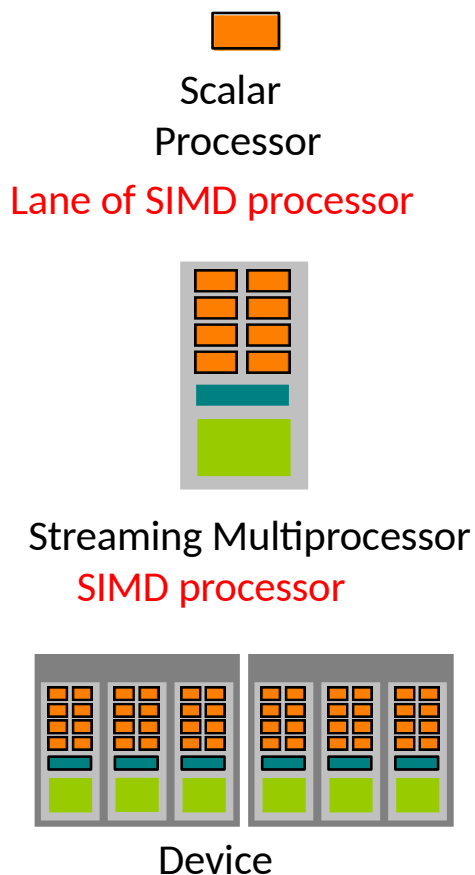
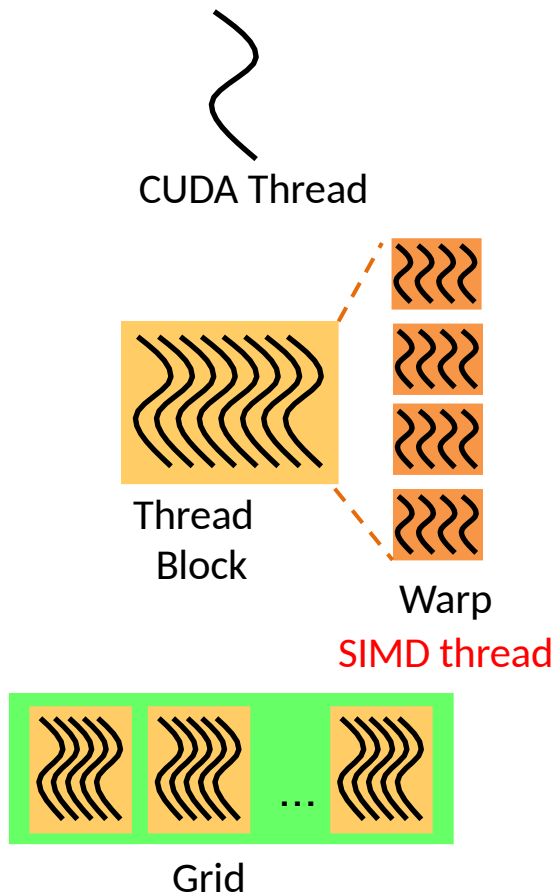
Each of SMs

- 64 SP units (“cores”)
- 32 DP units
- LD/ST units
- FP16 @ 2x SP rate
- cc7.0: TensorCore
- P100/V100 2/4 warp schedulers
- Volta adds separate int32 units
- P100: 56 SM's, 16GB
- V100: 80 SM's, 16/32GB



Software

Hardware

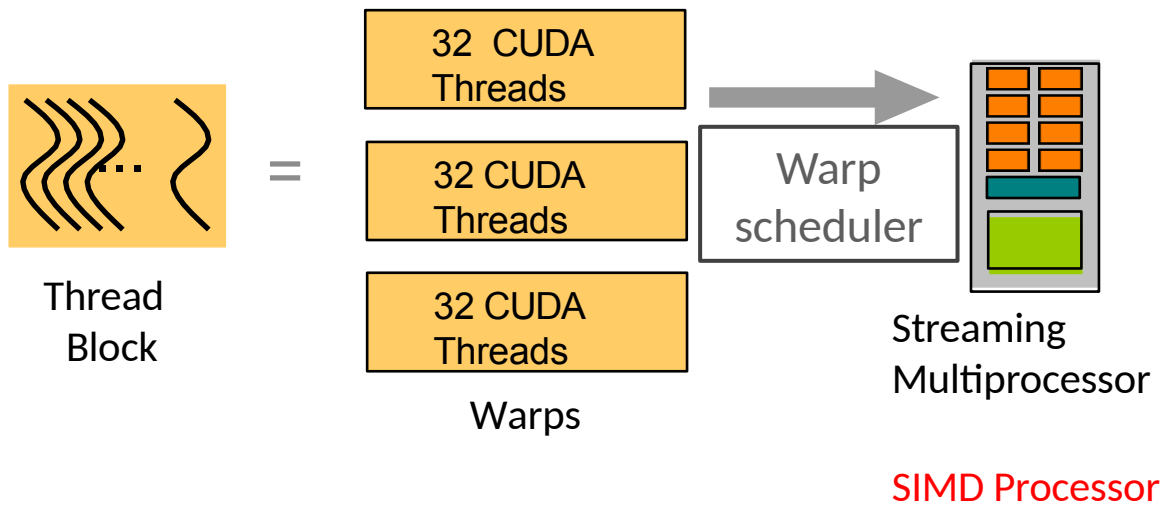


CUDA threads are executed by scalar processors (**Lane of SIMD processor**)
车道

Thread blocks are split to warps and executed on multiprocessors (**SIMD processor**) in SIMT.

Several concurrent thread blocks can reside on one SIMD processor (when the number of thread blocks greater than multiprocessors) - limited by SIMD processor resources (shared memory and register file)

A kernel is launched as a grid of thread blocks



A thread block consists of thread warps

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

LAUNCH CONFIGURATION

- Key to understanding:
 - Instructions are issued in order
 - A SIMD thread stalls when one of the operands isn't ready:
 - Memory read by itself doesn't stall execution
 - **Latency is hidden** by switching threads
 - GPU MEM latency: >100 cycles e.g. 100 (varies by architecture/design)
 - Arithmetic latency: <100 cycles e.g. 10 (varies by architecture/design)
- How many threads/threadblocks to launch?
- Conclusion:
 - Need enough threads to hide latency

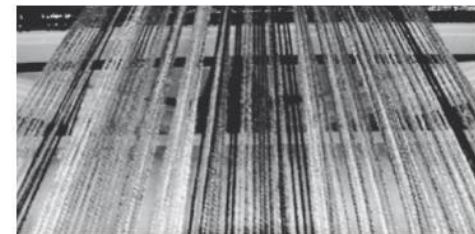
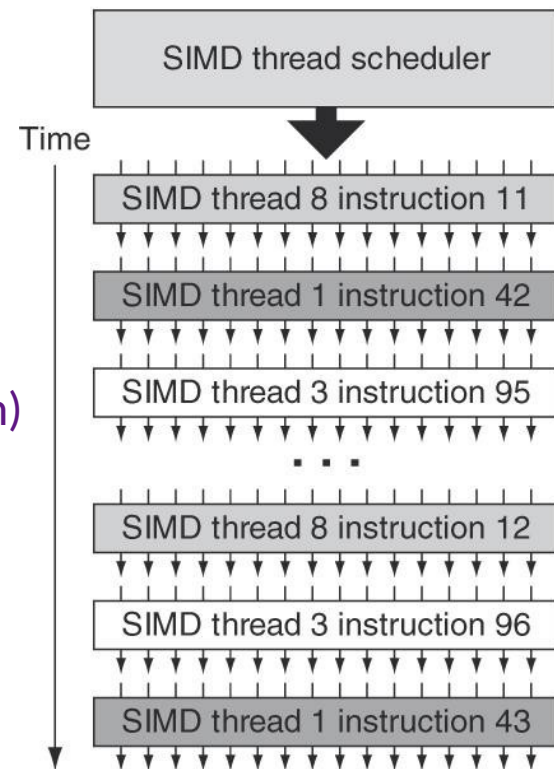


Photo: Judy Schoonmaker



- **Suspect unhidden latency if:**

- Neither memory nor instruction throughput is close to HW theoretical rates
- Poor overlap between mem access and computation
 - Full-kernel time is significantly larger than $\max\{\text{mem-only}, \text{compute-only}\}$
 - Refer to SC10 or GTC10 Analysis-Driven Optimization slides for details

- **Two possible causes:**

- Insufficient concurrent warps per SIMD processor to hide latency
 - Occupancy too low
 - Too few threads in kernel launch to load the GPU
 - elapsed time doesn't change if problem size is increased (and with it the number of blocks/threads)
- Too few concurrent warps per SIMD processor when using `__syncthreads()`
 - `__syncthreads()` can prevent overlap between compute and mem within the same threadblock



Kernel where most math cannot be executed until all data is loaded by the warp



Full-kernel time, one large threadclock (with many warps) per SIMD processor

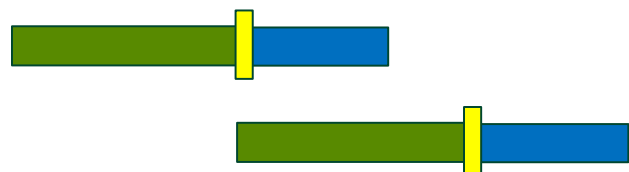
time →



Kernel where most math cannot be executed until all data is loaded by the threadblock



Full-kernel time, one warp per SIMD processor



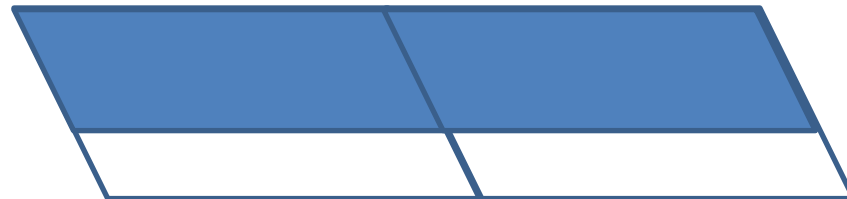
Full-kernel time, two warps per SIMD processor (each half the size of one large one)

time →

- In CUDA C source code:
- `int idx = threadIdx.x+blockDim.x*blockIdx.x;`
- `c[idx] = a[idx] * b[idx];`



- In machine code:
- I0: LD R0, a[idx];
- I1: LD R1, b[idx];
- I2: MPY R2,R0,R1



More warps sequentially access data elements without overlapping.

- I0: LD R0, a[idx];
- I1: LD R1, b[idx];
- I2: MPY R2,R0,R1

clock cycles:

C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17 C18 ...

warps

W0:

W1:

W2:

W3:

W4:

W5:

W6:

W7:

W8:

W9:

...

- I0: LD R0, a[idx];
- I1: LD R1, b[idx];
- I2: MPY R2,R0,R1

clock cycles:

C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17 C18 ...

warps

W0: 

W1:

W2:

W3:

W4:

W5:

W6:

W7:

W8:

W9:

...

- I0: LD R0, a[idx];
- I1: LD R1, b[idx];
- I2: MPY R2,R0,R1

clock cycles:

C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17 C18 ...

warps

W0: 

W1:

W2:

W3:

W4:

W5:

W6:

W7:

W8:

W9:

...

- I0: LD R0, a[idx];
- I1: LD R1, b[idx];
- I2: MPY R2,R0,R1

clock cycles:

C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17 C18 ...

warps

W0:



W1:



W2:

W3:

W4:

W5:

W6:

W7:

W8:

W9:

...

- I0: LD R0, a[idx];
- I1: LD R1, b[idx];
- I2: MPY R2,R0,R1

clock cycles:

C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17 C18 ...

warps

W0:	I0	I1																
W1:			I0	I1														
W2:																		
W3:																		
W4:																		
W5:																		
W6:																		
W7:																		
W8:																		
W9:																		

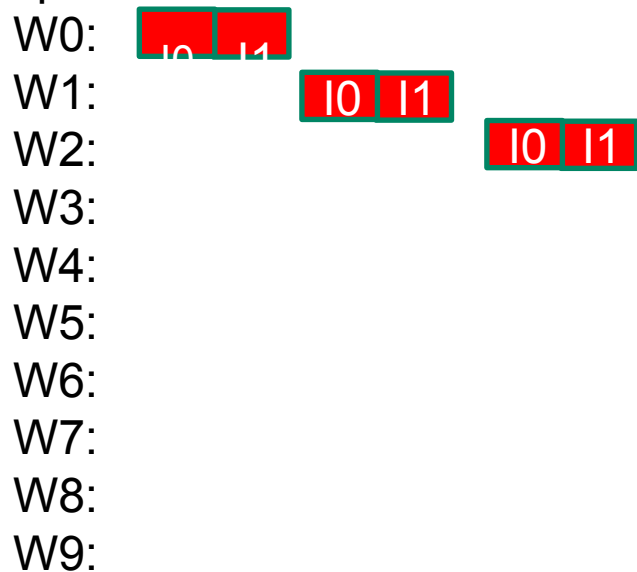
...

- I0: LD R0, a[idx];
- I1: LD R1, b[idx];
- I2: MPY R2,R0,R1

clock cycles:

C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17 C18 ...

warps



Warps scheduled by a thread scheduler

clock cycles:

C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17 C18 ...

warps

W0: 

W1: 

W2: 

W3: 

W4: 

W5: 

W6: 

W7: 

W8:

W9:

...

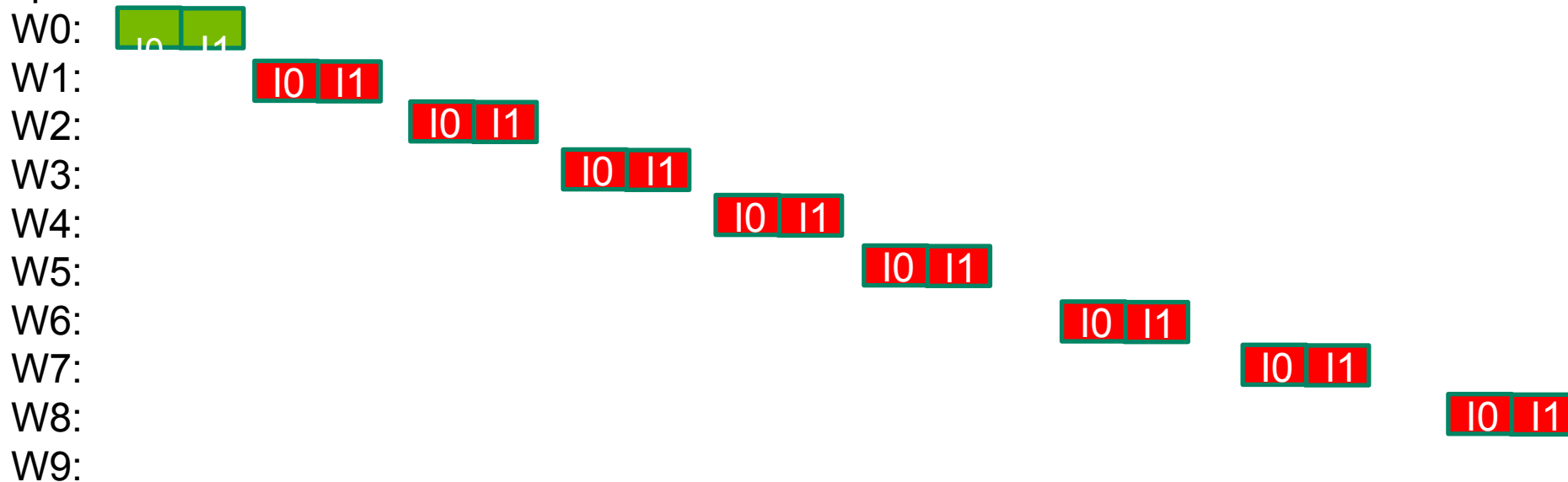
- I0: LD R0, a[idx];
- I1: LD R1, b[idx];
- I2: MPY R2,R0,R1

- I0: LD R0, a[idx];
- I1: LD R1, b[idx];
- I2: MPY R2,R0,R1

clock cycles:

C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17 C18 ...

warps

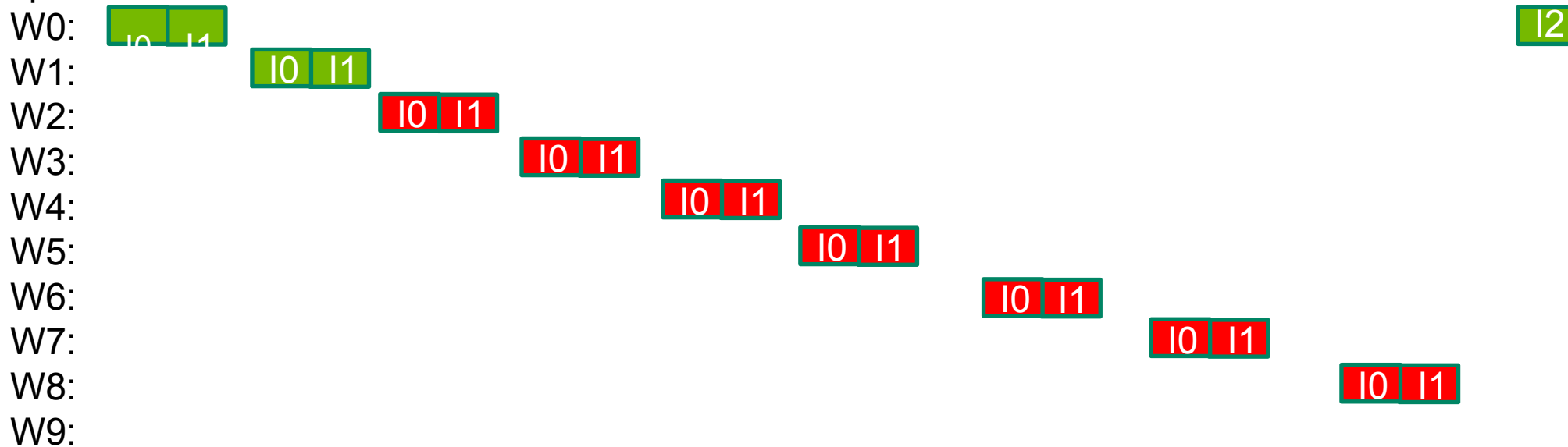


- I0: LD R0, a[idx];
- I1: LD R1, b[idx];
- I2: MPY R2,R0,R1

clock cycles:

C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17 C18 ...

warps

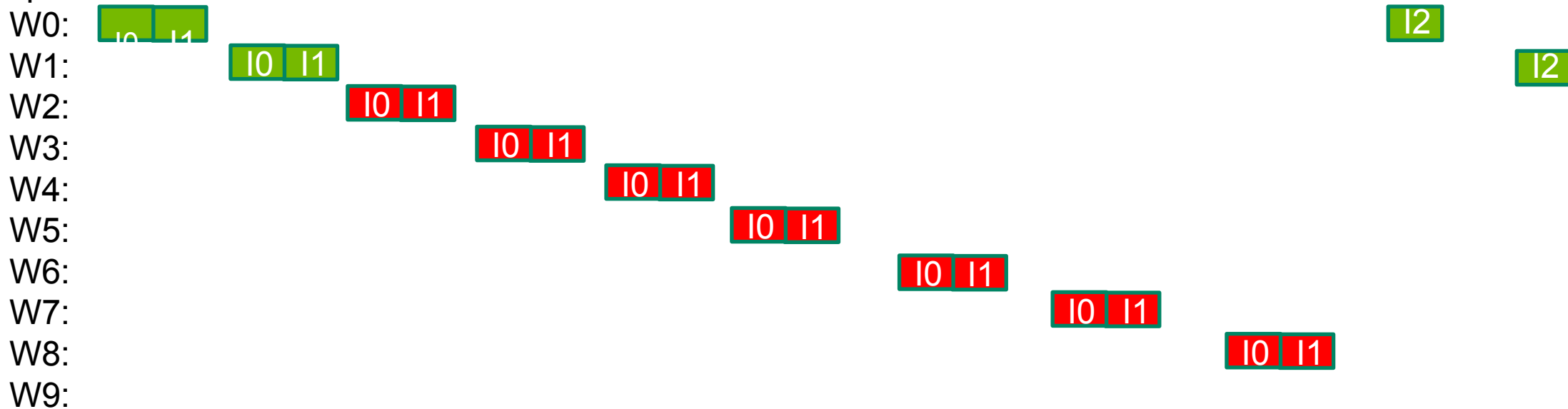


- I0: LD R0, a[idx];
- I1: LD R1, b[idx];
- I2: MPY R2,R0,R1

clock cycles:

C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17 C18 ...

warps

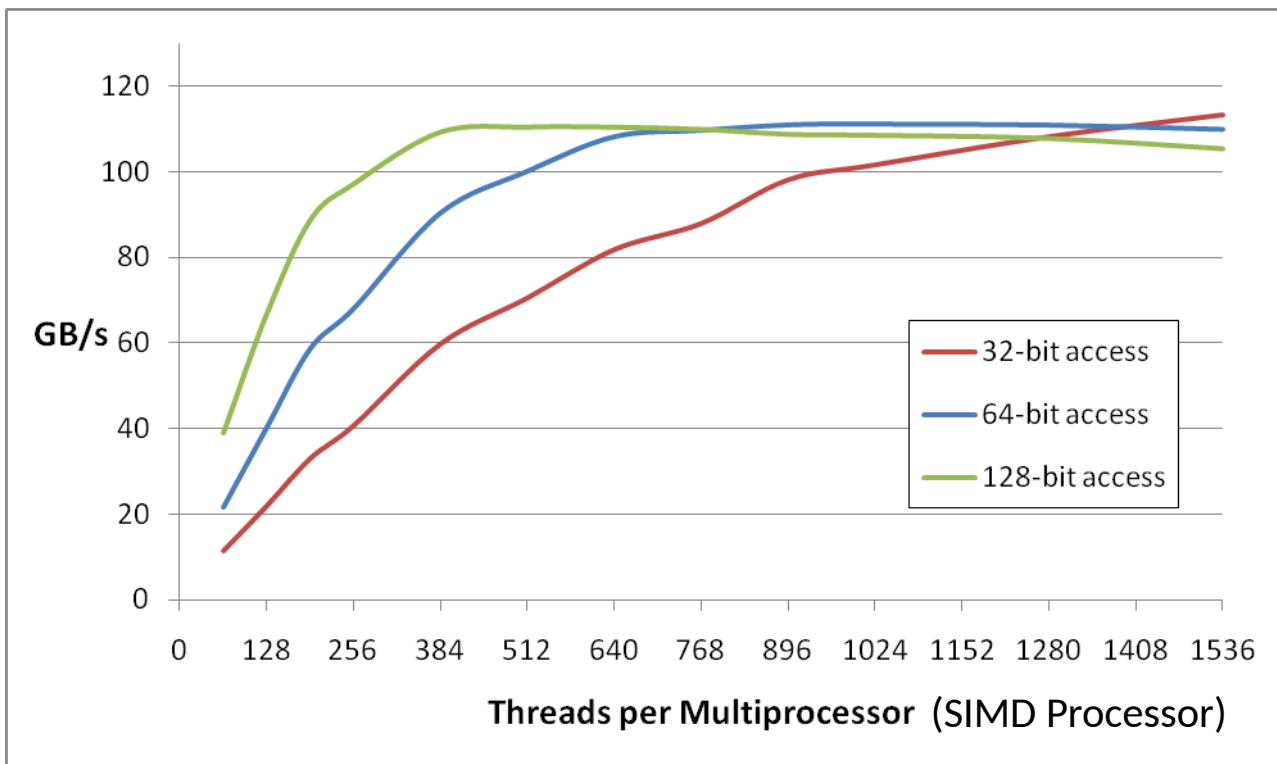


- Hiding arithmetic latency:
 - Need ~e.g., more than 10's warps (~320 CUDA threads, or 10 SIMD threads) per SIMD processor
 - Or, latency can also be hidden with independent instructions from the same warp
 - ->if instructions never depends on the output of preceding instruction, then only 5 warps are needed, etc.
- Maximizing global memory throughput:
 - Depends on the access pattern, and word size
 - Need enough memory transactions in flight to saturate the bus
 - Independent loads and stores from the same warp
 - Loads and stores from different warps
 - Larger word sizes can also help (float2 is twice the transactions of float, for example)

Increment of an array of 64M elements

Two accesses per thread (load then store) - dependent, so really 1 access per thread at a time

theoretical bandwidth: ~120 GB/s



Several independent smaller accesses have the same effect as one larger one.

For example:

Four 32-bit ~= one 128-bit

- Need enough total threads to keep GPU busy
 - Typically, you'd like **512+ CUDA threads** per SIMD Processor (aim for 2048 - maximum "occupancy")
 - More if processing one fp32 element per thread
 - Of course, exceptions exist
- Threadblock configuration
 - Threads per block should be a **multiple of warp size (32)**
 - SIMD Processor can concurrently execute **at least 16** thread blocks (Maxwell/Pascal/Volta/Ampere: 32)
 - Really small thread blocks prevent achieving good occupancy
 - Really large thread blocks are less flexible
 - Could generally use **128-256 threads/block**, but use whatever is best for the application

- A measure of the actual thread load in an SIMD Processor, vs. peak theoretical or peak achievable
- CUDA includes an occupancy calculator spreadsheet
- Achievable occupancy is affected by limiters to occupancy
- Primary limiters:
 - Registers per thread (can be reported by the profiler, or can get at compile time)
 - Threads per threadblock
 - Shared memory usage

- GPU is a massively thread-parallel, latency hiding machine
- Kernel Launch Configuration:
 - Launch enough threads per SM to hide latency
 - Launch enough threadblocks to load the GPU
- Use analysis/profiling when optimizing:
 - “Analysis-driven Optimization” (future session)
 - -> Nsight Compute can show you information about whether you’ve saturated the compute subsystem or the memory subsystem.

- Fundamental Optimization, Part 2
- Atomics, Reductions, Warp
- Shuffle Using Managed Memory
- Concurrency (streams, copy/compute overlap, multi-
- GPU) Analysis Driven Optimization
- Cooperative Groups

- Optimization in-depth:
 - <http://on-demand.gputechconf.com/gtc/2013/presentations/S3466-Programming-Guidelines-GPU-Architecture.pdf>
- Analysis-Driven Optimization:
 - <http://on-demand.gputechconf.com/gtc/2012/presentations/S0514-GTC2012-GPU-Performance-Analysis.pdf>
- CUDA Best Practices Guide:
 - <https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html>
- CUDA Tuning Guides:
 - <https://docs.nvidia.com/cuda/index.html#programming-guides>

(Kepler/Maxwell/Pascal/Volta)