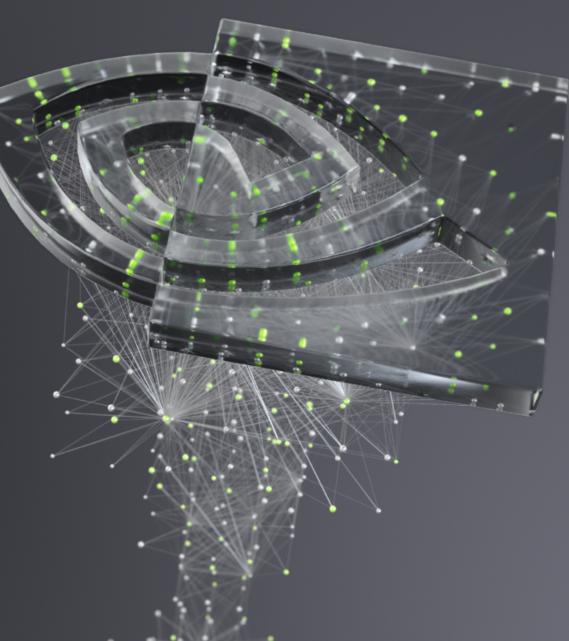


CUDA OPTIMIZATION, PART 1 NVIDIA Corporation



OUTLINE

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

KEPLER CC 3.5 SM (GK110)

- "SMX" (enhanced SM)
- 192 SP units ("cores")
- 64 DP units
- LD/ST units, 64K registers
- 4 warp schedulers
- Each warp scheduler is dual-issue capable
- ► K20: 13 SMX's, 5GB
- K20X: 14 SMX's, 6GB
- K40: 15 SMX's, 12GB

SMX Instruction Cache																			
Warp Scheduler					Warp Scheduler				Warp Scheduler					Warp Scheduler					
Di	Dispatch Dispatch			Dispatch Dispatch			tch	Dispatch Dispatch			Dispatch Dispatch				ch				
	*		+		Register File (6				↓ ↓					+ +					
4	+	+				+	- L			JJ,33			•	+	+	+		L	+
Core		Core			Core		DP Unit	LD/ST	SFU	Core	Core						DP Unit	LD/ST	
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU	Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
Core	Core	Core	DP Unit	Core	Core	Core	DP Unit	LD/ST					DP Unit	Core	Core	Core	DP Unit	LD/ST	SFU
-										ct Ne									
_				_		_	64 KB	Share B Rea											
	Tex		Tex			Tex		Тех	_	_	ata C Tex		Tex			Tex	_	Tex	
	Tex		Tex			Tex		Tex			Tex		Тех			Tex		Tex	

🕺 NVIDIA.

MAXWELL/PASCAL CC5.2, CC6.1 SM

- "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

SMM														
PolyMorph Engine 3.0 Vertex Fetch Tassellator Viewport Transform														
Attribute Setup Stream Output														
Instruction Cache														
		nstructi	on Buffe	r] [Instruction Buffer							
	ispatch Uni	Warp Si It	:heduler D	(spatch Un	vit			ispatch Uni	heduler D	Dispatch Unit				
	Regist	er File ('	16,384 x	4 32-bit)		11	Register File (16,384 x 32-bit)							
Core	Core	Core	Core	LDIST	SFU		Core	Core	Core	Core	LDIST	SFU		
Core	Core	Core	Core	LDIST	SFU		Core	Core	Core	Core	LDIST	SFU		
Core	Core	Core	Core	LDIST	SFU		Core	Core	Core	Core	LDIST	SFU		
Core	Core	Core	Core	LDIST	SFU		Core	Core	Core	Core	LDIST	SFU		
Core	Core	Core	Core	LDIST	SFU		Core	Core	Core	Core	LDIST	SFU		
Core	Core	Core	Core	LDIST	SFU		Core	Core	Core	Core	LDIST	SFU		
Core	Core	Core	Core	LDIST	SFU		Core	Core	Core	Core	LD/ST	SFU		
Core	Core	Core	Core	LDIST	SFU		Core	Core	Core	Core	LDIST	SFU		
Texture / L1 Cache														
					Texture	/ L	1 Cache							
	Tex			Tex	Texture	I	1 Cache	Tex			Tex			
		nstructi	on Buffe		Texture		1 Cache		nstructi	on Buffe				
		Warp S	on Buffe cheduler				1 Cache	ispatch Un	Warp S	on Buffe cheduler		4		
	ispatch Un	Warp S	on Buffe cheduler 16,384 x	Napatch Ur			1 Cache	ispetch Un	Warp Si	on Buffe cheduler 16,384 x	M Dispatch Ur	**		
Core	ispatch Un	Warp S	cheduler C	Napatch Ur			Core	ispetch Un	Warp Si	cheduler C	M Dispatch Ur	et and a set of the se		
	Regist	Warp S H Ier File (cheduler C 16,384 x	Nigatch Ur 32-bit)	nit		-	Regist	Warp Si it er File (cheduler 16,384 x	Napatch Ur 1 32-bit)			
Core	Regist	Warp S # er File (Core	thedulor 16,384 x Core	Nispatch Ur 32-bit) LDIST	nit		Core	Regist	Warp Si it er File (Core	cheduler c 16,384 x Core	ingerich Ur 32-bit) LDIST	SFU		
Core Core	Regist Core Core	Warp S er File (Core Core	16,384 x Core Core	r Napatch Ur 32-bit) LDIST LDIST	nit SFU SFU		Core	Regist Core Core	Warp So tr Core Core	16,384 x Core Core	er Dispatch Uk 32-bit) LDIST LDIST	SFU SFU		
Core Core Core	Regist Core Core Core	Warp S er File (Core Core Core	Core Core Core	a Sapatch UV 32-bit) LDIST LDIST LDIST	SFU SFU SFU		Core Core	Rogist Core Core Core	Warp Si it er File (Core Core Core	Core Core Core	ar Dispatch (V 32-bit) LOIST LOIST LOIST	SFU SFU SFU		
Core Core Core Core	Regist Core Core Core Core Core	Warp S At Core Core Core Core Core	Chedular Core Core Core Core Core	Aligentich UK 32-bit) LDIST LDIST LDIST	SFU SFU SFU SFU		Core Core Core Core	Regist Core Core Core Core Core	Warp So at Core Core Core Core	Core Core Core Core Core	Aligeatch Uk 32-bit) LOIST LOIST LOIST	SFU SFU SFU SFU		
Core Core Core Core Core	Regist Core Core Core Core Core Core	Warp S et Core Core Core Core Core Core	Core Core Core Core Core Core Core	Angestch Uv 32-bit) LDIST LDIST LDIST LDIST	SFU SFU SFU SFU SFU		Core Core Core Core Core	Regist Core Core Core Core Core Core	Warp Si t Core Core Core Core Core Core Core	Core Core Core Core Core Core Core	Angestich Us 32-bit) LOIST LOIST LOIST LOIST	SFU SFU SFU SFU SFU		
Core Core Core Core Core	Regist Core Core Core Core Core Core	Warp S er File (Core Core Core Core Core Core	Core Core Core Core Core Core Core	ar Signatch Ur 32-bit) LDIST LDIST LDIST LDIST	SFU SFU SFU SFU SFU SFU		Core Core Core Core Core Core	Regist Core Core Core Core Core Core	Warp Si it Core Core Core Core Core Core Core	Core Core Core Core Core Core Core	Conservation Units 32-bit() LossT LossT LossT LossT LossT LossT	SFU SFU SFU SFU SFU		
Core Core Core Core Core Core	Regist Core Core Core Core Core Core Core	Warp S er File (Core Core Core Core Core Core Core	Core Core Core Core Core Core Core Core	Ringestch UN 32-bit) LDIST LDIST LDIST LDIST LDIST LDIST	8FU 8FU 8FU 8FU 8FU 8FU 8FU 8FU 8FU		Core Core Core Core Core Core Core	Regist Regist Core Core Core Core Core Core Core	Warp Si Triangle Si Core Core Core Core Core Core Core Core	Core Core Core Core Core Core Core Core	Auguston UX 32-bit) LOIST LOIST LOIST LOIST LOIST LOIST	SFU SFU SFU SFU SFU SFU		
Core Core Core Core Core Core	Regist Core Core Core Core Core Core Core	Warp S er File (Core Core Core Core Core Core Core	Core Core Core Core Core Core Core Core	Ringestch UN 32-bit) LDIST LDIST LDIST LDIST LDIST LDIST	8FU 8FU 8FU 8FU 8FU 8FU 8FU 8FU 8FU		Core Core Core Core Core Core Core	Regist Regist Core Core Core Core Core Core Core	Warp Si Triangle Si Core Core Core Core Core Core Core Core	Core Core Core Core Core Core Core Core	Auguston UX 32-bit) LOIST LOIST LOIST LOIST LOIST LOIST	SFU SFU SFU SFU SFU SFU		

PASCAL/VOLTA CC6.0/7.0

- 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

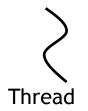
							L1 Instru	tion Cache							_	
			nstruct					L0 Instruction Cache								
		-			hread/c			Warp Scheduler (32 thread/clk)								
	Di	spatcl	n Unit	(32 th	read/cli	k)		Dispatch Unit (32 thread/clk)								
		Register File (16,384 x 32-bit)														
FP64	INT	INT	FP32	FP32				FP64	INT	INT	FP32	FP32	H			
FP64	INT	INT	FP32	FP32	++			FP64	INT	INT INT	FP32	FP32	\vdash			
FP64	INT	INT	FP32	FP32	TENSOR		FP64	INT	INT	FP32	FP32	TENSOR		TENSOR		
FP64	INT	INT	FP32	FP32		TENSOR	FP64	INT	INT	FP32	FP32					
FP64	INT	INT	FP32	FP32	COF		CORE	FP64	INT	INT	FP32	FP32		ORE	CORE	
FP64	INT	INT	FP32	FP32	\square	+		FP64	INT	INT	FP32	FP32	\vdash			
FP64	INT	INT	FP32	FP32				FP64	INT	INT	FP32	FP32				
FP64	INT	INT	FP32	FP32	++	\pm		FP64	INT	INT	FP32	FP32	\vdash			
LD/ LD/ ST ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	SFU	LD/ LD/ ST ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	LD/ ST	SFU	
	_		Istruci	aon C	ache					L0 h	nstruc	tion C	ache			
		p Sch	edule	r (32 tl	hread/c	_				rp Sch	nedule	r (32 t	hread			
	Di	r <mark>p Sch</mark> spatcl	edulei 1 Unit	r (32 tl (32 th		k)	-		Di	rp Sch spatcl		r (32 t (32 th	hread read/o	cik)		
FP64	Di	r <mark>p Sch</mark> spatcl	edulei 1 Unit	r (32 tl (32 th 16,384	hread/c read/cli	k)		FP64	Di	rp Sch spatcl	edule h Unit File ('	r (32 t (32 th 16,384	hread read/o	cik)		
FP64 FP64	Di: Reg	rp Sch spatcl ister	eduler h Unit File (1	r (32 th (32 th 16,384 FP32	hread/c read/cli	k)		FP64 FP64	Di Reg	rp Sch spatci jister	edule h Unit File (' FP32	r (32 t (32 th 16,384	hread read/o	cik)		
	Di: Reg INT	p Sch spatch ister INT	eduler 1 Unit (File (1 FP32	r (32 th (32 th 16,384 FP32 FP32	hread/c read/cli	k)			Di Reg INT	rp Sch spatc jister INT	File (* FP32	r (32 t (32 th 16,38 FP32	hread read/o	cik)		
FP64	Di Reg INT INT	rp Sch spatch ister INT INT	edulen 1 Unit File (1 FP32 FP32	(32 th (32 th (32 th (32 th (32 th (32 th) (32	hread/c read/cli	k) bit)	TENSOR	FP64	Di Reg INT INT	rp Sch spatc ister INT INT	File (FP32 FP32 FP32 FP32	r (32 th (32 th 16,384 FP32 FP32	hread read/c 4 x 32	cik)	TENSOR	
FP64 FP64	Dis Reg INT INT INT	rp Sch spatch ister INT INT INT	edulen Unit File (1 FP32 FP32 FP32	r (32 th (32 th 6,384 FP32 FP32 FP32 FP32	hread/cl read/cl 4 x 32-	k) bit) SOR	TENSOR	FP64 FP64	Di Reg INT INT INT	rp Sch spatc jister INT INT INT	File (* File (* FP32 FP32 FP32 FP32	r (32 th (32 th 16,38 FP32 FP32 FP32	hread read/o 4 x 32 TEN	cik) 2-bit)	TENSOR	
FP64 FP64 FP64	Dis Reg INT INT INT INT	p Sch spatch ister INT INT INT	edulen Dunit File (1 FP32 FP32 FP32 FP32	(32 th (32 th 6,384 FP32 FP32 FP32 FP32 FP32	hread/cl read/cl 4 x 32-l TENS	k) bit) SOR		FP64 FP64 FP64	Di Reg INT INT INT INT	rp Sch spatc ister INT INT INT INT	File (* File (* FP32 FP32 FP32 FP32	r (32 t (32 th 16,38 FP32 FP32 FP32 FP32	hread read/o 4 x 32 TEN	2-bit)		
FP64 FP64 FP64 FP64	Reg INT INT INT INT INT	p Sch spatch ister INT INT INT INT	edulei 1 Unit File (1 FP32 FP32 FP32 FP32 FP32	(32 th (32 th (32 th 6,384 FP32 FP32 FP32 FP32 FP32 FP32	hread/cl read/cl 4 x 32-l TENS	k) bit) SOR		FP64 FP64 FP64 FP64	Di Reg INT INT INT INT	INT INT INT INT INT INT	File (* FP32 FP32 FP32 FP32 FP32 FP32 FP32	r (32 th (32 th 16,384 FP32 FP32 FP32 FP32 FP32	hread read/o 4 x 32 TEN	2-bit)		
FP64 FP64 FP64 FP64 FP64 FP64 FP64	Dis Reg INT INT INT INT INT INT	P Sch spatcl ister INT INT INT INT INT INT	edulei 1 Unit File (1 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	r (32 th (32 th 6,384 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	tread/cl	k) bit) GOR		FP64 FP64 FP64 FP64 FP64 FP64 FP64	Di Reg INT INT INT INT INT INT	INT INT INT INT INT INT INT INT INT	File (FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	r (32 th (32 th 16,384 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	hread read/o 4 x 32	2-bit) SORE		
FP64 FP64 FP64 FP64 FP64 FP64	Difference	p Sch spatcl ister INT INT INT INT INT	edulei 1 Unit (1 FP32 FP32 FP32 FP32 FP32 FP32 FP32	r (32 th (32 th 16,384 FP32 FP32 FP32 FP32 FP32 FP32 FP32	tread/cl	k) bit) SOR		FP64 FP64 FP64 FP64 FP64 FP64	Di Reg INT INT INT INT INT	INT INT INT INT INT INT INT INT	File (FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	r (32 t (32 th 16,384 FP32 FP32 FP32 FP32 FP32 FP32 FP32	hread read/o 4 x 32 TEN	2-bit)		
FP64 FP64 FP64 FP64 FP64 FP64 FP64 FP64	Dis Reg INT INT INT INT INT INT INT LD/	p Sch spatcl ister INT INT INT INT INT INT INT	edulei h Unit File (1 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32 LD	r (32 th (32 th 16,384 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	hread/cl read/cl 4 x 32-	k) bit) SOR RE	CORE	FP64 FP64 FP64 FP64 FP64 FP64 FP64 LD/LD/LD/	Di Reg INT INT INT INT INT INT INT LD/ ST	rp Sch spatcl spatcl ister INT INT INT INT INT INT INT INT INT	File (FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	r (32 t (32 th 16,384 FP32 FP32 FP32 FP32 FP32 FP32 FP32 FP32	tread 4 x 32	cik) 2-bit) SSOR DRE	CORE	

🕺 NVIDIA.

EXECUTION MODEL

Software

Hardware



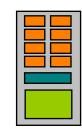


Threads are executed by scalar processors

Thread blocks are executed on multiprocessors



Thread Block



Multiprocessor

Thread blocks do not migrate

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



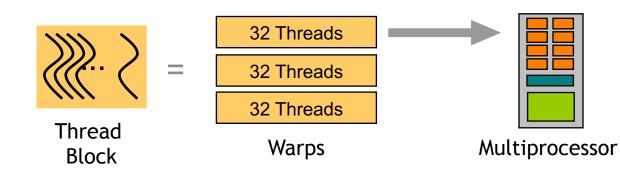




Devi	ce	

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 (SIMD) on a multiprocessor



LAUNCH CONFIGURATION

LAUNCH CONFIGURATION

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

GPU LATENCY HIDING

- 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

- 10: LD R0, a[idx];
- I1: LD R1, b[idx];
- 12: 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:

. . .



IO: LD R0, a[idx];

•I1: LD R1, b[idx];

I2: MPY R2,R0,R1



○IO: LD RO, a[idx];

●I1: LD R1, b[idx];

○I2: MPY R2,R0,R1



○IO: LD RO, a[idx];

●I1: LD R1, b[idx];

○I2: MPY R2,R0,R1



○IO: LD RO, a[idx];

●I1: LD R1, b[idx];

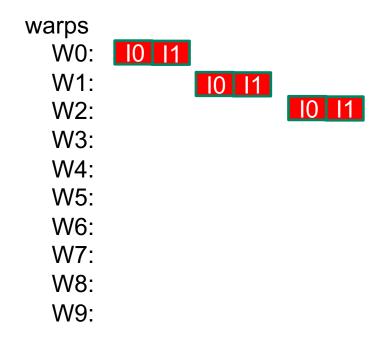
○I2: MPY R2,R0,R1



IO: LD R0, a[idx];

I1: LD R1, b[idx];

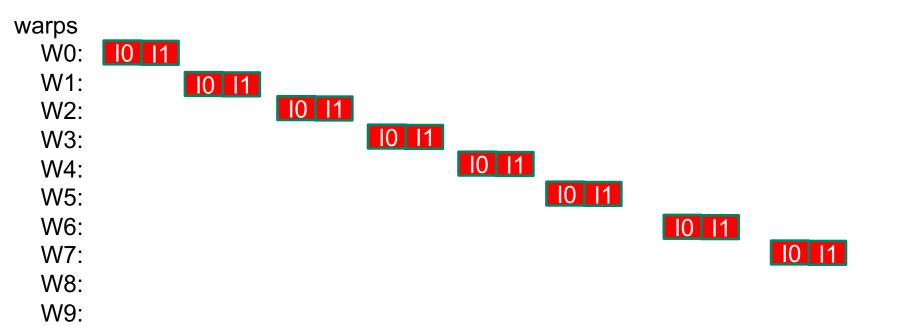
I2: MPY R2,R0,R1



○IO: LD RO, a[idx];

●I1: LD R1, b[idx];

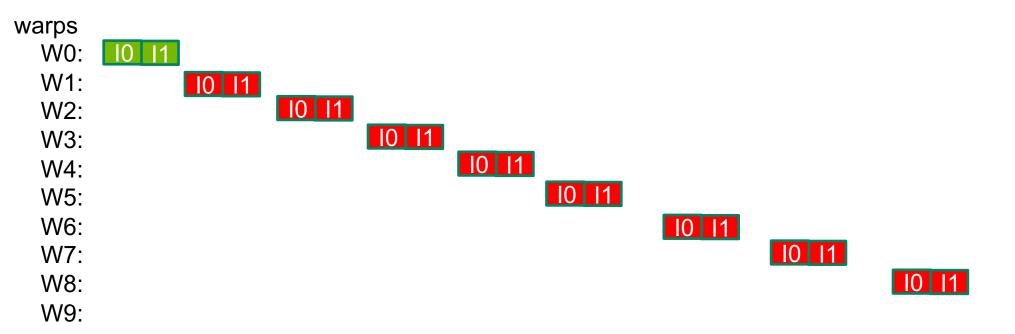
○I2: MPY R2,R0,R1



○IO: LD RO, a[idx];

●I1: LD R1, b[idx];

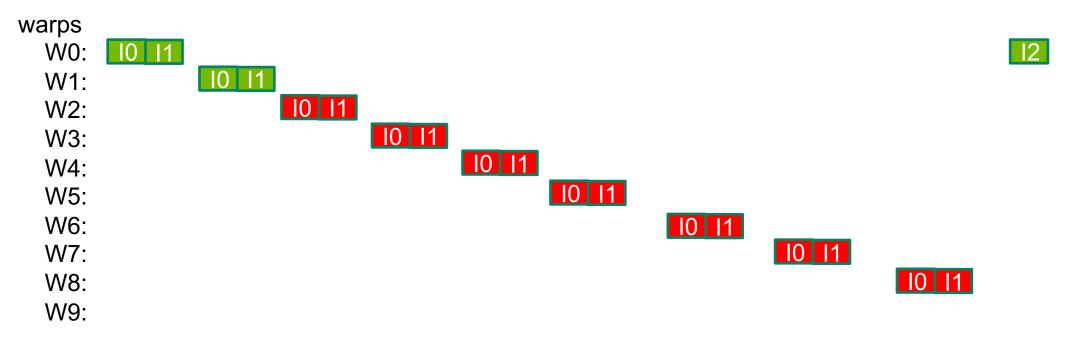
○I2: MPY R2,R0,R1



○IO: LD RO, a[idx];

●I1: LD R1, b[idx];

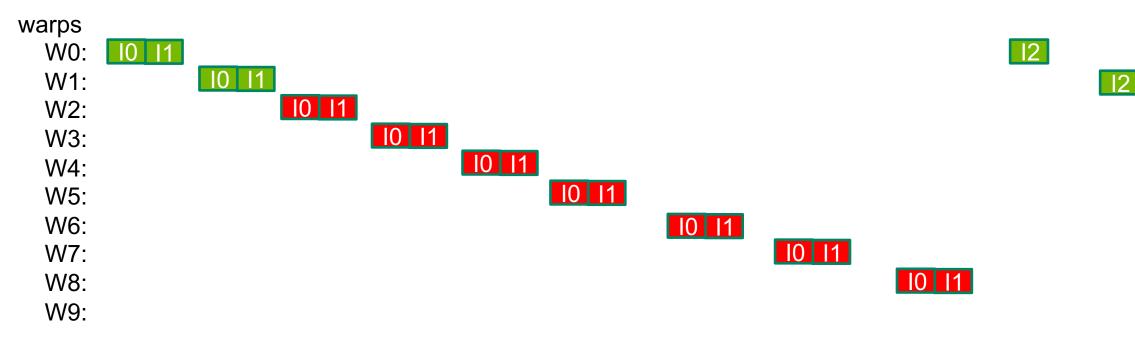
○I2: MPY R2,R0,R1



○IO: LD RO, a[idx];

●I1: LD R1, b[idx];

○I2: MPY R2,R0,R1



LAUNCH CONFIGURATION

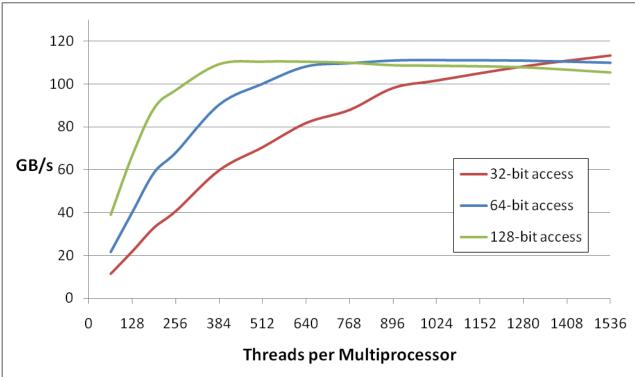
- Hiding arithmetic latency:
 - Need ~10's warps (~320 threads) per SM
 - Or, latency can also be hidden with independent instructions from the same warp
 - ->if instructions never depends on the output of preceding instruction, then only 5 warps are needed, etc.
- Maximizing global memory throughput:
 - 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 thread
 - Loads and stores from different threads
 - Larger word sizes can also help (float2 is twice the transactions of float, for example)

MAXIMIZING MEMORY THROUGHPUT

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

LAUNCH CONFIGURATION: SUMMARY

- Need enough total threads to keep GPU busy
 - Typically, you'd like 512+ threads per SM (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)
 - SM can concurrently execute at least 16 thread blocks (Maxwell/Pascal/Volta: 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

ASIDE: WHAT IS OCCUPANCY?

- A measure of the actual thread load in an SM, vs. peak theoretical/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

SUMMARY

- 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.

FUTURE SESSIONS

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

FURTHER STUDY

- 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)

HOMEWORK

- Log into Summit (ssh <u>username@home.ccs.ornl.gov</u> -> ssh summit)
- Clone GitHub repository:
 - Git clone git@github.com:olcf/cuda-training-series.git
- Follow the instructions in the readme.md file:
 - https://github.com/olcf/cuda-training-series/blob/master/exercises/hw3/readme.md

Prerequisites: basic linux skills, e.g. ls, cd, etc., knowledge of a text editor like vi/emacs, and some knowledge of C/C++ programming





